Dynamic File Bundling for Large-scale Content Distribution

Song Zhang[†], Niklas Carlsson[§], Derek Eager[‡], Zongpeng Li[†], Anirban Mahanti[SS]
[†]University of Calgary, Canada
[§] Linköping University, Sweden
[‡] University of Saskatchewan, Canada
[SS] NICTA, Australia

***Abstract*—One highly-scalable approach to content delivery is to harness the upload bandwidth of the clients. Peer-assisted content delivery systems have been shown to effectively offload the servers of popular files, as the request rates of popular content enable the formation of self-sustaining torrents, where the entire content of the file is available among the peers themselves. However, for less popular files, these systems are less helpful in offloading servers. With a long tail of mildly popular content, with a high aggregate demand, a large fraction of the file requests must still be handled by servers. In this paper, we present the design, implementation, and evaluation of a dynamic file bundling system, where peers are requested to download content which they may not otherwise download in order to "inflate" the popularity of less popular files. Our system introduces the idea of a *super bundle*, which consists of a large catalogue of files. From this catalogue, smaller bundles, consisting of a small set of files, can dynamically be assigned to individual users. The system can dynamically adjust the number of downloaders of each file and thus enables the popularity inflation to be optimized according to current file popularities and the desired tradeoff between download times and server resource usage. The system is evaluated on PlanetLab.**

I. Introduction

Characterization studies of file popularity [1]–[5] have shown that there typically is a long tail of mildly and less popular content. Companies like Amazon (books) and Apple (music) have leveraged this long tail of niche content to increase sales of more popular items as well. Efficiently serving the long tail can have significant monetary impact, and may directly impact the bottom line of many companies. With peer-assisted approaches being an attractive content distribution solution for organizations with limited server and/or bandwidth resources, an important problem is how to best leverage the upload contributions of peers to serve a large collection of files, including both popular and less popular files.

Peer-assisted approaches help offload the original content servers. By splitting each file into many small pieces, each piece can be downloaded from different peers and/or servers. The approach allows flexible distribution paths, based on current piece availability, and is highly effective for popular content, for which the request rates enable the formation of self-sustaining torrents, where the entire content of the file is available among the peers themselves [6], [7]. On the other hand, the approach does not efficiently serve the contents in the long tail, for which the request rates are not sufficient for the corresponding torrents to be self-sustaining, and the contents must instead be served primarily using server (or seed) bandwidth.

Bundling has been proposed to improve the content availability, and increase the set of peers that can provide upload bandwidth to other peers downloading the same content [8]. With bundling, a set of contents are grouped (bundled) into a single file which is made available for download. Thus, peers downloading a bundle containing the desired file(s), effectively help inflate the popularity of the other files in the bundle.

Both static [8], [9] and dynamic [7] bundling approaches have been proposed. With *static* bundling, a pre-determined file collection is grouped together by the publisher. In contrast, with *dynamic* bundling, peers may be assigned complementary content (files or parts of files) to download at the time they decide to download a particular file. This additional flexibility provides many advantages. First, dynamic bundling can adapt to current popularities and avoid wasting download resources in cases when the content is popular and popularity inflation is not necessary for the purpose of content availability. With static bundling, however, peers are forced to download all the content, resulting in increased download times, even when this does not benefit the system. Dynamic bundling also allows peers downloading the same popular file to help in the distribution of different less popular files. This is an important property as file popularities typically are highly skewed.

In this paper, we present the design, implementation, and evaluation of a dynamic file bundling system. Our system is implemented using the mainline BitTorrent source code, and builds upon the BitTorrent specifications. We focus on developing a dynamic bundling system while only making incremental changes to the BitTorrent source code. This presents interesting design challenges as much of BitTorrent's communication and file handling policies (such as disk writing and piece validation) rely heavily on all peers in the same swarm sharing exactly the same content and having a common indexing of particular data blocks. In contrast, a dynamic solution should allow peers to effectively share content between peers that have been assigned different bundles from a set of files hosted by the tracker.

