Data Plane Throughput vs Control Plane Delay: Experimental Study of BitTorrent Performance

Claudio Testa[1], Dario Rossi[1], Ashwin Rao[2], Arnaud Legout[2]
[1] Telecom ParisTech, Paris, France – first.last@enst.fr
[2] INRIA Planete, Sophia Antipolis, France – first.last@inria.fr

***Abstract*—In this paper, we address the trade-off between the data plane efficiency and the control plane timeliness for the BitTorrent performance. We argue that loss-based congestion control protocols can fill large buffers, leading to a higher end-to-end delay, unlike low-priority or delay-based congestion control protocols. We perform experiments for both the uTorrent and mainline BitTorrent clients, and we study the impact of uTP (a novel transport protocol proposed by BitTorrent) and several TCP congestion control algorithms (Cubic, New Reno, LP, Vegas and Nice) on the download completion time.**

Briefly, in case peers in the swarm all use the same congestion control algorithm, we observe that the specific algorithm has only a limited impact on the swarm performance. Conversely, when a mix of TCP congestion control algorithms coexists, peers employing a delay-based low-priority algorithm exhibit shorter completion time.

I. Introduction

Recently, BitTorrent replaced TCP for a low-priority delay-based congestion control protocol known as uTP [1]. This change was motivated by the inflation of buffer delays in the current Internet [2] due to the TCP loss-driven congestion control algorithm, a problem later called "bufferbloat" by J. Gettys [3]. Indeed, in case of a large buffer size (typical at the access point), congestion control algorithms such as TCP Cubic may cause the buffer queuing delay to grow up to several seconds. Conversely, lower-than-best effort protocols such as uTP explicitly limit the additional queuing delay irrespective of the buffer size. However, such a limit might result in data plane inefficiency, that is, the protocol may not be able to fully utilize the available network resources.

Motivated by the above observations, we investigate the implication of uTP on BitTorrent's performance. Put in abstract terms, in the design of non realtime peer-to-peer (P2P) systems, we are interested in studying the relative importance of data plane throughput and control plane delay.

The task of the data plane is essentially to efficiently transfer data, so the *throughput* is the desired metric to evaluate the data plane performance. The task of the control plane is instead to spread the knowledge that data has moved in the system. If we define the *delay* as the time needed to propagate information about the data transfer, this delay is the desired metric to evaluate the control plane efficiency.

Fig. 1 shows a dual interdependence of the control and data planes. The control plane propagates the information about availability of data at a peer after this data has been transferred by the data plane to that peer. The data plane replicates the data available at a peer, when information about the availability of the data at that peer is disseminated to the other peers that need a copy. Yet, the control and data planes are mere logical abstractions that co-exists in the same software, run on the same hardware, and share the same network communication path and physical channel. Therefore, the data plane transfer may interfere with the control plane information dissemination. For example, the control plane information can get queued behind[1] a large volume of data injected in the network by the data plane; this can slow down the information spreading, and affect the data plane in turn.

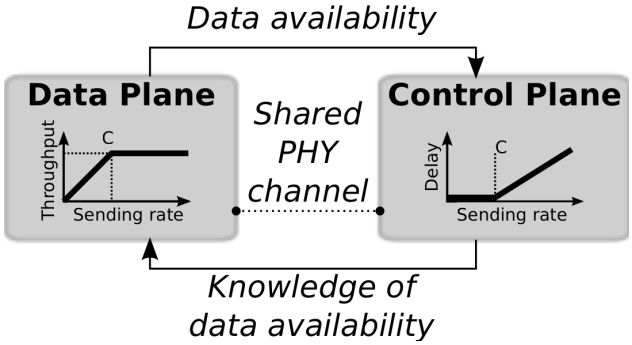

Fig. 1. Control and data plane interdependence.

We argue that systems should be optimized for the data plane throughput, as long as this does not hurt the control plane delay. However this is a delicate trade-off. As shown in Fig. 1, when the sending rate exceeds the system capacity C, the throughput saturates, but the delay grows due to buffering. When the sending rates falls below C, the data transfer is not efficient, and this affects not only the data plane, but also the control plane since there is no new information to be sent out.

In this paper, we perform experiments, considering both the uTorrent and mainline BitTorrent clients, and we study the impact of uTP (a novel transport protocol proposed by

[1]If control messages queue up behind large data chunks system information propagates more slowly, with possibly negative influence of future system decisions. For instance, the Rarest First policy of BitTorrent can select chunks that are instead not rare in the data plane, but whose availability has not been yet advertised by the control plane due to the delay – which can in turn lead to data plane bottlenecks later on.

