Exploring the Use of BitTorrent as the Basis for a Large Trace Repository

Anthony Bellissimo Brian N. Levine Prashant Shenoy

Abstract—

Motivated by the need to deploy a public repository of multi-gigabyte trace files, we studied the BitTorrent protocol's ability to disseminate very large files among peers. BitTorrent is a popular peer-to-peer protocol that allows parallel downloads of large files. In this paper, we analyzed user activity on BitTorrent over a four-month period with respect to supportable file sizes, file popularity, session lengths, transfer speeds, and the likelihood of service-interrupting flash crowds.

Our results show that file sizes tend to be on the order of gigabytes, far larger than other peer-to-peer applications. File popularity has a distribution similar to other peer-to-peer file sharing systems. Unlike other systems, the majority of users require multiple sessions to retrieve a file, and they are willing to remain connected to the system for a very long time. Most users we observed appear to have asymmetric Internet connections, and their generally poor upload performance is mitigated by their willingness to remain connected to the system and upload for an amount of time far longer than they spent downloading. We found that service disruption due to flash crowds is unlikely, as the vast majority of users were able to begin contributing resources back to the system within seconds of connecting. Our results indicate that BitTorrent provides an effective foundation for dissemination of files that are multi-gigabyte or larger, provided more sophisticated features are added like versioning, availability, and content management.

I. Introduction

PEER-TO-PEER systems provide a useful mechanism for disseminating large files to users. We are currently designing a system to disseminate very large files and data sets to the scientific community over the Internet. Our system will be primarily used for the *UMass Trace Repository.* The repository, which will be functional in late summer 2004, will house multiple terabytes of network, web, operating systems, and programming language traces. Many traces will be updated on a daily or weekly basis, as new data is gathered. In addition, we plan to use our system to distribute CDs and DVDs of popular open source software, such as Linux distributions, and to disseminate DVDs of lecture videos as part of our distance education effort.

Two existing methods are commonly used to support downloads of large files: *mirror servers* and *content distribution networks*. However, we do not expect volunteer mirror sites to be able or desire to replicate our entire multi-terabyte repository, nor do we have the resources to set up our own CDN or pay for a commercial CDN service. Therefore, in this paper, we explore a third approach: *community-based, on-demand mirroring*. This method harnesses relatively small amounts of storage and network bandwidth, donated by a large number of volunteer sites, to disseminate large amounts of data. When a user downloads any portion of the data set, they are given an option to share this data with other members of the scientific community. The protocol provides an incentive whereby users that share downloaded data are allowed faster downloads of subsequent data sets. Unlike mirror servers, not all content is replicated on all nodes. Further, unlike commercial CDNs, our community approach does not involve payments for providing or using this service. Parallel TCP downloads from multiple mirrors can also be used for better throughput.

BitTorrent is popular among Internet users for its ability to handle large file downloads. It differs from other peer-to-peer systems in that it is not intrinsically searchable or even unified—to download a file or a set of files, the user must first manually obtain a metafile. It also permits parallel TCP downloads. In addition, since at least one study has shown that users are not likely to contribute to a P2P system without any tangible reward, BitTorrent specifically rewards users who contribute resources to the system [12].

To understand the benefits and limitations of using BitTorrent as a building block for our system, we conducted a four month measurement study of popular BitTorrent sites. This paper presents the results of our measurement study. We analyze our results to identify performance issues. We present the lessons learned from our study and the implications of using such a protocol on the design of our system.

In sum, we find that (1) BitTorrent can support very large files. The median file size in our study was over 600MB and the maximum file size was over 10GB. (2) Session lengths in BitTorrent can last for several hours or days. We observed a mean session length of 13 hours and a maximum session length of 35 days. (3) BitTorrent's ability to support parallel TCP downloads and out-of-order downloads of chunks of a file can mitigate the effects of poor upload bandwidths at the peers. (4) The incentive mechanisms in BitTorrent and the ability of peers to quickly start sharing downloaded portions of large files helps dissipate the effects of flash crowds. Overall, we find that BitTorrent can be an effective foundation for disseminating very large files, provided it is enhanced with additional features such as content control, versioning, and availability.

The rest of this paper is structured as follows. We present the literature in the area and an overview of the BitTorrent protocol in Section II. We present our experimental methodology in Section III. Sections IV and V present the characteristics of files and peers seen in our measurement study. We discuss the lessons learned and the implications of these results on system design in Section VI. Finally, Section VII presents our conclusions and ongoing work.

II. Previous Work

In this section, we summarize related work, and provide an overview of the BitTorrent protocol.

Department of Computer Science, University of Massachusetts, Amherst 01003. Email: {twon, brian, shenoy}@cs.umass.edu .

A. Related Measurement Studies

There have been several measurement studies in recent years on peer-to-peer networks. A measurement study of Napster and Gnutella with respect to file popularity, file-type dominance, and node availability was presented in [1]. More recently, traces of open-Napster were used to evaluate the performance of the Chord protocol under real-world conditions [2].

An analysis of Gnutella traces in terms of resource demand, popularity of particular search terms, overlay topology, and node latency was presented in [7]. Ripeanu and Foster [10] also examined Gnutella data, focusing on node connectivity, overlay topology, and protocol message overhead. A trace analysis of the network traffic from the perspective of traffic characterization and bandwidth provisioning was presented in [13].

Saroiu, et al [12] analyzed Gnutella and Napster traffic with respect to node availability and the amount of bandwidth capacity individual peers could contribute to the system. The study found that if there is no incentive to do otherwise, users are inclined to falsely report their connection speed. Markatos [6] conducted a study of Gnutella protocol traffic aimed at caching query/response traffic to reduce network load. Leibowitz et al [5] examined Kazaa traffic to determine proportion of total network traffic by file popularity.

None of the above efforts have specifically focused on sharing and downloading of very large files, or specifically on BitTorrent. A recent study has specifically focused on BitTorrent, although in a somewhat different context [4]. The study is based on the observation of a single long-running torrent of RedHat Linux, whereas our study uses data from several thousand torrents of varying size. Though none of the torrents we observed were as popular as the RedHat torrent in [4], the variation in torrent size allowed us to observe differences in the behavior of connected users relative to the size of their torrents. In addition, we were concerned about some of its observations, in particular the apparently low success rate for downloading a torrent. Because a major goal of our system is load distribution, our system must ensure that the benefits gained by participating in the system and offering data to other peers outweigh the drawbacks. Our observations were substantially different, and we speculate about why in Section V.

Recently, there have been several efforts at constructing distributed storage and file systems from untrusted storage [9], [11]. The goals of our work are somewhat different from these efforts. We are interested in efficient distribution and parallel download mechanisms for very large files, while the above efforts have focused on issues such as availability, replication, and secrecy when using untrusted storage from volunteer sites. Similarly, unlike structured P2P systems such as Chord and CAN [15], [8] that focus on efficient search mechanisms and efficient content routing, we are less concerned about the ability to search across peers, since all content is always housed in the primary repository in our system.

B. BitTorrent Overview

BitTorrent is a peer-to-peer application that allows use of peers' unused upstream bandwidth to take load off a content-providing host. Its popularity is growing rapidly—a recent study revealed that while BitTorrent traffic was negligible in May 2003, it accounted for over 9% of the sampled traffic by October 2003 [14]. In BitTorrent, each file is split into *chunks*, and peers download not only from the peer with the initial, complete copy of the file (called a *seed*), but also download completed, checksummed chunks from each other. This substantially reduces the load on the seed, when compared with old versions of Napster, Gnutella, or KaZaA where all peers interested in a piece of content contend for bandwidth.

Peers downloading a particular file are tracked by a *tracker*, a piece of software that resolves queries and keeps track of what IP/port pairs are downloading a particular file. In BitTorrent, the term *torrent* is used to refer to a file or set of files with a common tracker entry and hash file.