Our system introduces the idea of a *super bundle*, which

NOTICE: This is the authors' version of a work that was accepted for publication in IEEE LCN 2012. The final/official version will appear the conference proceedings and IEEE Xplore.

consists of a large catalogue of files. From this catalogue, smaller bundles, consisting of a small number of files, can dynamically be assigned to individual users. We refer to this smaller file collection as the peer's *individual bundle*. The use of dynamic assignment of individual bundles enables the relative file availabilities to be adjusted. Our system leverages tracker information about peer participation in the different files to make informed decisions about which files should be included in each peer's individual bundle. Of course, the larger each super bundle is the more flexibility is given to the dynamic bundling policies assigning individual bundles to clients. Architecting a system where bundling decisions are made from an arbitrary set of files hosted by the tracker is left for future work.

At a high level, the primary modifications to the mainline client include modifications to peer-to-tracker communication, the addition of a bundle file selection mechanism at the trackers, a generalized piece selection rule, and optimizations to the piece disk writing function. First, we modified the peer-to-tracker communication and how the tracker handles incoming HTTP requests from the peers, including the introduction of a new *bundle negotiation event*. Second, we implemented two bundle file selection policies for determining which files should be bundled, which take the current file popularity into account. Third, we modified the piece selection rule and the actions taken when receiving HAVE messages in different peers states (depending on if the peer is choked or unchoked, for example) to give priority to the pieces from the requested file and ensure that each peer only expresses interest in the pieces from within its assigned bundle. Finally, we changed the piece writing sequence to disk, to accommodate for the changes to the scope of our modified piece selection rule that only downloads pieces from a peer's assigned bundle (containing a subset of all potential pieces of the super bundle). Without these disk writing modifications, the downloads can become very slow and in some cases not complete.

In this work, we do not attempt to determine the best possible dynamic bundling policy. Instead we focus on the system design, present a proof-of-concept implementation, and evaluate the general system performance using two example policies for dynamic bundle selection. Our system is evaluated on PlanetLab. Using both steady-state and time-varying scenarios, the operation of our design is verified and the performance of the example policies is evaluated. Our system shows significant improvement in the download times of less popular files, compared to a non-bundling implementation. Our work is novel in that no previous work has developed and/or evaluated a working dynamic bundling system. Throughout the paper we provide insights on lessons learned while undertaking this challenge.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 presents our system design. System validation and performance results are presented in Section 4. We conclude the paper with a discussion of the lessons learned and future work directions.

II. Background and Related Work

This section provides a brief overview of content distribution, peer-to-peer systems, and the BitTorrent protocol.

Content distribution and peer-to-peer: Content delivery is responsible for a large fraction of today's Internet traffic. Content delivery over the Internet has typically relied on dedicated servers and data centers. To offload the origin server(s) and to provide enhanced quality-of-service, a variety of techniques such as replication [10] and peer-to-peer [11] systems have been proposed. With replication, clients are typically directed to one of many geographically distributed servers that share the load of processing client requests. This approach is taken by CDNs such as Akamai. With peer-to-peer techniques, clients may contribute to the total service capacity of the system by helping other clients. This can further reduce the load of the original server.

BitTorrent: Perhaps the most promising peer-to-peer technology for content delivery thus far is BitTorrent [11]. By splitting files into small pieces, BitTorrent allows pieces to be downloaded from multiple peers in parallel, significantly improving the file-sharing efficiency [12]. BitTorrent-like approaches have also been used successfully for streaming [13].

A BitTorrent client typically downloads a file by first obtaining a .torrent file from a Web server. The .torrent file has information about the tracker(s) that maintain state information about the peers currently downloading the file; this set of peers is referred to as a swarm. In a swarm there may be both leechers (which currently are downloading the file) and seeds (which have a complete copy of the file). The .torrent file also contains information that allows the client to validate the correctness of the downloaded pieces, as well as the info hash value for the file. The info hash is a unique identifier for the file and can be used in communications with both the tracker and with other peers. Each peer typically establishes connections to a large set of peers with which it can then exchange file pieces. This set of peers is typically referred to as the peer's peer set. For efficiency, at each point in time, a peer typically only uploads pieces to a smaller subset of its peer set [14].

Content availability: While efficient for popular content, the BitTorrent system exhibits the content availability problem [15]–[17]. In particular, swarms with fewer simultaneous file-sharing participants experience performance degradations with respect to download times [6], [18]. With a long tail of less popular files [1] this can be a significant problem.

There has been some preliminary work studying this problem, without providing specific solutions. One of the most promising solutions is to request that peers help in the download of other files [7], [8], [16], [19]. To the best of our knowledge, the work by Menasche *et al.* [8], who apply a queuing model and static bundling experiments to illustrate the problem, and a simulation-based evaluation of dynamic bundling policies by Carlsson *et al.* [7] are closest to our current work. We are not aware of any works that design, implement, and evaluate a prototype system.

III. System Design

This section describes the design of the Super Bundle system. A preliminary version of the design was reported in a work-in-progress paper [20].

A. Design Overview

Our system design had three primary goals. First, the system should allow clients to assist in the delivery of files other than those they request. Second, the system should be flexible and be able to dynamically assign files to peers for the purpose of assisting other peers. Finally, the changes to the BitTorrent client should be kept at a minimum and allow for incremental deployment. Our clients and trackers must therefore gracefully interact with regular peers and trackers.

Our system introduces the idea of a *super bundle*, which consists of a large catalogue of files. Clients request individual files from this super bundle. In addition, based on communication with the tracker, peers are instructed to download additional files from the super bundle. The subset of super bundle files assigned to a peer is referred to as an *individual bundle*. Our tracker adjusts individual bundles as required by the implemented policy. For instance, the tracker can dynamically adjust the bundles to adapt to the relative file availabilities. In contrast with static bundling techniques [8], our system can adapt to changes in demands.

Our implementation builds upon the mainline BitTorrent structure and specifications and makes revisions as necessary. To enable an efficient system that can make informed decisions about which files a new peer should serve in addition to its requested file, a number of BitTorrent components must be modified. The primary modifications include changes to peer-to-tracker communication, the addition of a bundle file selection mechanism at the trackers, a generalized piece selection rule, and optimizations to the piece disk writing function.

- The communication protocol between the local peer and the tracker must be revised. (We use "local" peer to refer to the peer on which our BitTorrent software is running.) For each peer, the tracker should be able to create an individual bundle consisting of the file the peer wants to download and the other files it must download according to the tracker's bundle selection policy.
- The tracker's functionality will change, with addition of a bundle selection policy to determine which files constitute a peer's individual bundle.
- The piece selection policy and actions taken when HAVE messages are received must be revised. Peers should only download files that are part of their individual bundle.
- The piece disk writing algorithm must be revised. The piece priority in the disk writing algorithm must be modified so that the pieces of interest can be written to the disk with high priority.

Overall, our design will supplement the current system and will not affect the mainline BitTorrent system mechanism. Of course, the clients could easily be extended to include additional mechanisms to prevent free-riding or be equipped with software (provided by the content provider, for example) that monitors the peers' participation. The following sections discuss the first three of the aforementioned implementation issues. The reader is referred to [21] for details. A preliminary version of our design was also presented in a three page work-in-progress paper [20].

B. Peer-to-Tracker Communication

BitTorrent relies on two types of communication protocols: peer-to-tracker and peer-to-peer. The peer-to-tracker communication, described in this section, is done using the HTTP protocol. Peers inform the tracker about which files they are interested in (using the file info hash) and listening ports. The tracker maintains information about current peer participation, uses incoming messages to update its peer information, and returns information about other downloaders to the local peer. To ensure up-to-date information, communication is periodic.