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

BitTorrent) and several TCP congestion control algorithms (Cubic, New Reno, LP, Vegas and Nice), on the completion time of a file downloaded using the BitTorrent protocol. Summarizing, we observe that in case peers in the swarm all use the same congestion control algorithm (homogeneous scenario) then the specific algorithm has only a limited impact, while the application settings and policies have a paramount role in determining the swarm performance. Conversely, when a mix of congestion control algorithms coexists (heterogeneous scenario), peers employing a delay-based low-priority congestion control can opportunistically reduce their completion time (due to faster signaling on their uplink) by a significant amount.

II. Background

To the best of our knowledge, this specific question of the interactions between the data and control planes has received little attention. With client-server communications [3], when the TCP data plane sends a very large window of data to a narrow pipe with a large[2] buffer, the TCP control plane will be impacted because it relies on timely notification of packet losses for correct operation. Citing Cerf *et al.* [3] "with such large buffers, TCP's slow-start algorithm does not see any drops and thus greatly overestimates the correct pipe size and requires multiple packet drops before TCP can enter its congestion-avoidance phase".

In the case of P2P communications, BitTorrent performance has been investigated a lot since the seminal work [4], even if the focus has been on the overlay layer [5] or the impact of lower layers on traffic localization [6]. Few works addressed the impact of the transport layer on BitTorrent performance [7], [8]. Eger *et al.* [8] proposed an open-source packet-level BitTorrent module for `ns2`. They show that the transport-layer congestion control dynamics interacts with application-level dynamics, so that simulations considering the application-level only [5], [6] may be optimistic. Testa *et al.* [7] adopted the same simulator as the one of Eger *et al.* to study the impact of uTP on BitTorrent completion time. As such this work is the closest to our one. They show an unexpected phenomenon — namely, peers employing uTP in their uplink have a shorter completion time than with TCP. The explanation is that uTP peers have a faster control plane, so they are more opportunistic in "stealing download slots to TCP peers".

Yet, Testa *et al.* relied solely on simulations, so that we cannot exclude results to be simulation's artifacts, whereas this work is instead based on experiments using real BitTorrent clients and transport protocols. We indeed argue that performing experiments is an important step in the understanding of the complex interactions between the data and control planes.

III. Methodology

In this paper, our goal is to study the trade-off between the data plane efficiency and the control plane timeliness on the content dissemination performance. We address this question with controlled experiments with different congestion control algorithms that intrinsically favor either the data plane throughput or the control plane delay. In this section, we first summarize the congestion control algorithms used and then we provide relevant details on our experimental setup.

[2]Here "large" is relative to the pipe capacity: a 128 KB buffer can queue 1 second worth of packets when in front of a 1 Mbps channel.

TABLE I. Congestion Control Design Space: Aim vs Strategy

		Strategy	
		Delay-based	**Loss-based**
Aim	**High-priority**	Vegas	Cubic, NewReno
	Low-priority	Nice, uTP	LP

A. Congestion Control Algorithms

For our analysis, we consider four different categories of congestion control protocols, two based on the *design strategy* to detect losses and two based on the *aggressiveness* in grabbing the available bandwidth. The two categories based on design strategies are defined by the *loss-based* or *delay-based* congestion control algorithm. The loss-based algorithms infer congestion using a packet loss. The delay-based algorithms infer congestion using delay's variation on the flow path. The two categories based on the aggressiveness are called *high-priority* and *low-priority*. The high-priority algorithms are efficient (and aggressive) in using the spare bandwidth, while the low-priority algorithms are designed to use the spare bottleneck bandwidth as a scavenger service.

In Tab. I, we report the most representative congestion control algorithms for each of the above categories. IETF endorses NewReno [9], a high-priority loss-based congestion control algorithm for TCP. Vegas [10] was proposed as a high-priority delay-based congestion control algorithm as an alternative to the traditional loss-based NewReno algorithm. Recent evolution of loss-based algorithms include Cubic [11] and Compound [12]. Cubic has become the default algorithm for TCP in Linux since kernel version 2.6.18 and Compound the default one in Windows. LP [13] is a low-priority loss-based congestion control algorithm available in Linux, and Nice [14] and uTP [15] are two low-priority delay-based congestion control algorithms that react on Round Trip Time (RTT) and One Way Delay (OWD) variations respectively. Unlike Nice, uTP explicitly limits the additional delay it adds to the bottleneck buffer. This limit is currently set to 100 ms to avoid harming interactive, delay-sensitive traffic [1].

In this work, we implemented Nice as a kernel module (relatively simple extension from Vegas) and uTP as an application-layer protocol on top of UDP. As we shall see, though this may seem at first a detail, this difference has an important impact on the methodology of this study.

Intuitively, high-priority or loss-based design favors data plane efficiency, whereas delay-based or low-priority congestion control favors control plane timeliness. But, this statement is an oversimplification. For example, the low-priority but loss-based LP algorithm can still generate losses, and the queue can grow unbounded and be an obstacle to control plane timeliness. Similarly Vegas, a high-priority delay-based protocol, should keep the queue short and favor control plane timeliness in spite of being designed for data plane efficiency. We therefore

TABLE II. APPLICATIONS AND CONGESTION CONTROL IN OUR EXPERIMENTS

Client		CC-L4	CC-L7
(BT)	mainline	Cubic, Vegas, Nice, LP	
(UT)	uTorrent	Cubic	uTP

recommend on-field tests to precisely assess the actual impact of the congestion control algorithms.

B. Experiment Description

We performed our experiments on a private cluster, considering a flash crowd scenario in which 75 leechers join a swarm initially populated by a single seed. The seed is used to distribute a 100 MBytes file in the torrent. Each machine of the cluster runs a single peer, seed or leecher. The uplink is limited to 1 Mbps on the machines running the leechers and 5 Mbps on the machine running the seed. We consider a queue size of 1 second for the uplink, which we believe represents a conservative estimate based on the results made available by Netalyzer [2], [3]. BitTorrent enables the users to limit the upload rate. For our experiments we set the upload rate limit on each peer to five times the uplink bandwidth. This ensured that the upload rate is limited by the uplink bandwidth on the machine and not by the BitTorrent application.

We study the impact of the congestion control algorithms on the data plane efficiency and control plane timeliness by using the uTorrent 3.0 (UT for short) client and an instrumented version of the mainline BitTorrent 4.0.2 (BT for short) client [16]. We conduct experiments on two scenarios. In the *homogeneous* scenario, all peers use the same congestion control algorithm, and in the *heterogeneous* scenario half of the peers use TCP Cubic, and half of the peers use another congestion control among uTP, NewReno, Vegas, Nice and LP. In the remainder of this work, we denote a set of peers in an experiment with the notation X:Y, with $X \in \{$BT, UT$\}$ and $Y \in \{$uTP, NewReno, Vegas, Nice, LP$\}$. Additionally, we employ the notation X:$\star$ to indicate any congestion control algorithm used with application X.

All outgoing connections of a BT peer will use the same congestion control algorithm, because the operating system imposes a single algorithm at the *transport-layer* (CC-L4). However, as reported in Tab. II and unlike other protocols, uTP is implemented at the *application-layer* (CC-L7) and is only available for uTorrent. UT implements a dual-stack connection management, so that: (i) a uTP and a TCP connections are opened in parallel; (ii) in case a uTP connection is successfully established, the TCP one is dropped; (iii) connections are bidirectional, so that a uTP incoming connection can be used in the reverse direction. We notice that it is possible to disable uTP in UT with an application level setting (`bt.transp_disp` in the GUI). In the following, we consider the UT:Cubic variant as well, in order to gauge whether performance differences are rooted in the scenario, in congestion control choice, or in the application-layer implementation and settings (e.g., dual-stack connection management policies, fine tuning of timers, maximum number of simultaneous connections, etc.)

The instrumented mainline BitTorrent client logs the receive time of all control plane messages (e.g., the HAVE or BITFIELD messages) and of all data plane messages (the PIECE messages). However, we use the original, non-instrumented, uTorrent client that does not log such events. As such, we measure the performance using metrics that are available for both clients: (i) the download completion time T_i of leecher i as reported by the tracker and (ii) the instantaneous queue size $B_i(t)$ measured at kernel-level by peer i. Intuitively, $B_i(t)$ is a metric of the delay in the control plane, and T_i is a metric combining the data plane throughput and the control plane delay.

IV. EXPERIMENT RESULTS

A. Download time and Buffer occupancy

We first focus on the download completion time and the buffer size. We consider different applications (BT, UT), congestion control algorithms (Cubic, uTP, NewReno, Vegas, Nice, LP), and scenarios (homogeneous, heterogeneous). We report our results in Fig. 2, the top figures are for the homogeneous scenarios and the bottom figures are for the heterogeneous one. We use for the buffer occupancy (left plots) an x-axis in KBytes and the corresponding buffering time in seconds for new packets entering the queue.

Fig. 2-(a) shows the cumulative distribution function (CDF) of the completion time for the homogeneous scenario. We observe that the congestion control algorithm has a moderate impact on the completion time of BitTorrent mainline clients (BT:$\star$) and that the completion time is generally shorter with uTorrent (UT:$\star$ < BT:$\star$). We also observe a higher fairness among uTorrent peers than among BT peers, because the UT:$\star$ CDF has a steeper increase, meaning that completion times are very similar for all uTorrent peers. Finally, we observe that the completion time for uTorrent is shorter with uTP than with Cubic (UT:uTP < UT:Cubic). These observations suggest that both BitTorrent applications and congestion control algorithms have an impact on BitTorrent's performance, though the impact of the congestion control algorithms seems, at least for BT:$\star$, limited according to Fig. 2-(a).

Fig. 2-(b) shows the CDF of the buffer occupancy for the homogeneous scenario. We observe that the delay-based algorithms (UT:uTP, BT:Nice, BT:Vegas) leads to the shortest buffer occupancy, followed by the loss-based algorithms (BT:LP and BT:NewReno), and by BT:Cubic as Cubic is more aggressive than NewReno. Interestingly, the behavior of UT with Cubic (UT:Cubic) is very different from the one of BT with Cubic (BT:Cubic), confirming the complex interactions between the application and the congestion control algorithm.

We see in Fig. 2-(a,b) no systematic correlation between the completion time and the buffer occupancy. Indeed, whereas UT:uTP has the shortest completion time and lowest buffer occupancy, BT:Nice has among the worst completion time (slowest peers for all swarms are BT:Nice), but the second shortest buffer occupancy.

Interestingly, results change significantly with an heterogeneous scenario, see Fig. 2-(c,d). In this scenario, half of the peers are using Cubic and half of them are using a single other

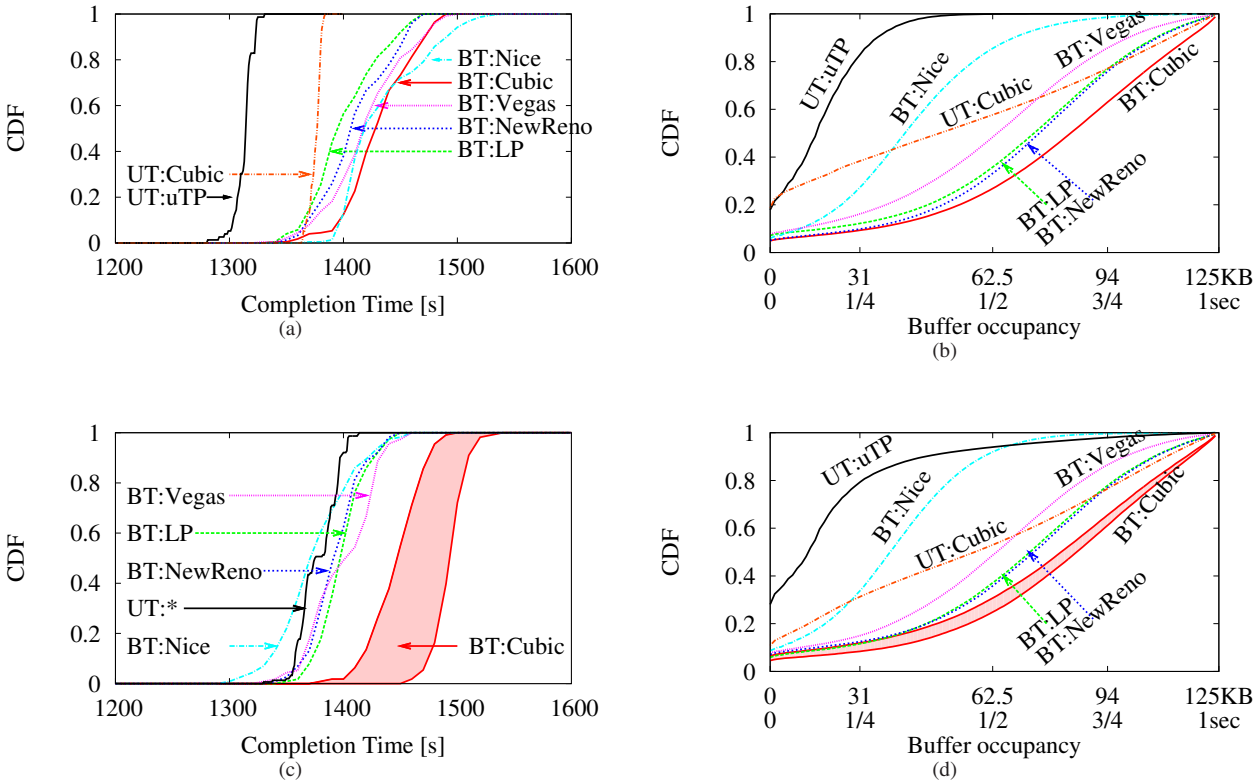

Fig. 2. Performance of (a)-(b) Homogeneous and (c)-(d) Heterogeneous swarms.

congestion control algorithm among uTP, NewReno, Vegas, Nice and LP. For the sake of clarity, we show a CDF curve for each half of the peers using the same congestion control algorithm. In order to reduce the set of CDF curves and because the performance of the BT:Cubic peers is different from the other sets of peers, we show an envelope (with light gray shaded color) instead of independent curves. For instance, BT:Vegas is for half of peers using Vegas only for an heterogeneous scenario, and the other half using Cubic falls into the envelope BT:Cubic.

We observe in Fig. 2-(c,d) that the completion time for BT:Cubic is greater than with any other combination of application and congestion control algorithms, and that the completion time for UT:uTP and UT:Cubic peers are indistinguishable[3], so we represent it with a single curve UT:⋆. We also observe that the shortest completion time is for BT:Nice, whereas it was for UT:uTP for the homogeneous scenario.

We notice two important changes in the buffer occupancy for the heterogeneous scenario compared to the homogeneous one in Fig. 2-(d). The BT:Nice buffer occupancy is closer to the one of UT:uTP, and a few percent of peers have a higher buffer occupancy with UT:uTP than with BT:Nice because the CDF crosses.

[3]This results from UT connection management, that allows on the one hand UT:Cubic peers to employ reverse uTP traffic, but also forces UT:uTP peers to send TCP Cubic traffic to UT:Cubic peers with whom they have not successfully established a uTP connection yet.

In summary, results for the heterogeneous scenario confirm that application implementation may have an important impact (i.e., UT connection management allowing uTP and Cubic traffic to mix), but also show that the congestion control algorithm has a significant impact. Contrasting our findings with related work, the difference of performance between BT:Nice and BT:Cubic is similar to the one noticed by Testa *et al.* using simulations [7], but our results on UT are different because the simulations presented by Testa *et al.* do not take into account the complex UT connection management policy.

B. Correlation-based Analysis

We present in Tab. III a correlation based analysis, in order to compactly summarize our findings and assess whether differences in the buffer statistics explain the completion time difference. We compute the median and 90-th percentile of the T_i and B_i metrics, and we evaluate the Spearman's correlation coefficient as $\rho(X(T_i), Y(B_i))$. As an example, to compute the Spearman's correlation between the 90-th percentile of the queuing delay and the completion time for BT, we define a set of $(50th(T_i), 90-th(B_i))$ pairs, where statistics are computed for any given swarm $i \in$BT:⋆. The correlation among the pairs is then computed over the set of swarms.

Intuitively, the Spearman's correlation quantifies whether an order exists between two different metrics. In our case, it will show whether the order observed for the completion time is the same as for the buffer occupancy. In more details, while

TABLE III. SPEARMAN'S RANK CORRELATION

			Buffer B			
			50-th	90-th	50-th	90-th
Hetero	Time T	50-th	0.54	0.32	0.10	0.10
		90-th	0.36	0.14	-0.30	-0.30
Homo		50-th	0.77	0.56	0.62	0.67
		90-th	0.76	0.58	0.60	0.69
			Multi applications { UT:⋆ ∪ BT:⋆ }		**Mono application** { BT:⋆ }	

Pearson's correlation coefficient is directly evaluated over two metrics, and expresses the existence of a *linear* relationship between them, the Spearman's correlation coefficient is instead evaluated over their *rank* and expresses the existence of a monotonous (but not necessarily linear) relationship between the metrics.

For the homogeneous scenario, we see that in the BT:⋆ case, the buffer occupancy and completion time are not correlated. This is coherent with findings in [17], whereby an additional delay equal for all peers have only minimal impact on system performance. Rather, completion time of slowest peers (90-th) is negatively correlated with delay (recall that slowest BT:Nice peers complete last). We only observe correlation when we consider both BT:⋆ and UT:⋆, hinting for an important impact of application settings.

In the heterogeneous scenario instead, buffer occupancy and completion time are highly correlated, for both BT and UT (and even considering BT:⋆ alone). This extends the validity of previous simulation findings [7] to the real world, additionally showing that congestion control algorithms, neglected by previous studies, play an important role as well.

V. DISCUSSION AND FUTURE WORK

In this paper, we revisit the bufferbloat problem under a novel light, i.e., the trade-off between data plane efficiency and control plane timeliness in distributed systems. We performed controlled experiments with two different BitTorrent clients and several congestion control algorithms designed with different aims (i.e., low-priority and high-priority) and strategies (i.e., loss-based and delay-based).

Though preliminary, this work show important results. We see that the congestion control algorithm can significantly impact the completion time, because peers experiencing lower buffer delays will experience a shorter download completion time in an heterogeneous scenarios. However, the completion time and buffer occupancy are uncorrelated in homogeneous scenarios, which shows that the buffer occupancy alone cannot explain all observed differences in performance.

As expected, application-layer settings (e.g., dual stack uTP/Cubic connection management, fine tuning of application parameters) have a paramount role as well. Specifically, we found that completion time for peers is the fairest with uTorrent for both the homogeneous and heterogeneous scenarios. Fairness in completion time between users is due to the dual-stack connection management with an hard-coded preference toward the low-priority delay-based uTP protocol. Yet, uTorrent being closed-source, a radical methodological shift is possibly needed (e.g., capture and parse BitTorrent traffic) to expose more details (e.g., control information) that would allow to drill this issue further.

More experiments, on PlanetLab and the wild Internet, are required to assess the generality of these findings, to ensure these phenomena to hold in more heterogeneous scenarios, e.g., with mixtures of congestion control flavors, (BSD-based MacOS employs TCP NewReno as default TCP flavor, while Windows and Linux resort to TCP Compound and TCP Cubic respectively), or considering application-layer limit of uplink bandwidth consumed by BitTorrent. Additionally, we would like to quantify the impact of each parameter (e.g., congestion control, connection management, etc.) so to give guidelines for optimal tuning of legacy BitTorrent clients.

ACKNOWLEDGEMENT

This work has been carried out at LINCS http://www.lincs.fr. The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n. 318627 (Integrated Project "mPlane"). Experiments presented in this paper were carried out using the Grid'5000 experimental testbed (see https://www.grid5000.fr).

REFERENCES

[1] http://forum.utorrent.com/viewtopic.php?pid=379206#p379206.

[2] http://netalyzr.icsi.berkeley.edu.

[3] V. Cerf, V. Jacobson, N. Weaver, and J. Gettys, "Bufferbloat: what's wrong with the internet?" *Commun. of the ACM*, vol. 55, no. 2, Feb 2012.

[4] D. Qiu and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks," *ACM SIGCOMM*, 2004.

[5] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and Improving a BitTorrent Performance Mechanisms," in *IEEE INFOCOM)*, 2006.

[6] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang, "Improving Traffic Locality in BitTorrent via Biased Neighbor Selection," in *ICDCS*, 2006.

[7] C. Testa and D. Rossi, "The impact of utp on bittorrent completion time," in *IEEE P2P*, 2011.

[8] 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*, 2007.

[9] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," *IETF RFC5681*, 2009.

[10] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: new techniques for congestion detection and avoidance," *ACM SIGCOMM*, 1994.

[11] I. S. Ha, Rhee and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," in *ACM SIGOPS*, 2008.

[12] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *IEEE INFOCOM*, 2006.

[13] A. Kuzmanovic and E. Knightly, "TCP-LP: low-priority service via end-point congestion control," *IEEE/ACM Transactions on Networking*, vol. 14, no. 4, p. 752, 2006.

[14] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP Nice: A mechanism for background transfers," in *USENIX OSDI*, 2002.

[15] http://www.bittorrent.org/beps/bep_0029.html

[16] http://www-sop.inria.fr/members/Arnaud.Legout/Projects/p2p_cd.html

[17] A. Rao, A. Legout, and W. Dabbous, "Can realistic bittorrent experiments be performed on clusters?" in *IEEE P2P*, 2010.