The impact of uTP on BitTorrent completion time

Claudio Testa and Dario Rossi
Telecom ParisTech, Paris, France, firstname.lastname@telecom-paristech.fr

***Abstract*—BitTorrent, one of the most widespread file sharing P2P applications, recently introduced uTP, an application-level congestion control protocol which aims to efficiently use the available link capacity, while avoiding to interfere with the rest of user traffic (e.g., Web, VoIP and gaming) sharing the same access bottleneck.**

Research on uTP has so far focused on the investigation of the congestion control behavior on rather simple settings (i.e., single bottleneck, few backlogged flows, etc.), that are fairly far from the P2P settings in which the protocol is deployed. Moreover, prior work typically addressed questions, such as fairness and efficiency, that are natural from a congestion control context perspective, but are not directly related with the performance of the overall P2P system.

In this work, we refine the understanding of uTP, by gauging its impact on the primary BitTorrent user-centric metric, namely the torrent download time, by means of packet level simulation. Results of our initial investigations show that: (i) in case uTP clients fully substitute TCP clients, no performance difference arise; (ii) in case of heterogeneous swarms, comprising peers using uTP and TCP congestion control, completion time of uTP peers can possibly benefit of lower uplink queuing delays, as signaling traffic (e.g., chunk requests) are not slowed down by long waits in the ADSL buffers.

I. Introduction

In the latest years, BitTorrent represented the closest synonym to *filesharing*, before drops in the capacity and storage costs witnessed a widespread of several filehosting servers around the Internet. Yet, BitTorrent still represents the closest synonym to *peer-to-peer filesharing*, with a very large ecosystem consisting of hundreds of trackers and millions of concurrently active peers [1]. Recent changes in μTorrent, a popular implementation of the BitTorrent protocol, include uTP [2], a new congestion control paradigm for data transfer and, more recently, the ability to perform sequential download, i.e., to *"start watching before [...] download completes, or simply preview content before committing to a full download"* [3].

These changes have refueled the interest of the scientific community [1], [4]–[8] on the application, with a recent body of work focusing on uTP [6]–[8] (while no work focused so far on μTorrent streaming capabilities which have been released on 19th April, 2011). The novel congestion control algorithm, implemented at the application layer over an UDP framing, is designed to fully exploit the access capacity while introducing only a *small amount of extra queuing delay* on the ADSL buffer. Indeed, as ADSL modems can store large quantities of data (up to some seconds [7]), the use of TCP for transferring large files can harm the performance of other interactive traffic such as VoIP, gaming and Web browsing. Shortly, as TCP congestion control is based on *the inference of losses* (via duplicate acknowledgement or timer expiration), this entails that TCP necessarily fills the access buffer, hence perturbing the other traffic of the very same user, with consequently low quality of service for interactive applications.

To solve this QoS issue, there are two general, orthogonal means. On the one hand, we have active queue management techniques, that are becoming commonplace in recent home gateways [7], and can be used to prioritize user traffic. On the other hand, we have distributed congestion control techniques, such as the one proposed by uTP, that aim at realizing efficient but low-priority transport services. The main aim of uTP is perhaps better understood in BitTorrent words *"The main purpose of uTP is not to increase transfer speeds, it's to fix the problem of delay. [...] That said, the fact that uTP doesn't add significant delay also means that utilization (and speed) can be increased."* [9]. Also, it is clearly understood that users will be the ultimate judge of BitTorrent performance, as *"unless we can offer the same performance [of TCP], then people will switch to a different BitTorrent client."* [2].

In this work, we dig precisely into this question, by performing an initial but careful investigation of the primary swarm performance metric from the user perspective, i.e., the torrent completion time, via `ns2` simulation, exploiting and integrating the open-source BitTorrent `ns2` module [5] with our open-source uTP implementation for `ns2` [8].

II. Related work

BitTorrent is not only a successful application, but also a widely studied research topic [1], [4]–[8]. While early study on BitTorrent focused on the evaluation of its performance by means of fluid model [4], simulations studies [5] or analysis on real swarms [1], a more recent research body focus on the uTP protocol [6]–[8], and is thus closer to ours. At the same time, we point out that while [6]–[8] all focus on uTP, none of the above shares the same aim of this work. More precisely, [6], address the queuing delay estimation by solving problems tied to the clock drift. Authors in [7] study uTP in a local testbed, employing different real ADSL modems. In [8] we instead implement uTP in `ns2`, and then study the congestion control protocol in simple settings (e.g., a couple of flows sharing a bottleneck link), focusing especially on the fairness of the transfer, but from individual flows perspective.