Basic updates: The basic HTTP-based communciation between a peer and the tracker in the mainline BitTorrent system has four "event" types: `started`, `no event`, `completed`, and `stopped`. A peer sends a `started` message to the tracker on beginning the download, and a `completed` message on finishing a download. While downloading, the peer uses the `no event` message to periodically update the tracker. A `stopped` message is used when a peer is gracefully shut down. These four messages and their corresponding events have individual processing modules on the tracker. The information in the messages is used to update the tracker's information about the participation in the swarms.

Bundle negotiation and selection: We introduce a new `bundle negotiation` event and devote to it a separate processing module and message exchange. A peer uses a `bundle negotiation` message to expresses its interest in one or more files. The modified tracker uses this information together with information on other peers currently downloading the different parts of the super bundle, to determine which additional files the peer should also download, creates an individual bundle for the peer, and communicates this information back to the peer.

The constituent files of each individual bundle are determined using a dynamic bundle selection policy. This policy selects the supplementary bundle file(s) during the bundle negotiation stage and is implemented in the processing module for the `bundle negotiation` event at the tracker. To facilitate dynamic bundling, the tracker maintains a mapping of the files (within the super bundle) and the peers associated with each of these files. For such record keeping, a new status file is created and maintained on the tracker. The new status file is updated when the `started` and `stopped` events are received by the tracker. The file allows the tracker to easily obtain the number of peers currently downloading or seeding a specific file. For the purpose of our evaluation we will use two basic example policies.

C. Peer-to-Peer Comunication

Peer-to-peer communication is used to exchange information about which pieces peers have downloaded, as well as exchange the pieces themselves.

Communication overview: The Peer Wire Protocol (PWP) is used for communication. A peer establishes a TCP connection with each peer in its peer set. After connection establishment and a successful PWP handshake, the peers can exchange piece information. Blocks of pieces are uploaded and downloaded over these connections. Different blocks of the same piece can be obtained from different peers. After download of a complete piece, a peer assembles the blocks into a piece and sends a HAVE message to all its neighbors.

Unchoke policy: While peers maintain connections with all peers in their peer set, for efficiency reasons, at each point in time, the local peer typically only uploads pieces to a subset of these peers. This subset is said to be unchoked and is determined using BitTorrent's rate-based tit-for-tat policy. By giving upload preference to the peers that upload pieces to the local peer at the highest rates, this policy provides incentive to upload content to the local peer. To incentivize peers downloading different contents to share pieces and ensure high piece sharing efficiency, we apply the unchoke policy on the entire bundle rather than on a per-file basis.

Piece selection policy: When unchoked, each peer must select which piece to download next. At a high level, we use a slightly modified version of BitTorrent's rarest-first policy, which gives preference to the pieces that are the least replicated among the peers within the local peer's peer set and belong to the local peer's individual bundle. We make use of a piece index filter which requires minimal changes to the piece selection policy. The peer simply requests and downloads only pieces belonging to files in its bundle.

At a detailed level, the modifications to the piece selection policy must take into account the status at the local peer when it receives a HAVE message from the remote peer. In the case that the local peer is choked when it receives a remote HAVE message, applying our filter is trivial as we can pick the rarest piece among the set of pieces that the peer is interested in and the other peer has at the time the peer becomes unchoked.

On the other hand, if the local peer is already unchoked, the default BitTorrent behavior is for the peer to directly request the piece whose index is contained in the HAVE message from the remote peer provided the peer does not already have the piece. However, in our system the local peer may not be interested in the piece (as it may not be part of its individual bundle). We therefore modify this policy to be based on interest.

D. Implementation

Our dynamic bundling system is built using the mainline BitTorrent (version 3.3) source code. Although there have been changes and improvement in later versions, version 3.3 provides the most clear structure among the open source versions we examined. Version 3.3 includes all the main features of BitTorrent and it strictly follows the BitTorrent specification. This allows our high-level modifications to easily be applied to later versions as well. Overall, our bundling approach is relatively independent of the exact version of the client software, and our choice of using version 3.3 for our experiments should not impact the relative bundling performance in any significant way.

IV. Design Evaluation and Validation

A. PlanetLab Setup

PlanetLab is mainly used for academic research. The PlanetLab platform consists of computers distributed all over the world. As of 2011, there were over 1,000 nodes at about 500 participating sites.[1] Each PlanetLab project is assigned a slice, which effectively provides access to a virtual machine on some set of nodes/machines across the world.

In our experiments, a local Linux machine with a public IP address served as the management host for the experiments, as well as a tracker and seed.

We use PlanetLab tools such as `CoMon`[2], `pssh`[3], and `pslurp` to distribute our software, manage experiments, and collect results. The `CoMon` tool is used to obtain nodes that satisfy desirable conditions. We use `CoMon` to obtain information about the workload of each node. From the set of lightly loaded hosts with good response time, we randomly select a subset for our experiments. We use the `pssh` software to push our modified source code to the selected nodes, and to run commands remotely. Finally, we use `pslurp` to collect the log files generated during experiments back to our local host machine, where results are processed and analyzed.

B. Experimental Design

To evaluate the performance improvement with our super bundle approach, we ran both steady-state and dynamic transient experiments. In all experimental setups, the peers tell the tracker which file of the super bundle they are interested in, and the tracker tells the peers which files to download. In contrast to a regular BitTorrent system, each file now corresponds to a part of a larger super bundle, and the tracker keeps track of the number of downloaders of each file. In all experiments, we assume that the peers leave the swarm as soon as they finish downloading, without serving as a seed.

Bundling policies: We compare the performance of three different tracker policies for creating individual bundles: (i) no bundling, (ii) balanced bundling, and (iii) random bundling. Approach (i) corresponds to the case where no bundling is applied, and peers download nothing but the content they are genuinely interested in. With the other two policies, any peer requesting a file may also be asked to participate in the distribution of one additional file, in addition to its requested file. This file is selected from the super bundle such that there exists at least one downloader but no more than some threshold number of downloaders (e.g., 10). If no such additional file

[1] PlanetLab, http://www.planet-lab.org/

[2] CoMon software, http://comon.cs.princeton.edu

[3] Pssh software, http://www.theether.org/pssh/

exists, no additional file is assigned to the peer, and the peer only downloads the file it initially requests. With the random policy, the tracker picks an additional file at random. With the balanced policy, the tracker picks the file that has the fewest downloaders, with ties broken at random.

Experimental setup: Each experiment employs between 60-70 PlanetLab nodes, distributed across the world and selected as described above. We use the same set of nodes for each experimental comparison, and to reduce PlanetLab node status variations, we finish each round of experiments for comparison on the same day. As we are interested in the download time characteristics of the above three approaches, we ensure that each node has downloaded the .torrent file of the super bundle, generated from all the files, prior to the experiment. The .torrent file is created using 10 files, each about 20 Mbytes. The size was selected to accommodate many downloads per node, without reaching any bandwidth thresholds set by PlanetLab. For all experiments, we use a single seed and a single tracker, with both functions performed by our local Linux host. Finally, to avoid external perturbations, such as outside peers, we keep the torrent private.

For each experiment, we developed one or more control scripts that we run in parallel on the different nodes. At each node, the scripts determine when nodes should download a file, and the duration of time between downloads. For the purpose of our experiments, each peer always stops participating at the time it has completed download of its requested file, at which point it checks the correctness of the content and deletes all its downloaded content (so that it will not have any content to share at the beginning of its next file download).

During an experiment, we first start the tracker and the seed on the local host, and then use `pssh` to invoke the script(s) on the remote nodes. The scripts collectively control the experiments, including how many rounds the PlanetLab hosts should run the BitTorrent client software. At the end, the log files are collected to the local host using `pslurp`, where they can be further analyzed. The seed has an upload throughput limit of 800 kBps. Other nodes in the experiments do not have any imposed limit for upload or download throughput.

Performance measurements: Each BitTorrent client and the tracker are instrumented to record various items of information. For each file download, the peers record statistics such as download time, download and upload rate throughout their download sessions, the time when each block of the file is downloaded, as well as who provides this block. The nodes on PlanetLab generally have unique public IP addresses, allowing us to distinguish each node by its IP address (the IP address is contained in BitTorrent messages). The control scripts generate a log file that keeps track of the high-level statistics of what is done by each node. For example, which file ended up being downloaded, how long did it take, and how long did the node sleep between downloads. At each point in time, the tracker records how many peers participate in the download of each file. At the end of each experiment, the log files and statistics are collected to our local host, where we can analyze the traces. We next present a selected set of analysis results.

C. Steady-state Download Delay Analysis

As a proof-of-concept analysis, we consider two simple steady-state scenarios. In both scenarios, we have a single super bundle containing one popular file and nine less popular files. More specifically, we select the file popularity such that the popular file is fifty times more popular than the other files.

In the first scenario, we let each node randomly choose its interested file every time it begins to download. We call this scenario *random, steady state*. In the second scenario, each node is assigned a fixed file as its interested file for each of its downloads. We call this scenario *static, steady state*. Clearly, there will be more variability in the first of these scenarios.

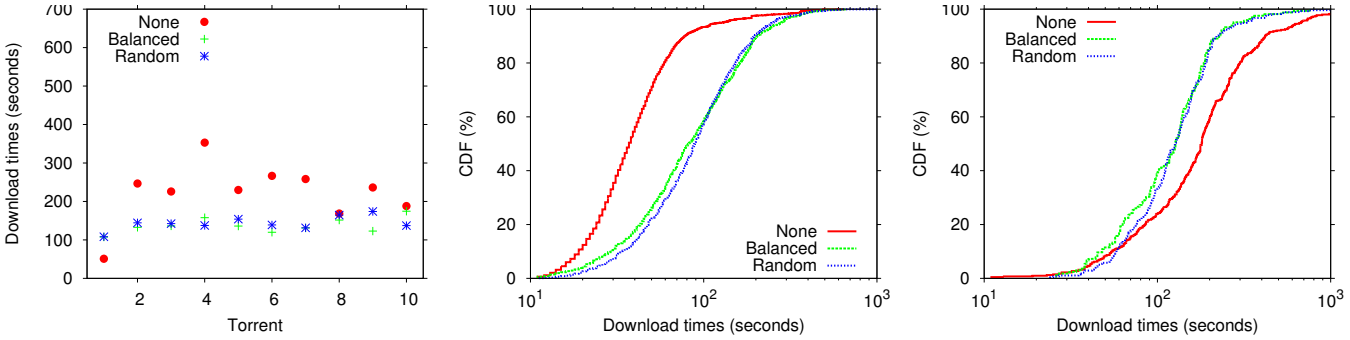
Figures 1(a) and 2(a) show the download times for the three policies, for each of our two scenarios, respectively. For each file (labeled on the x-axis), we show the average download time in seconds (y-axis). Figures 1(b) and 1(c) show the download time distribution of the popular file and the less popular files, respectively, for the random, steady-state scenario. Figures 2(b) and 2(c) show the corresponding distributions for the static scenario.

We note that the two bundling policies typically see substantially improved download times for the less popular files, compared to the download times in the non-bundled experiments. While this comes at the (small) cost of a slightly increased download time for the popular file, we believe that this improvement in the availability of the less popular files (files 2 to 10) is desirable, as it significantly helps offload the seed. This is caused by the popular file peers helping other peers by downloading less popular files in the super bundle, and thus offloading the seed. While neither the average delay metric or the delay distribution provides a clear winner between the balanced and the random bundle selection policies, both policies significantly improve the download time of the less popular content.

Comparing the two scenarios themselves, we note that much of the differences in download times between the two scenarios (particularly without bundling) is due to the random scenario having a larger fraction of active peers that have requested the less popular files. In the static scenario there will on average only be a fraction 9/59 of the peers downloading these files. In contrast, in the random scenario, the longer download times of these files will cause this fraction to be higher.

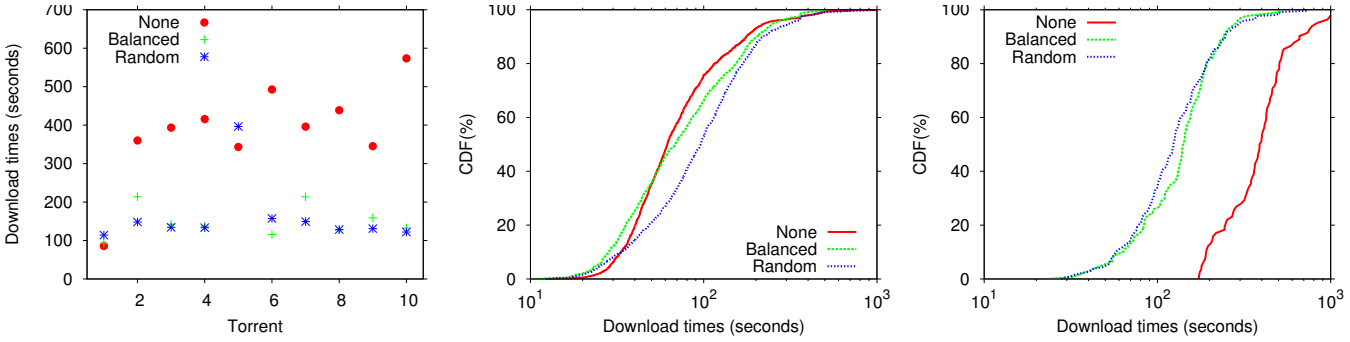
D. Steady-state Download Progress

Let us now have a closer look at one of our bundling policies. Figures 4(a) and 4(b) show the download progress using the random bundling policy for peers downloading the popular file and the less popular files, respectively. Corresponding results without bundling are shown in Figures 3(a) and 3(b). While the time to download a certain percentage of the file is consistently better for the popular file, for both popular and unpopular files the relatively evenly distributed spacings of the curves suggests that the download progress for both types of files is quite steady, when bundling is employed. (Note the log-spacing of curves, as well as the log scale of the x-axis.) However, the corresponding lines for the case in which no



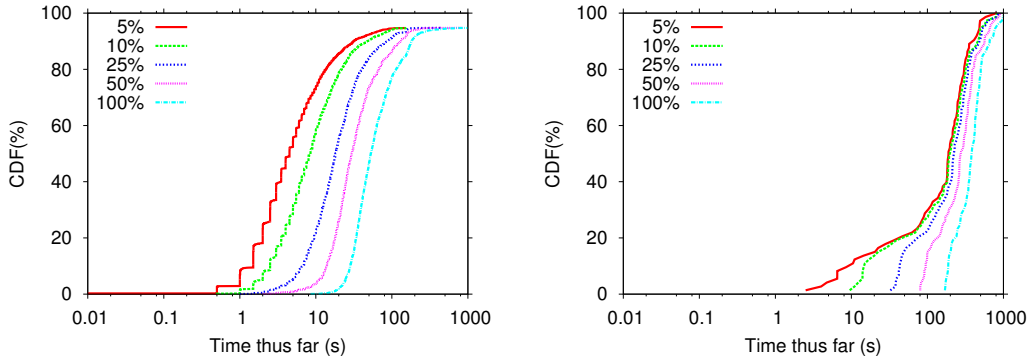
(a) Average (b) CDF popular (1) (c) CDF non-popular (2-10)

Fig. 1: Download time in the random, steady-state scenario.



(a) