BTLab: A System-Centric, Data-Driven Analysis and Measurement Platform for BitTorrent Clients

Martin Szydlowski*, Ben Y. Zhao†, Engin Kirda‡ and Christopher Kruegel†
msz@seclab.tuwien.ac.at,ravenben@cs.ucsb.edu,ek@ccs.neu.edu,chris@cs.ucsb.edu
*Vienna University of Technology, 1040 Vienna, Austria
†University of California, Santa Barbara, CA 93106, USA
‡Northeastern University, Boston, MA 02115, USA

***Abstract*—We present BTLab, a distributed platform to measure and analyze the differences between BitTorrent clients. Due to extensibility, and a certain vagueness in the BitTorrent specification, many clients diverge in some aspects from each other. Most research to date disregarded the effects of these differences. BTLab allows us to create and control BitTorrent swarms, composed of hundreds of clients of our choice. We use captured network traffic to measure the performance and uncover the reasons for observed differences. For our experiments, we deployed BTLab on a cluster and on Planetlab and selected four popular BitTorrent clients. Our analysis reveals flaws in piece selection and connection management algorithms that adversely affect the performance of some clients.**

I. Introduction

BitTorrent (BT) is a peer-to-peer file transmission protocol, designed to efficiently distribute large files over the Internet. Since its inception, it has experienced tremendous success, and currently makes up a significant fraction of Internet traffic (a 2009 study estimates 25 to 60% [10]). BitTorrent was created and introduced (as a freely available open source client) in 2001 by a single individual, Bram Cohen. Only in 2003, after it gained popularity (and other developers started creating alternative implementations), a semi-formal BitTorrent specification [5] was published by Cohen. However, the protocol and its key algorithms never underwent a rigorous standardization process, and parts of it remain vague to this day. As a consequence, developers have much leeway in implementing the specification and adding (often incompatible) extensions.

We have created BTLab to study the effects these different interpretations and improvements have on the performance (download speed) of BT clients. BTLab allows us to configure and launch hundreds of instances of real-world BT clients to carry out experiments with realistic peer numbers (swarm sizes). In addition, we have developed a set of tools that allow us to record and analyze the network-level activity and behavior of these clients. We argue that this *system-centric* approach has certain advantages over studies which rely on modeling and simulation and, inherently, require simplification and abstraction. A large part of BT research belongs to that category, and may overlook side effects or emergent properties that only appear in real systems. The main contribution of BTLab is the possibility to study the entire system (all peers in a swarm) with an unprecedented level of detail (individual protocol messages). This allows us to measure and analyze properties (e.g., global distribution of pieces) that cannot be examined by looking at a single (or a few) peers alone. By making our analysis *data-driven*, that is, based solely on data the clients emit in the course of regular operation, we can analyze a wide range of implementations, including closed-source ones.

We have deployed BTLab on a cluster of 32 machines connected with fast network links (an easily controllable environment), and on Planetlab, a global network of hosts frequently used by researchers (offering realistic network conditions), and used it to study four popular BitTorrent clients: Azureus (Java), μTorrent (closed-source), Transmission (C++) and Mainline (Python). Our results show that Azureus has the best overall performance and that clients exhibit (sometimes wildly) diverging behavior depending on the experimental conditions. The analysis tools we developed allowed us to investigate the root causes of the observed behavior.

II. BTLab Infrastructure

The BTLab infrastructure has two parts: The controller, responsible for preparing, executing and monitoring the experiments and recording the network traffic, and the analysis toolbox, used to process the data and extract information.

BTLab Controller. We have developed a distributed platform that allows us to configure and control BitTorrent clients, and to record all network traffic. The platform is modular and can be deployed on any number of hosts running Linux. The platform is split in a set of global components that manage the experiments and local components that are deployed on each physical host that participates in an experiment. The global components include a BT tracker and scripts that control the configuration, the collection of network dumps, and monitoring of the experiment progress. The local components configure and control individual BT clients (several clients can run on one physical host), network traffic capturing, and auxiliary services needed by some clients (e.g., `X`, `wine`).

Analysis Toolbox. We have developed a set of tools that allows us to analyze all data exchanged over the network in a BT swarm from different perspectives, and at different granularity levels. In a preprocessing step, we use `Bro` (http://bro-ids.org) with a customized BT protocol parser to extract individual protocol messages from captured network traffic. Using this as raw data, we reconstruct higher level views from the bottom up. On the lowest level, we analyze the types, frequency, order and content of messages from individual connections. We are able to reconstruct the connection state, which the peers keep internally, and we can verify the validity of all exchanged messages. In particular, while the flow of messages should follow the protocol rules, we noticed that, occasionally, clients both violate the rules and gloss over violations from peers.

On the next level, we group individual connections to examine the mechanisms that a client uses to select peers, and how it downloads the parts of a file from other nodes. We can follow the download progress, by counting the complete unique pieces of the file that the client has received, to compute net throughput and completion time. Additionally, we can see if clients send unsolicited pieces or request redundant pieces (both actions break the rules governing piece exchange). The announcements a client receives from its peers allow us to reconstruct the client's view of the availability of individual pieces, and reason about the criteria a client uses to choose which pieces to download next. Finally, by looking at the state, throughput and timing of all connections from one client, we can draw conclusions about how it manages the connections with other peers.

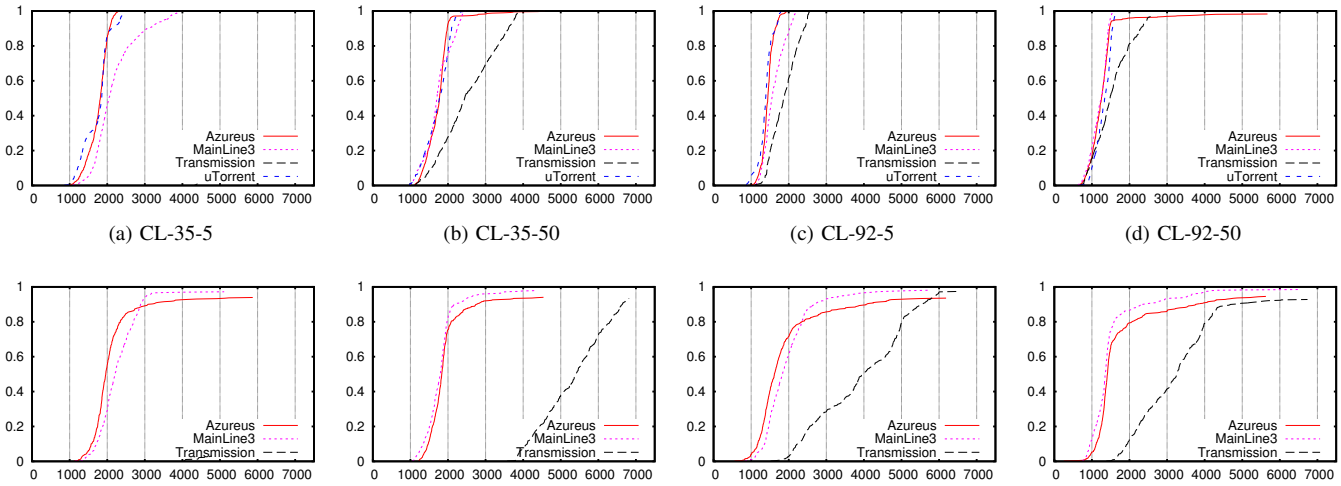
On the highest level, we aggregate individual peers to a swarm-wide view. This allows us to examine the evolution of the swarm population (seeders/downloaders, fast/slow peers) and a detailed global view of the piece distribution over the course of an experiment. Apart from the traffic between peers, we also look at peer-to-tracker communications. This data is coarse-grained, but accurate enough to quickly compute basic statistics like swarm size and download times.

III. Experimental Setup

Deployment. We installed BTLab in two environments. The first installation was on a cluster of 32 (28 available to us) physical hosts with 2.4GHz CPUs and 2GB RAM, linked via 1Gbps Ethernet connections. This allowed us to run experiments with up to 580 peers in a controlled environment. To study the BT clients in a globally distributed, heterogeneous environment, we also deployed BTLab on Planetlab. We obtained a slice (virtual server) on 512 nodes and performed experiments with one peer per node. This facilitated swarm sizes of the same magnitude as on the cluster, and comparable to medium-sized swarms on the Internet. As the focus of our analysis, we chose three highly popular (according to several Internet surveys) BitTorrent applications: Azureus (ver. 3.0.4.2), μTorrent (ver. 1.8.0), and Transmission (ver. 1.51). In addition, we decided to include the original BitTorrent client developed by Bram Cohen (referred to as Mainline) as baseline comparison. We used the oldest version (ver. 3.9.1) available from the `bittorrent.com` archives.

Limitations. Due to the nature of the BT protocol, the Bro parser (and subsequent analysis) is sensitive to packets dropped during capture. In general, it is impossible to extract the messages from a connection after the point where a packet is missing. Therefore, dropped packets degrade the precision of the analysis. On the cluster, we were able to empirically determine the maximum number (15-20) of peers per node that could run simultaneously without interfering with lossless packet capture. On Planetlab, the resources of a single physical machine are shared between many virtual machines. Therefore, we ran one peer per node at a time, but even with that restriction, lossless capture was not guaranteed. We observed a median packet loss ratio of 0.3% with 20% of the nodes having a packet loss ratio higher than 1%. This translated to, on average, 4% of connections in each experiment to be not recoverable in their entirety. Additionally, we were unable to run μTorrent on Planetlab because of an anti-scam warning that the client displays in a modal dialog. The circumvention of this dialog required a single manual interaction for each host which we intended to use, therefore it was feasible on the 28 cluster hosts but not feasible on 500+ Planetlab hosts. Despite the downsides, the more realistic network topology of Planetlab still provided interesting insights. Another limitation of the Bro parser is the inability to process obfuscated BT traffic that many clients support in an attempt to avoid ISP throttling. Since the use of protocol obfuscation is configurable in every client, we disable it for our experiments.

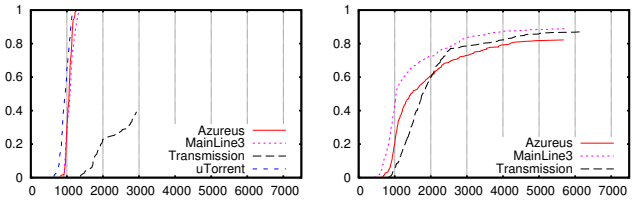
Experiments. For our experiments, we decided to simulate *flash crowds*, a common and challenging scenario, where a single initial seed has to face many downloaders joining in short intervals. To model the inter-arrival times, we used a Poisson process (as suggested by [19]) with a rate parameter of 10 new peers per minute. To reflect observed user behavior, a fraction of clients remains connected after completing the download, that is, these clients become additional seeds. We chose that fraction to be 30%. We repeated every experiment for each of the applications, using identical settings, to gather directly comparable data. Between experiments, we varied several client parameters: upload bandwidth, number of connections and file size. Since altering the file size while other parameters remained fixed did not reveal additional insights, we used it rather to control the duration of the experiment, that is, we chose small files for experiments with low bandwidth and large files for experiments with high bandwidth. The files we used were 64, 128 and 512 MiB in size for upload bandwidths of 35, 92 and 512 kB/s, respectively, and contained only null bytes. Because BitTorrent does not employ compression, these provided identical results as files containing random bytes but with the advantage that the network traffic we captured was highly compressible. The upload bandwidths were chosen to approximate upload bandwidths available to DSL/Cable subscribers. The number of connections ranges from 5 (the minimum number suggested in BitTorrent's tit-for-tat algorithm description) to 100 (more than the default



(a) CL-35-5 (b) CL-35-50 (c) CL-92-5 (d) CL-92-50

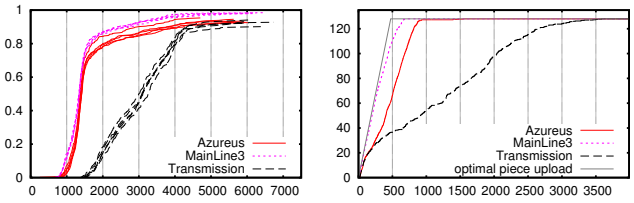
(e) PL-35-5 (f) PL-35-50 (g) PL-92-5 (h) PL-92-50

Fig. 1. Download time CDFs for different experiments. [x = time in seconds; y = fraction of clients]



(i) CL-512-50 (j) PL-512-50

Fig. 1. Download time CDFs for different experiments. (cont.)



(a) PL-92-50 (b) PL-95-50

Fig. 2. (a) Experiment repeated 5 times with identical settings. [x = time in seconds; y = fraction of clients] (b) Unique pieces uploaded by initial seed. [x = time in seconds; y = MiB]

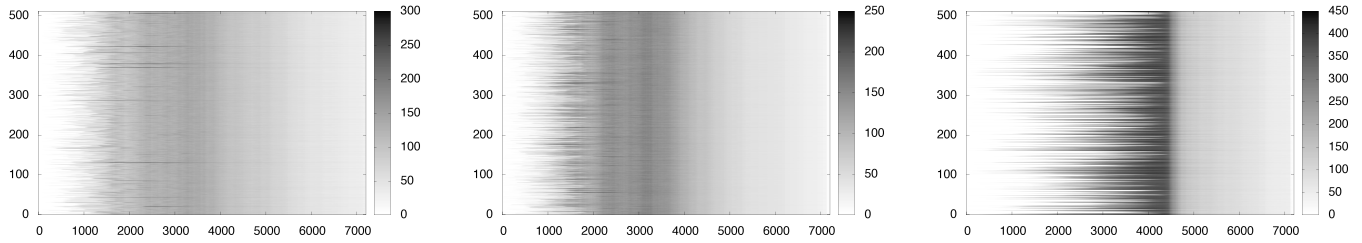
setting in all clients used). For the following evaluation, we used the data from 10 experiments (40 swarms) performed on the Cluster and 16 experiments (48 swarms) on Planetlab. We will refer to individual experiments by a combination of their defining parameters: Platform (PL or CL), upload speed limit (in kB/s) and connection limit (for example, CL-92-50).

IV. Experimental Results

In this section, we present the results of the experiments outlined in the previous section. First, we compare download completion times as a basic metric for performance. Then, we use our analysis tools to find explanations (reasons) for the observed differences.

Completion Time. Since the purpose of BitTorrent is to distribute files over the network as fast as possible, the basic measure for the performance of a client (and the entire swarm) is the download completion time (t_{dl}). Intuitively, shorter download times are considered better. For each experiment, we plot a cumulative distribution function (CDF) that captures t_{dl} for all clients from that experiment. The CDF plots from selected experiments are shown in Figure 1. Note that some lines are cut off before they reach 1; this means that only this fraction of clients was able to complete the entire download.

Comparing the CDF plots of all performed experiments, the first observation we can make is that there is a clear difference between the results of experiments performed on the cluster (top row - Figures 1(a)- (d),(i)) and experiments on Planetlab (bottom row - Figures 1 (e)-(h), (j)). The homogeneous environment of the cluster reduces the variance in t_{dl} for each client. In contrast, the t_{dl} observed on Planetlab are more wide-spread. Moreover, the results on Planetlab exhibit a long tail, where the slowest 20% of clients are much slower than the average, and, typically, a non-negligible number of clients never finish at all. We have to mention that, as a consequence of the fluctuating conditions on Planetlab hosts, there is, for each experiment, a small percentage of nodes where clients fail to start. We exclude the data (if we were able to obtain any) from these nodes from the evaluation for that particular experiment. To ascertain that these limitations, and also the packet loss mentioned in Section III, do not impede the precision of our analysis too much, we performed a repeatability test: We conducted five experiments with identical settings and plotted the CDFs in one graph. As we can see in Figure 2(a), the results are consistent within a reasonable margin of error. Comparing the results for the different clients, we can draw the following conclusions:



(a) PL-92-50 (Azureus) (b) PL-92-50 (Mainline) (c) PL-92-50 (Transmission)

Fig. 3. Global distribution of piece copies in swarm. [x = time in seconds; y = piece id; z (grayscale) = number of piece copies]

- **Azureus** has most consistent performance under all conditions, both on the cluster and on Planetlab. It is always among the better-performing clients (Fig. 1).
- **Mainline** has a performance comparable to Azureus under most conditions, sometimes even outperforming it (Fig. 1(f),(h),(j)). This is somewhat surprising, but it demonstrates that enhancements in the more modern clients did not improve the basic functionality, that is downloading files as quickly as possible. The only scenarios where Mainline is outperformed is when the number of connections is limited (Fig. 1(a),(e),(d),(g)).
- **μTorrent** shows performance similar to Azureus, and it is hardly affected by various parameter settings. μTorrent seems to have a slight advantage over other clients in experiments where available bandwidth was high (Fig. 1(i)), but we could not confirm that result on Planetlab.
- **Transmission** consistently displays the poorest performance. It is adversely affected by low bandwidth and a low connection limit. With strictly limited resources, most Transmission peers are unable to to complete the download in the allotted time (Fig. 1(e)), whereas high bandwidth (Fig. 1(j)) allows Transmission almost to compete with Azureus.

Global Piece Distribution. Having detailed records of all protocol messages exchanged by all clients in a swarm allows us to precisely reconstruct the distribution of piece copies (Δpcs) over the course of each experiment. We performed a systematic comparison of the distributions for all experiments, using different ways to visualize and quantify the data. As an example, Figure 3 shows a graphical representation of the piece distribution for each run from experiment PL-92-50. In these plots, we can observe that Mainline and Azureus achieve a balanced piece distribution much quicker (uniform gray area) than Transmission, where many pieces remain very rare for a long time (white horizontal bands). As a result, clients have to wait for these pieces before they can complete the download, leading to longer download times for all peers.

To effectively compare all experiments, we devised a more concise and quantitative representation for Δpcs. Intuitively, in a uniform distribution (where all pieces have roughly the same number of copies), the standard deviation from the mean number of copies ($\sigma_{\Delta pcs}$) is small, whereas in a non-uniform distribution, the standard deviation is large. As $\sigma_{\Delta pcs}$ is changing over the lifetime of the swarm, we integrate over that curve to obtain a single number that captures the uniformity of Δpcs for that experiment. Analogously, we use the median of the download times CDF ($P_{50}(t_{dl})$) as a single value to quantify client performance.

We compare both scores for all clients in every experiment (Table I) to find correlations. We can see a correlation for PlanetLab experiments: $\int \sigma_{\Delta pcs}(c_i) < \int \sigma_{\Delta pcs}(c_j) \Leftrightarrow P_{50}(t_{dl}(c_i)) < P_{50}(t_{dl}(c_j))$ where c_i, c_j are different clients from the same experiment. We cannot observe the same regularity on the cluster. Because the median download speeds of the faster clients are close together, the best client cannot be determined unambiguously. However, we can still see that faster clients tend to have lower scores than slower clients.

Our results allow us to quantify the relationship between download times and piece distribution. In particular, they support the thesis that a uniform distribution of pieces is an important factor affecting client performance.

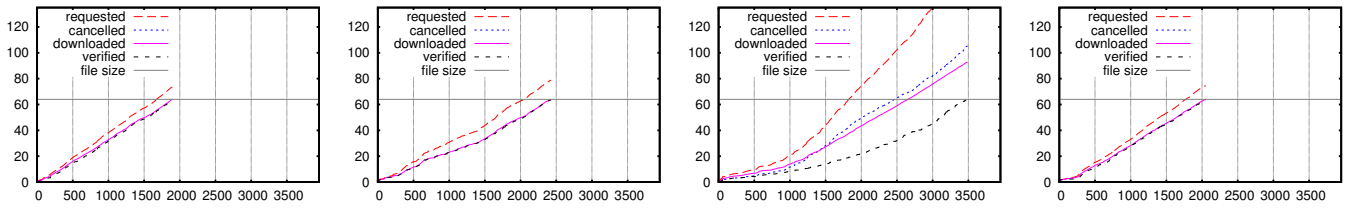
Seeding. In the previous section, we have established that a uniform Δpcs indicates higher overall download speeds. The initial *seed* (peer having the only copy of the file) has great influence over the development of Δpcs, as each piece of the file has to be uploaded by the seed at least once before it can start spreading in the swarm. This implies that the seed should avoid uploading redundant pieces when there are still pieces that have not been shared yet. However, the original BitTorrent specification includes no mechanism that would allow the seed to prioritize pieces. Instead, it is the piece selection strategy employed by the downloaders that determines how well the file pieces get distributed initially.

The detailed message information we have available allows us to determine (and plot) the *unique pieces* uploaded by the initial seed for each experiment. Figure 2(b) compares plots for the seeds from experiment PL-92-50. The point where the curve turns horizontal indicates the time at which the seed has uploaded one complete copy of the file. The line for the optimal piece upload represents the case where the seed uploads each unique piece only once until the first copy of the entire file has been distributed in the swarm.

Figure 2(b) shows that Mainline achieves the best seed behavior, and it is able to upload one complete copy of the file in $682s$. Azureus is able to upload 99.4% of the file in $990s$, however, it needs additional $615s$ for the remaining

TABLE I
RELATION BETWEEN MEDIAN DOWNLOAD TIME ($P_{50}(t_{dl})$) AND A PIECE DISTRIBUTION SCORE ($\int \sigma_{\Delta pcs}$). MINIMUM VALUES HIGHLIGHTED IN BOLD.

Experiment	Azureus		Mainline		Transmission		μTorrent	
	$P_{50}(t_{dl})$	$\int \sigma_{\Delta pcs}$	$P_{50}(t_{dl})$	$\int \sigma_{\Delta pcs}$	$P_{50}(t_{dl})$	$\int \sigma_{\Delta pcs}$	$P_{50}(t_{dl})$	$\int \sigma_{\Delta pcs}$
PL-35-5	**2,003**	**78,261**	2,253	206,849	N/A	365,579	-	-
PL-35-10	**1,863**	**63,398**	1,943	156,239	N/A	889,041	-	-
PL-35-50	1,861	128,833	**1,825**	**128,412**	5,519	789,100	-	-
PL-35-100	1,836	144,245	**1,779**	**109,507**	5,769	822,405	-	-
PL-92-5	**1,680**	**55,930**	1,895	117,739	4,300	481,036	-	-
PL-92-10	**1,460**	**38,822**	1,598	82,335	4,441	554,816	-	-
PL-92-50	1,435	60,637	**1,379**	**58,158**	3,351	355,816	-	-
PL-92-100	1,434	70,252	**1,363**	**41,032**	3,261	331,415	-	-
PL-512-5	**2,026**	**51,473**	2,228	100,490	2,748	133,273	-	-
PL-512-10	**1,635**	**38,492**	1,712	58,923	3,380	209,479	-	-
PL-512-50	1,799	57,813	**1,136**	**39,209**	1,933	140,731	-	-
PL-512-100	1,687	57,721	**1,127**	**31,567**	1,995	150,112	-	-
CL-35-5	**1,818**	87,070	2,049	258,910	N/A	**14,950**	1,851	134,087
CL-35-10	**1,805**	82,963	1,858	232,627	2,475	483,748	1,808	**66,052**
CL-35-50	1,773	**73,611**	**1,704**	170,325	2,444	461,583	1,777	101,240
CL-35-100	1,739	**78,728**	**1,713**	128,471	2,469	457,726	1,717	154,923
CL-92-5	1,444	**44,488**	1,554	92,738	1,861	166,858	**1,386**	68,538
CL-92-10	1,313	**43,166**	**1,308**	87,425	1,562	212,273	1,425	65,451
CL-92-50	1,255	**41,862**	**1,245**	65,210	1,481	191,311	1,340	45,678
CL-92-100	1,234	**43,207**	**1,224**	44,204	1,530	204,571	1,344	45,445
CL-512-50	1,061	**12,210**	1,090	28,525	N/A	103,044	**951**	18,849



(a) CL-35-50 (Azureus) (b) CL-35-50 (Mainline) (c) CL-35-50 (Transmission) (d) CL-35-50 (μTorrent)

Fig. 4. Pieces requested, downloaded and verified by one peer. [x = time in seconds; y = MiB]

pieces. Transmission requires $3,535s$ to complete the upload. This correlates with the scores from Table I. Interestingly, we found that Mainline has always the fastest seed behavior on Planetlab, but it is not always the fastest client. As the reduced performance is correlated with low connections, we analyzed the *connection management* of the different clients in more detail. One significant difference between Mainline and the other clients is the fact that Mainline has separate pools for managing outgoing and incoming connections, while other clients assign these dynamically from a single pool.

We compared the number, duration, and inter-arrival intervals of incoming and outgoing connections for the initial seed in each experiment. For a low connection limit, we observed the following: While Azureus, Transmission and μTorrent keep accepting new connections and dropping established ones, Mainline will not accept or open new connections when the set limit is reached. That causes the first few peers, able to connect to the seed, to monopolize it for a long time. This is an advantage for these clients but a disadvantage for the swarm, because it limits the total available bandwidth of the swarm during the ramp-up phase. Because the other clients change connections more frequently, they can distribute the same number of pieces to more peers and, thus, boost the available bandwidth in the swarm.

Piece Exchange. By analyzing the message flow in individual connections, we were able to identify another cause for the performance results of Transmission. Looking at absolute message counts (Table II) from all experiments on the cluster, we discovered that, compared to other clients, Transmission uses a disproportionally large amount of *request*, *piece*, and, in particular, *cancel* messages. These messages are used to request and transfer a piece of the file, and to cancel the request, respectively. A detailed piece exchange analysis revealed that Transmission frequently requests a chunk from more than one peer at the same time. Upon receiving a chunk from one peer, it sends *cancel* messages to the other peers. Unfortunately, this is often too late to stop the others from sending that chunk. Thus, the same chunk is downloaded multiple times. The BitTorrent specification recommends to use this strategy, dubbed *endgame mode*, only to quickly obtain the last few pieces of the file. Using it all the time, as Transmission does, wastes bandwidth, as illustrated in Figure 4(c). There, we can see that the amount of *downloaded* bytes is much larger then the amount of *verified* (with the SHA1 checksum from the

TABLE II
MESSAGE COUNTS (IN THOUSANDS) OVER ALL EXPERIMENTS ON THE CLUSTER

Message	Azureus	Mainline	Transmission	μTorrent
bitfield	1,478 (0.96%)	670 (0.35%)	313 (0.11%)	3,107 (1.23%)
have	28,640 (18.52%)	63,683 (33.74%)	56,822 (20.40%)	107,196 (42.36%)
request	60,014 (38.81%)	60,849 (32.24%)	99,185 (35.62%)	68,529 (27.08%)
piece	54,735 (35.39%)	55,819 (29.58%)	71,818 (25.79%)	55,800 (22.05%)
cancel	192 (0.12%)	500 (0.26%)	42,924 (15.41%)	145 (0.06%)
others	9,593 (6.20%)	7,214 (3.82%)	7,423 (2.67%)	18,254 (7.21%)
Total	154,652 (100.00%)	188,735 (100.00%)	278,485 (100.00%)	253,031 (100.00%)

.torrent file), and ultimately stored, bytes. Comparing that to the behavior of a the other investigated clients (Figures 4(a), (b),(d)), we see that these clients download exactly as many bytes as necessary and do not request unnecessary pieces.

V. RELATED WORK

Pouwelse [17] was the first to study BitTorrent and its rising popularity. Early research focused on analyzing the algorithms used in BT through simulation and analytical models (e.g., [2], [14], [18]). Empirical studies with instrumented clients were first described in [12]. [7], [13], [16] showed how individual peers could gain from unfair behavior, while [3] demonstrated the negative impact of such selfish behavior and [4], [8] proposed how to prevent it. Most of the above-mentioned research is based on flow-level simulations [2], [8], [14], or data collected by instrumenting a tracker [17] or a single peer [12], [16]. [11] and [15] describe small-scale experiments, such as running tens of clients on PlanetLab. The work most closely related to ours has been presented in [1], [6] where a platform similar to our BTLab Controller is introduced, however, without the rich analysis tool set we developed. Moreover, the platform requires the direct instrumentation of clients, making it impossible to study closed-source clients such as μTorrent. The only study we are aware of, that investigates differences between implementations, was carried out in [9]. However, we believe our approach is more comprehensive and gives deeper insight into the root causes of the performance results.

VI. CONCLUSIONS

To better understand the real-world behavior of BitTorrent clients and different interpretations of the protocol specification, we have developed BTLab. BTLab is a platform that allows a system-centric, data-driven analysis of the performance of real-world BitTorrent clients. We found that different clients showed significantly different performance. Moreover, particular settings or experimental parameters could result in a drastic increase of the download times for certain clients. Using our analysis toolbox, we were able to explain the underlying mechanisms that lead to the measured differences.

VII. ACKNOWLEDGEMENTS

This work has been supported by the Austrian Research Promotion Agency (FFG) under grant 820854 (TRUDIE), by the European Commission under grant № FP7-257007 (SysSec) and under the Prevention, Preparedness and Consequence Management of Terrorism and other Security-related Risks Programme (i-Code) and by Secure Business Austria.

REFERENCES

[1] Barcellos, M.P., Mansilha, R.B., Brasileiro, F.V.: Torrentlab: Investigating bittorrent through simulation and live experiments. In: IEEE Symposium on Computers and Communications (ISCC'08). pp. 507–512 (July 2008)

[2] Bharambe, A.R., Herley, C., Padmanabhan, V.N.: Some observations on bittorrent performance. In: SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (2005)

[3] Carra, D., Neglia, G., Michiardi, P.: On the impact of greedy strategies in bittorrent networks: The case of bittyrant. In: Intl. Conf. on Peer-to-Peer Computing (2008)

[4] Chow, A.L., Golubchik, L., Misra, V.: Improving bittorrent: A simple approach. In: International Workshop on Peer-to-Peer Systems (IPTPS) (2008)

[5] Cohen, B.: Incentives Build Robustness in BitTorrent. Tech. rep., bittorrent.org (2003), http://www.bittorrent.org/bittorrentecon.pdf

[6] Deaconescu, R., Rughinis, R., Tapus, N.: A bittorrent performance evaluation framework. Networking and Services, Intl. Conf. on pp. 354–358 (2009)

[7] Guo, L., Chen, S., Xiao, Z., Tan, E., Ding, X., Zhang, X.: Measurements, analysis, and modeling of bittorrent-like systems. In: 5th ACM Conference on Internet Measurement (2005)

[8] Huang, K., Wang, L., Zhang, D., Liu, Y.: Optimizing the bittorrent performance using an adaptive peer selection strategy. Future Gener. Comput. Syst. 24(7) (2008)

[9] Iliofotou, M., Siganos, G., Yang, X., Rodriguez, P.: Comparing bittorrent clients in the wild: The case of download speed. In: 9^{th} International Workshop on Peer-to-Peer Systems (IPTPS'10) (2010)

[10] ipoque: Internet study 2008/2009. http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009 (2009)

[11] Konrath, M.A., Barcellos, M.P., Mansilha, R.B.: Attacking a swarm with a band of liars: Evaluating the impact of attacks on bittorrent. In: IEEE International Conference on Peer-to-Peer Computing (2007)

[12] Legout, A., Urvoy-Keller, G., Michiardi, P.: Rarest first and choke algorithms are enough. In: IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (2006)

[13] Liogkas, N., Nelson, R., Kohler, E., Zhang, L.: Exploiting bittorrent for fun (but not profit). In: International Workshop on Peer-to-Peer Systems (2006)

[14] Luan, H., Tsang, D.H.K.: A simulation study of block management in bittorrent. In: InfoScale '06: 1st Intl. Conf. on Scalable Information Systems (2006)

[15] Marciniak, P., Liogkas, N., Legout, A., Kohler, E.: Small is not always beautiful. CoRR abs/0802.1015 (2008)

[16] Piatek, M., Isdal, T., Anderson, T.E., Krishnamurthy, A., Venkataramani, A.: Do incentives build robustness in bittorrent? In: NSDI (2007)

[17] Pouwelse, J.A., Garbacki, P., Epema, D.H.J., Sips, H.J.: The bittorrent p2p file-sharing system: Measurements and analysis. In: IPTPS (2005)

[18] Qiu, D., Srikant, R.: Modeling and performance analysis of bittorrent-like peer-to-peer networks. In: Conference on applications, technologies, architectures, and protocols for computer communications (2004)

[19] Stutzbach, D., Rejaie, R.: Understanding churn in peer-to-peer networks. In: ACM Conference on Internet Measurement (2006)