In this work, we instead argue that a more pertinent view, at least from the user standpoint, is to consider the application performance as a whole, irrespectively of the performance of individual flows. Therefore, we focus on the torrent completion time, the primary user-centric performance metric in BitTorrent systems, that to the best of our knowledge has been

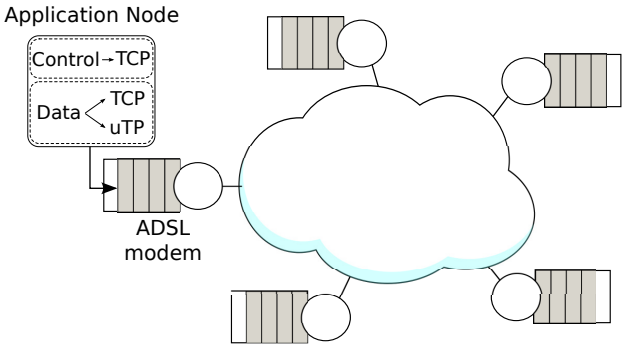


Fig. 1. Synoptic of simulation model.

neglected so far. Due to space limitation, we refer the reader to [10] or the above work for an overview of the uTP protocol.

III. Methodology and Scenario

To carry out our study, we resort as previously stated to the open-source implementations of BitTorrent [5] and uTP [8] for `ns2`. Nevertheless some modifications were necessary in order to successfully integrate both modules, which we shortly summarize in the following with the help of Fig. 1.

As commonly assumed, we consider the bottleneck to be represented by the access link, which is also one of the main motivations of uTP. As access links technology, we consider ADSL-like access with $C = 1$ Mbps uplink capacity (homogeneous for the whole peer population) and 8 Mbps down link capacity. Coherently with [7], unless otherwise stated, we set the buffer size to $B = 200$ full-size packets (corresponding to a few seconds given C). Access links also model propagation delay, which is chosen uniformly at random in the interval $[1, 25]$ so that the average one way delay of the swarm is about $D = 25$ ms (i.e., as two access links are crossed in the end-to-end path connecting any two peers), which is the default value of uTP target delay [8]. Access nodes are then interconnected by the Internet, that we model by means of infinite capacity, null delay links connected through an (infinite switching capacity) router, to which all ADSL modems are interconnected in a star topology configuration.

We implement two kind of nodes, that model different application settings as far as the congestion protocol used for *data transfer* is concerned. We assume that the latest application versions will by default initiate data transfers using uTP on their uplink, but that they can accept incoming TCP data connections (i.e., equivalent to setting `bt.transp_disposition=5` in μTorrent); similarly, older versions initiate TCP transfers, but accept incoming uTP connections anyway (i.e., `bt.transp_disposition=10`). We further model that, irrespectively of their data transfer settings, applications will use TCP for the exchange of *control messages*. Hence, a different traffic mixture will possibly compete for ADSL access link capacity and buffer space. Notice that, while nodes are free to decide what congestion control flavor use for the outbound connections, they have to comply to other peers settings as far as inbound data connections are concerned.

We simulate a (mild) flash-crowd scenario, in which at time t=0 the swarm is constituted by only one seed, and then 100 leechers join with exponentially distributed arrival times (mean rate 0.1 Hz). During each simulation, we discard the first 50 completion samples that happen during the transient period (more details later on), and consider only the subsequent 50 completion times. For each parameter settings, we repeat each simulation 10 times, so that statistics represent 500 individual torrent downloads per setting.

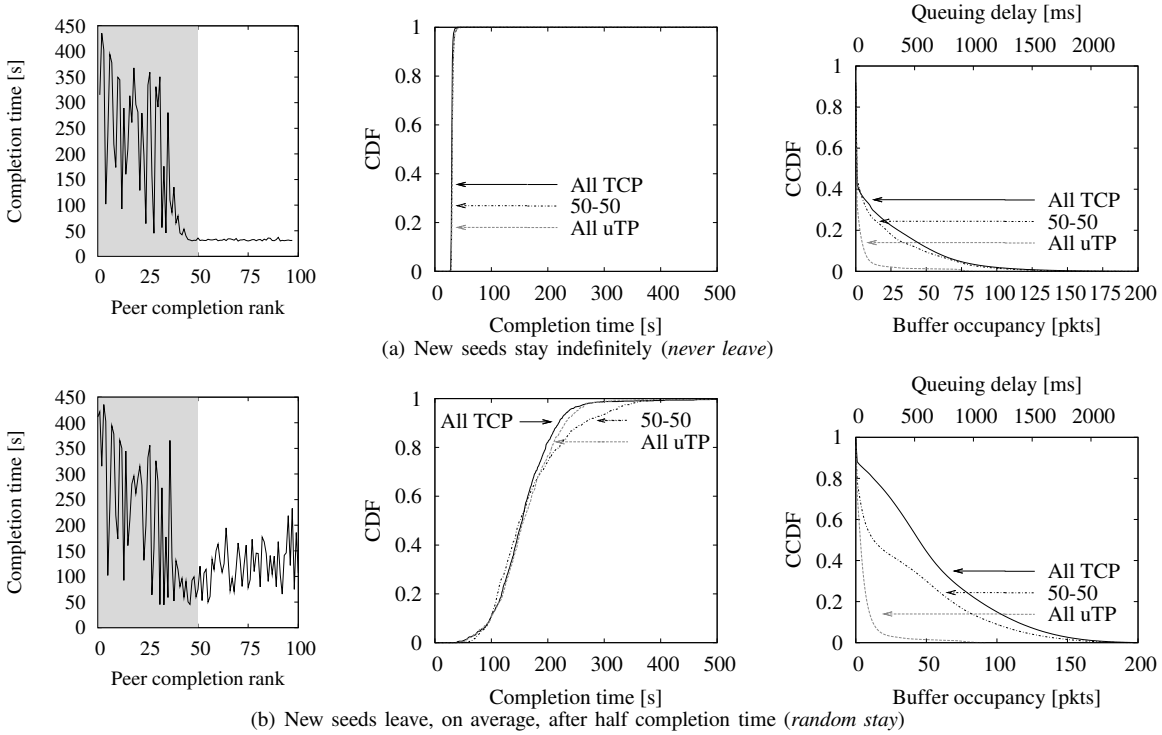
As far as the swarm population is concerned, we either consider *homogeneous swarms* (i.e., all TCP or all uTP) or *heterogeneous swarms* where the population is equally split, on average, among uTP and TCP peers (and that we denote in the following as 50-50). While we reckon that it would be interesting to explore different TCP vs uTP ratios, due to space limitation we believe this split ratio to be reasonable: while μTorrent client is among the most popular clients, there is no precise agreement on its popularity (e.g., μTorrent estimated share varies between 15% in 2006 to 60% in 2008 [1]), meaning that TCP versions are still around; moreover, as population sets size are not biased, the analysis of the results is simpler.

As far as the peer behavior is concerned, we again consider two scenarios. A first, optimistic one, where peers *never leave* the system after becoming seeds, and a second, more realistic one, where newborn seeds stay in the system for a *random time*. In this latter scenario, each time a peer leaves, a new leecher joins the swarm, so that the swarm size remains constant. Specifically, peers stay in the system after becoming seeds for an exponentially distributed time, with mean equal to half their download time (this choice roughly models the BitTorrent netiquette, i.e., to continue seeding until the uploaded data reaches 1.5 times the downloaded amount). We point out that, due to space limitation, we avoid reporting results for a worst-case scenario in which peers leave immediately as soon as they become seeds, as the above two scenarios already provide interesting insights.

IV. Simulation Results

Simulation results are reported in Fig. 2. Top plots refer to scenarios where seeds never leave the system, while bottom plots to random stay scenarios. Taking a single run as an example, plots on the left depict how the completion time evolves during the simulation, ordered by peer completion rank (dark gray background represents the discarded initial transient period). If seeds stay forever, completion times shrink down to a point in which leechers are close to fully exploit their downlink capacity, which is no longer the case when seeds stay only for a finite time.

Middle Fig. 2 plots report the completion time cumulative distribution (CDF) for the different populations. If peers never leave the system, no difference arises due to the congestion control algorithm: intuitively, as there is no resource hotspot, peers are able to download from many seeds at the same time,



(a) New seeds stay indefinitely (*never leave*)

(b) New seeds leave, on average, after half completion time (*random stay*)

Fig. 2. Swarm performance for different seeds holding time (never leave vs random stay) and populations (homogeneous uTP vs homogeneous TCP vs heterogeneous 50-50): completion time evolution and CDF, queue occupancy CCDF.

hence we do not expect congestion to play a major role in this case. Conversely, when seeds leave the system and are replaced by new leechers, resources become rare, translating into longer completion times. Notice that completion time in random stay scenarios is affected by the specific congestion control mechanism adopted by peers in case of heterogeneous swarms (while performance of homogeneous swarms are practically the same).

Reasons why this happens can be better understood by looking at the queue size complementary cumulative distribution (CCDF) shown in the right Fig. 2 plots (gathered by sampling all uplink queues at 10 Hz). Notice indeed that uplink queue of uTP peers is very similar in both (a) and (b), as uTP tries not to exceed a target delay: deviation from target are due to TCP control connection sharing the same queue, and latecomer advantage [8]. On the contrary, TCP queues can grow long: in scenario (b), about 20% of the cases, queues exceed 100 packets, corresponding to more than a second of queuing delay considering full size packets (as reported in the top x-axis for reference). In the 50-50 case, queuing delay aggregates both uTP and TCP uplink buffers, resulting in an intermediate average system queueing time (notice that in the 50-50 scenario, a further deviation from uTP target is due to bursty uplink ACK traffic of TCP connections opened by other peers in the swarm).

Interestingly, the completion time behavior changes if we consider a *heterogeneous swarm*, as shown in the middle plot of Fig. 2-(b). Investigating further, we find that uTP peers have shorter download times in the 50-50 random-stay scenario, as the uTP vs TCP breakdown of the completion time CDF of Fig. 3-(a) testifies (while no difference arises in the heterogeneous 50-50 never-leave scenario). This is a counter-intuitive result, as in principle we would not expect completion time to be tied to the congestion control used to handle chunks upload: indeed, the completion time metric relates to the *download* performance of a uTP peer $\mathcal{P}$ (i.e., a peer using uTP for its *uplink*), that rather depends on the uplink performance of other uTP and TCP peers of the swarm (i.e., peers that upload to $\mathcal{P}$ according to the protocol of their choice).

We suspect this unexpected phenomenon to arise due to (i) the decoupling of the data vs control connection, associated to (ii) the very large size of ADSL buffers: while large buffers are beneficial to backlogged data connections, they can conversely harm BitTorrent signaling. Indeed, TCP control connection competes with either uTP data traffic (that strive to keep a low extra delay on the access buffer) or TCP data traffic (that indiscriminately opens up the congestion window until loss occur). As ADSL buffers are large, queuing delay of TCP peers can grow up to seconds (Fig. 2): hence, this possibly hampers the performance of TCP peers, whose control traffic is significantly slowed down by competing chunk upload to other peers and by ACK traffic of downloaded chunks. Conversely the shorter queuing delay of uTP peers lead to more responsive control connections, that opportunistically "steal" download slots from TCP peers (whose request rate is, as we just saw, slowed down due to self-induced congestion in the access link), as they are faster in filling the request buffer of other

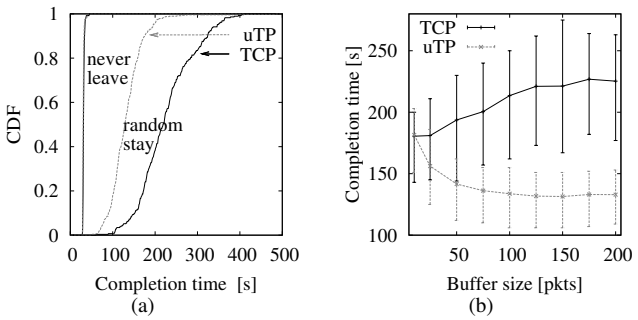


Fig. 3. (a) Breakdown of completion time CDF according to the different peer population in the heterogeneous 50-50 scenario, for different seed holding times. (b) Swarm completion time (mean, 1st and 3rd quartiles) as a function of the buffer size B for the 50-50 random-stay scenario, for different peer population.

peers. Conversely, this phenomenon was not observed in case of homogeneous TCP swarms, as all peers fairly competed against each other.

To confirm this intuition, we carried out a set of experiments varying the buffer size of the access links. Completion time statistics (mean, 1st and 3rd quartile of the distribution) are reported in Fig. 3-(b) for the 50-50 random-stay scenario, with separate curves for uTP and TCP peers. According to [8], uTP can be affected by the buffer size only whenever the target queuing delay exceeds the buffer capacity, in which case uTP becomes as aggressive as TCP and becomes a loss-driven protocol. Fig. 3-(b) confirms this, in that whenever the buffer size cannot sustain the target (e.g., $B = 10$ ms corresponding to a 12 ms full-payload equivalent delay lower than the 25 ms target), uTP and TCP performance cannot be distinguished. Conversely, uTP and TCP show an opposite behavior for increasing buffer size: TCP data transfer will benefit of the increasing buffer size, translating into slower signaling rates, while uTP data transfers will limit the extra queueing delay, whatever the buffer size may be (so that the completion time is roughly unaltered when B>50). We point out that additional dynamics may further come into play, tied to tit-for-tat and peer selection, which we aim at addressing as future work.

V. Conclusion

This work studies the impact of congestion control policies on the main BitTorrent performance metric, namely, the torrent completion time. We perform simulation in `ns2`, integrating the open-source implementations of the swarming protocol [5] and of the new congestion control protocol [8]. We simulate small size swarms consisting of 100 peers, in mild flash crowd scenarios. Peers initiate data connections with either uTP or TCP, but can accept any flavor of incoming connections. Swarm population is either homogeneous (i.e., all new uTP or all old TCP clients) or heterogeneous (i.e., half new and half old clients). Newborn seeds either stay in the system forever, or leave after a random time and are immediately replaced by a new leecher.

Initial simulation results show an interesting and unexpected behavior: namely, the *download time* at torrent completion can be moderately affected by the congestion control preferences on the *uplink traffic* direction. More precisely, in a heterogeneous peer population scenario where seeds leave after a random time, we observed that, as the use of uTP practically bounds the queuing delay, uTP can assist peers into faster signaling as a side effect: in turn, this translates into uTP peers opportunistically gaining download slots faster than TCP peers, whose control traffic competes with a self-congested access link. In other scenarios instead, no major difference can be noticed between uTP and TCP.

Still, we point out that simulation scenarios are often simpler than reality, where complex dynamics may yield to different behaviors (at least according to numerous posts [11], the latest of which dating to October 2010 at time of writing, μTorrent users seldom complain about unstable uTP performance). Hence, results presented in this work should be interpreted with care, as scenarios including heterogeneity in the access capacity and background traffic, should be taken into account as well.

Finally, while we believe open-source `ns2` implementations of BitTorrent and uTP to be extremely valuable for the research community, we also believe simulation not to be an entirely faithful representation of the real world: e.g., [5] does not implement BitTorrent anti-snubbing and end-game algorithms, while [8] does only implement the first draft specification, using furthermore a TCP framing, with therefore a different overhead than the actual uTP implementation. Therefore, another direction we aim at pursuing in the following is to perform PlanetLab experiments, analogous to the ones reported in this paper, to fully discard the hypothesis that the outlined phenomena arise due to simulation artifacts.

References

[1] C. Zhang, P. Dhungel, D. Wu, and K. Ross, "Unraveling the bittorrent ecosystem," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, no. 99, 2011.

[2] S. Morris. (2008, Dec) μTorrent release 1.9 alpha 13485. http://forum.utorrent.com/viewtopic.php?pid=379206#p379206.

[3] (2011, April) New μTorrent 3.0 Beta – Streaming, Sending, Remote Access and More. http://blog.bittorrent.com/2011/04/19/new-ending-remote-access-and-more/.

[4] D. Qiu and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 34, no. 4, pp. 367–378, 2004.

[5] K. Eger, T. Hoßfeld, A. Binzenhofer, and G. Kunzmann, "Efficient simulation of large-scale p2p networks: packet-level vs. flow-level simulations," in *ACM UPGRADE-CN*, Monterey, CA, Jun 2007.

[6] B. Cohen and A. Norberg, "Correcting for clock drift in uTP and LEDBAT," in *Invited talk at 9th USENIX International Workshop on Peer-to-Peer Systems (IPTPS 2010)*, San Jose, CA, Apr 2010.

[7] J. Schneider, J. Wagner, R. Winter, and H. Kolbe, "Out of my Way – Evaluating Low Extra Delay Background Transport in an ADSL Acciess Network," in *22nd International Teletraffic Congress (ITC22)*, Amsterdam, The Netherlands, Sep 2010.

[8] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "LEDBAT: the new BitTorrent congestion control protocol," in *19th IEEE International Conference on Computer Communications and Networks (ICCCN 2010)*, Zurich, Switzerland, Aug 2010.

[9] A. Norberg. (2010, May) Dev. comment on uTP main purpose. http://forum.utorrent.com/viewtopic.php?pid=488896#p488896.

[10] S. Shalunov, "Low Extra Delay Background Transport (LEDBAT)," IETF Draft, Mar 2010.

[11] http://forum.utorrent.com/viewforum.php?id=11.