To join a torrent, a user must download a metafile containing the tracker's URI, the file size of the torrent, the number of chunks, and SHA-1 hashes of each chunk. The client then connects to the tracker and requests a number of IP/port pairs of other peers. The client then opens a user-configurable number of TCP connections to these peers. When peers connect, they exchange information about what chunks of the file they have available. Peers can then request chunks from each other, and the data flows bidirectionally over the connection. If a user disconnects, she can reconnect later using the same metafile. Her client will reconstruct the list of completed chunks from the portions of the file already downloaded before querying the tracker for peers.

Once a chunk is completed, its SHA-1 hash is compared with the value in the metafile. If the values match, the client notifies each peer to which it is connected that it has completed that chunk. Thus, clients are kept constantly informed of what chunks are available to them. As may be evident from the above description, a file is not downloaded sequentially in BitTorrent—each chunk of the file can be downloaded independently by a client.

There are two popular algorithms for selecting which available chunk to download. One is for a peer to simply request a random chunk that its neighbor has. The other is for the peer to check its neighbors and request the chunk with the fewest copies. Once a user has the complete file, the client then drops connections with peers who are seeds, since they no longer need to download from each other. As long as the user keeps the client running, it will continue to distribute chunks of the file to other peers.

BitTorrent employs an incentive mechanism where peers prefer to send data to other peers that provide fast downloads. Each client has a user-configurable number of peers with which it is allowed to simultaneously exchange data. $N-1$ of those slots are reserved for peers with which the client is exchanging data the fastest at the moment. The last slot is reserved for a random peer, called the *optimistic unchoke*. This gives the client an opportunity to discover a peer potentially better than the ones from which it is currently downloading. These slots rotate if necessary every ten seconds, with the exception that the optimistic unchoke rotates every 30 seconds. In addition, in some popular client implementations peers that advertise data and do not provide it or provide it at extremely slow speeds (less than 1 kbps) are *snubbed*, and only limited data is exchanged with

TABLE I
TRACE STATISTICS

Period of Study	10/27/2003–1/16/2004
Total sessions	845,014
Total torrents	4,387
Total observations	24,822,391
Largest torrent	10.95 GB
Mean torrent size	760.12 MB
Median torrent size	651.77 MB
Longest session	35 days
Mean session length	13.25 hours
Median session length	8.46 hours

them. Currently, no analogous method is used to deny transfer to error-prone peers.

III. Experimental Methodology

To conduct our measurement study, we downloaded statistics from two popular trackers from October 27, 2003 until January 16, 2004. Each tracker continuously updates a public web page with various statistics on its torrents and connected users. The published statistics include user's random unique ID (one per user per session), the time the user had been connected to the torrent, and the number of bytes uploaded, downloaded, and remaining. For each torrent, we also retrieved its size and unique ID. Note that, since each peer is assigned a random ID for each session, the specific IP addresses of peers are not known. We downloaded the statistics page of each tracker every 30 minutes, parsed the HTML to extract all published information, and inserted this data into a relational database.

Table I summarizes the traces resulting from this measurement effort. As shown, our traces contain a total of 845,014 sessions observed across 4,387 individual torrents. Users invested a total of 800 session-years transferring data in sessions as long as a month. Torrents ranged in size from a few kilobytes to nearly 11 gigabytes. In total, more than three terabytes of data was available through these BitTorrent systems over the course of the study.

IV. Torrent Characteristics

A. Supportable file sizes

Very large files are supportable under BitTorrent.

Figure 1 reveals that BitTorrent downloads have a wide range in size. The largest torrent we observed was approximately 11 GB, with around 2% of all torrents at or above DVD-R capacity (4.38 GB). Only a very few, about 4%, were under 10 MB, a reasonable cap for common music files exchanged on many other peer-to-peer systems (A 5-minute song encoded at a constant 192 kbps is 7.2 MB).

B. Torrent popularity

Torrent popularity does not fit a Web model.

Figure 2 shows that the popularity of all torrents observed does not fit a common model (Zipf/power law). Had the data distribution been Zipf, the plot would have been linear on a log-log scale. Our previous studies have shown that requests for files are non-Zipf in P2P file sharing systems [1], and this was also seen by Gummadi et al. [3]. Our findings agree, showing a similar shape in a log-log plot of popularity vs. rank.

C. Scaling

The transfer rate scales well with the number of peers.

The primary argument for using BitTorrent is that it scales better than single-source downloads (e.g. from a non-clustered webserver). Figure 3 is an example of the correlation between connected users and aggregate bytes transferred. The existence of this correlation (around 90%) indicates that BitTorrent's performance does not degrade with an increased number of users connected to the system.

V. Peer Characteristics

A. Sessions

Most clients do not complete a download in one session.

Figures 4(a) and 4(b) show the percentage of a torrent peers bring to a new session, as well as how much of the torrent departing peers have when they leave. Note that only around 20% of new peers had not connected at all before. Around 15% of new peers had the complete torrent already, meaning they either acted as the initial seed or reconnected later to help their peers finish their downloads.

This shows that around two-thirds of sessions are returning peers: they received some data, disconnected, and then joined again under a new ID. Our study shows, as expected, that larger torrents require more sessions per user to download than smaller ones. For the approximately 5% of torrents in the region above 4 GB, the ratio of sessions during which a download has finished to all sessions is around 0.31. For torrents in the range between 1 GB and 4 GB, this ratio climbs to 0.34. For the smallest range of torrents we examined (0 to 200 MB) the ratio is approximately 0.55. Note that these ratios only count sessions during which a peer became a seed, and not sessions during which the peer joined as a seed. This is in contrast to Figure 4(b), which includes seed sessions.

In [4], individual sessions could be correlated because IP logs were available. In that study, only 19% of sessions were part of a transfer that eventually completed. Even the total of observed single-session downloads, multi-session downloads, and seed sessions was only 24% of the total. In contrast, Figure 4(b) shows that fully 50% of peers who left a torrent in our sample exited with the complete file, and this is without being able to discard sessions that would later resume and complete. We speculate that the sizable difference in observed behavior is due to the content of the torrents. If a user is dissatisfied with her performance while torrenting a Linux ISO, she can simply cancel the transfer and download it from a more reliable FTP or HTTP source. Most of the content in the torrents we observed was individual users distributing generally unavailable content in a way that distributed load, so there was no alternative distribution method on which a dissatisfied user could fall back.

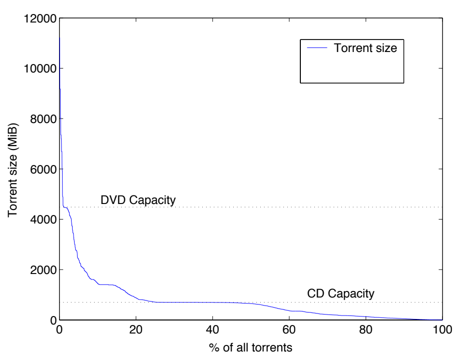


Fig. 1. Torrent sizes are widely distributed and large.

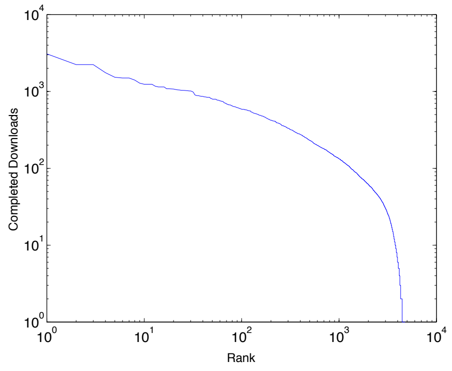


Fig. 2. Torrent popularity is non-Zipf.

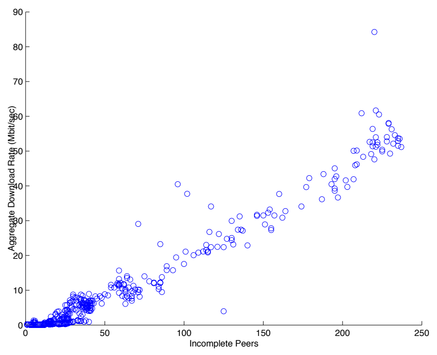
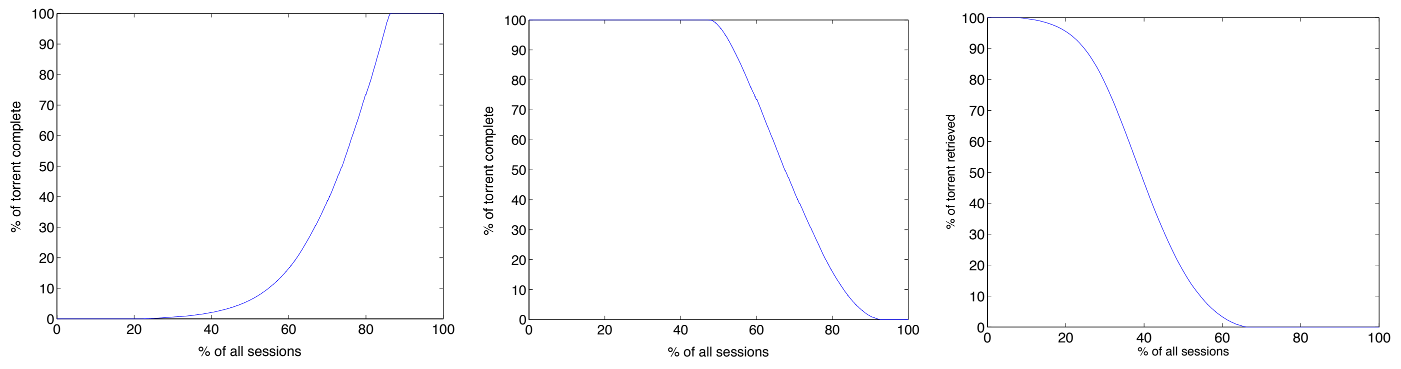


Fig. 3. Aggregate performance does not degrade as more peers connect to a torrent.



(a) Completeness of new peers at join. 15% begin as seeds.

(b) 50% of all sessions exit with the complete file.

(c) Relatively few peers obtain the entire torrent in one session.

Fig. 4. PDFs: Various statistics.

B. *Session lengths*

Sessions can last hours or days.

Since we have determined that users require multiple sessions to download most files, we can examine more closely the number of sessions they are likely to require. Figure 5 shows the proportion of the torrent retrieved in individual sessions, grouped by sizes of torrents. We see that fewer users tend to acquire the torrent in a single session for larger torrents, possibly suggesting a limit on how many bytes users are willing to transfer per session. (Figure 4(c) shows the same data, but in aggregate.) Overall, less than 10% of users obtained all of the data for their particular torrent during a single session. Figure 6 shows a similar graph, but is in terms of the ratio of bytes downloaded to torrent size. The discrepancy between the two data sets is explained by errors experienced during transfer. From this graph we can be certain that at least 4% of sessions experienced errors during a download, though the figure is almost certainly higher than that.

Figure 7 shows the distribution of amounts downloaded during single sessions in megabytes. More than 90% of the sessions we observed downloaded less than a gigabyte during the time they were connected.

A previous study of more conventional peer-to-peer systems such as Kazaa and Gnutella observed great patience on the part of users downloading content [3]. BitTorrent users have similar patience. Though we cannot map multiple individual sessions into a single download, we have observed individual sessions lasting as long as a month. 25% of sessions last for more than a day. Figure 8 shows the distribution of users' session lengths in hours. Interestingly, several of the longest-running sessions were spent unsuccessfully waiting for a seed to provide data.

Although the protocol allows users to report their connection time, some peers report extremely inconsistent data. We have been unable to determine whether or not this is a bug in some client implementations, but that is likely as peers have no incentive to lie about their statistics. Therefore, this graph is based on the amount of time for which the tracker kept a user's listing alive, rather than the user's reported connection time.

C. *Asymmetric connections*

Many clients' connections are asymmetric.

The typical user's home broadband connection is likely asymmetric. The data in Figure 9 appears to support this, with average upload speeds being noticably lower than download speeds. This not only implies asymmetry in connections but also that users spend more time uploading data than downloading it. This

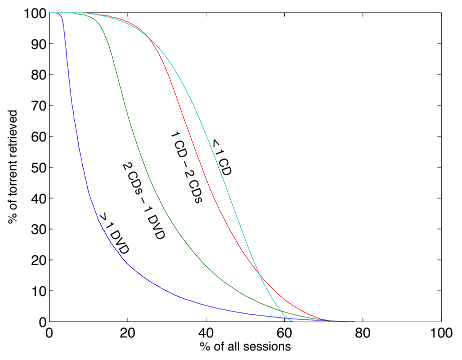


Fig. 5. Users obtain a larger portion of smaller torrents in one session.

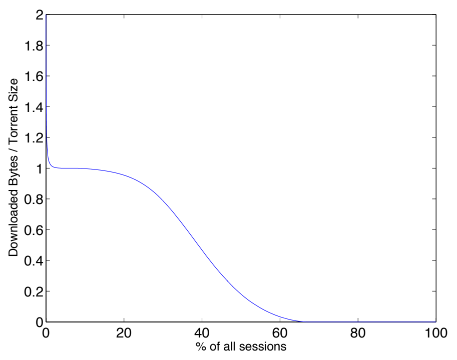


Fig. 6. A small percentage of users download many more bytes than the torrent size.

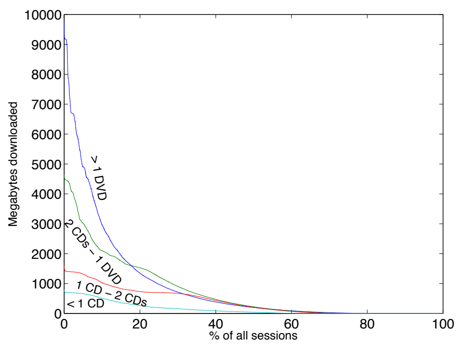


Fig. 7. Users retrieving larger torrents download more bytes per session.

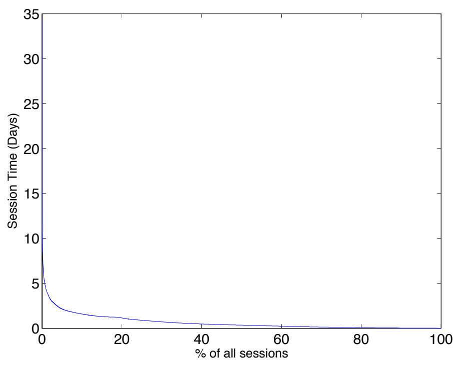


Fig. 8. Clients remain connected for very long periods of time.

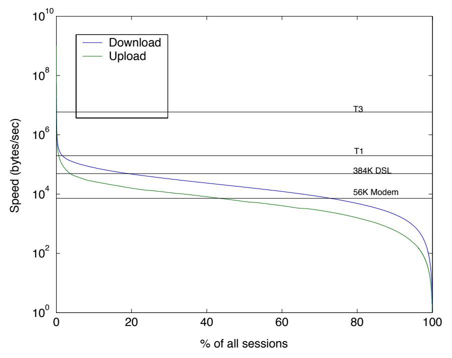


Fig. 9. Clients' connections tend to be asymmetric.

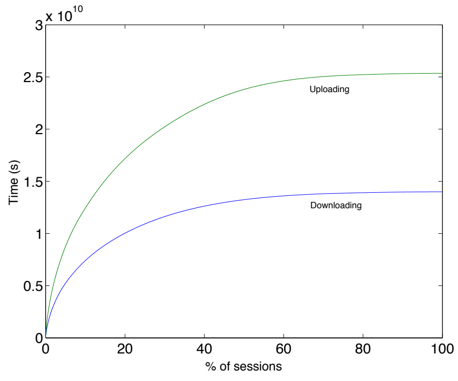


Fig. 10. Users spend twice as much time uploading as they spend downloading. (cumulative)

implication is confirmed in Figure 10—users contributed almost double the amount of total upload time in the system as download time. Note that upload time is defined here as the total time a user is connected to the system, while download time only includes the amount of time spent waiting for a download to finish. As we will see in the next section, it is reasonable to assume that upload time and connected time are equivalent.

D. Flash crowds

New content can result in a flash crowd but its effects are dissipated quickly by peers that share downloaded portions of the torrent within seconds.

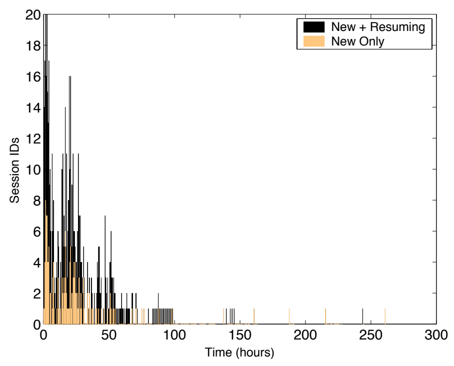
Figures 11(a)–11(c) show the rate of new connections to the torrent for the most popular torrent on four randomly selected dates in the sample period. The lighter part of the graph displays the number of new connections observed during the snapshot, while the darker part is the total of those connections with returning sessions (new IDs with partially complete content). We see that activity is generally highest toward the beginning of the torrent, and tapers off with time. We also observed that torrents do not tend to choke themselves while waiting for new peers to begin offsetting load—96% of all sessions were able to begin contributing back to the system in less than a minute, and 90% in less than 15 seconds.

VI. Implications of Our Results and Lessons Learned

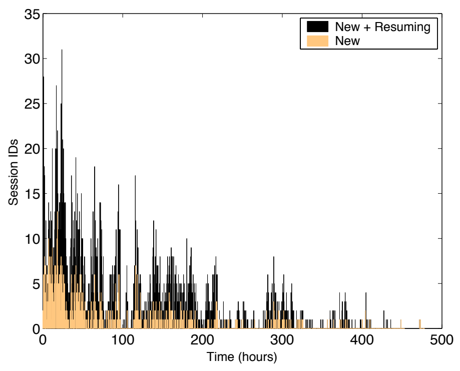
In general, our measurement study shows that BitTorrent can be an good starting point for our system based on the several factors. First, we find that BitTorrent can successfully serve very large files to its users. Our study showed that a non-trivial fraction of files are hundreds of megabytes or gigabytes in size. Second, the ability to parallelize download of a file and the ability to download chunks of a file out of sequence leads to good performance and dissipates load across peers. It is also effective at dissipating hot spots since downloading peers can quickly start sharing downloaded chunks and take on some of the load from the primary seed (thus, a peer need not wait until the entire file is donwloaded before sharing it with others). Third, the incentive mechanisms in BitTorrent are effective at encouraging content sharing—an important issue for our community-based mirroring approach to suceed. Last, we find that many users are willing to spend several hours and multiple sessions in order to download very large files. This is especially encouraging, since a community approach for disseminating large files cannot assume any single peer will contribute a large amount of bandwidth to the system.

However, BitTorrent lacks many features that are necessary in our system.

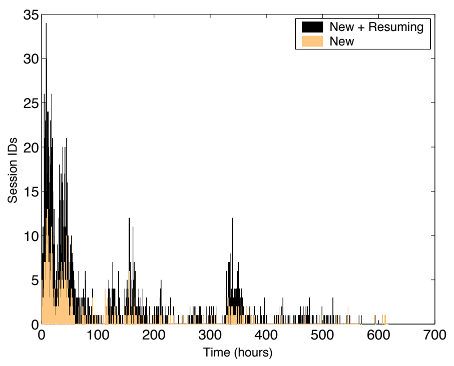
• *Content control:* One important issue in managing a large trace repository is content control. We would like the ability to assign version numbers to data sets as well as the ability to



(a) Torrent 1



(b) Torrent 2



(c) Torrent 3

Fig. 11. The rate of arrival to torrents is higher near the beginning than at the end.

replace an old or erroneous data set with a new file. Thus, it is important to prevent peers from sharing an old file that has been replaced by a new version. We plan to extend the protocol to add such content control features.

- *Reliable mirrors:* The architecture of our repository will include a small set of tightly-coupled sites that mirror content from the main repository. Each (reliable) mirror then shares its data sets with (less reliable) peers. Since the main repository can be a single point of failure, such a hybrid architecture ensures greater availability of the data. BitTorrent will need to be enhanced to account for such a two tier hierarchy; for example, to enable proactive replication across the mirrors as opposed to on-demand replication at peers.
- *Transparent failover:* We plan to enhance BitTorrent to automate the process of resuming failed sessions. This will reduce the amount of manual intervention necessary for downloading large files over long sessions.
- *Searching:* Although BitTorrent is not inherently searchable, in our system, all files are always available on the primary repository and its mirrors. Consequently, a simple, content-side search engine to index data sets based on content and user-provided keywords should suffice for our purpose.

Finally, we note it is unlikely we will gather stable partnering mirror sites if they have to assist in all downloads. We plan to extend the BitTorrent clients run at mirror sites to appear as seeds only if another seed does not current exist in the system. Similarly, stable mirror sites could offer only chunks currently unavailable from downloading peers. This way, mirrors ensure availability of files, but provide bandwidth only when necessary.

VII. Conclusions and Ongoing Work

As part of ongoing work, we are using the insights from our study to design a BitTorrent-based system for our trace repository. We plan to deploy our repository on a server at our univerity and will explore the use of PlanetLab to house a small number of reliable mirrors. A design of our system will be completed in the next few weeks and a prototype implementation will be completed by late summer. The trace repository will be functional shortly thereafter.

The traces gathered for this measurement study will be the first set of traces disseminated on our trace repository. Over the next year, traces of network packet headers, measurement data from a 802.11b network, disk I/O traces, web traces and memory behavior of Java programs will also be made available.

References

[1] J. Chu, K. Labonte, and B. N. Levine. Availability and locality measurements of peer-to-peer file systems. In *ITCom: Scalability and Traffic Control in IP Networks II*, July 2002.

[2] Jacky Chu, Kevin Labonte, and Brian Levine. Evaluating the use of chord with real-world peer-to-peer traces. Technical report, 2004. Submitted to IMC 2004.

[3] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 314–329, 2003.

[4] Mikel Izal, Guillaume Urvoy-Keller, Ernst W Biersack, Pascal A Felber, Anwar Al Amra, and Luis Garces-Erice. Dissecting BitTorrent: five months in a torrent's lifetime. In *PAM'2004, 5th annual Passive & Active Measurement Workshop, April 19-20, 2004, Antibes Juan-les-Pins, France*, Apr 2004.

[5] Nathaniel Leibowitz, Matei Ripeanu, and Adam Wierzbicki. Deconstructing the kazaa network. In *Proceedings of 3rd IEEE Workshop on Internet Applications (WIAPP '03)*, San Jose, CA, USA, June 2003.

[6] Evangelos P. Markatos. Tracing a large-scale peer to peer system: an hour in the life of gnutella. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

[7] D. Nogueira, L. Rocha, J. Santos, P. Araujo, V. Almeida, and W. Meira Jr. A methodology for workload characterization of file-sharing peer-to-peer networks. In *IEEE Workshop of Workload Characterization*, pages 118–126, 2002.

[8] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, 2001.

[9] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The oceanstore prototype. In *Proceedings of the Conference on File and Storage Technologies*. USENIX, 2003.

[10] M. Ripeanu and I. Foster. Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems, 2002.

[11] Timothy Roscoe and Steven Hand. Palimpsest: Soft-capacity storage for planetary-scale services. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS-IX), Lihue, Hawaii, CA, USA*, May 2003.

[12] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking (MMCN)*, January 2002.

[13] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. In *Proc. ACM Internet Measurement Workshop*, 2002.

[14] Stanislav Shalunov. Internet2 netflow weekly reports, 2003.

[15] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. pages 149–160.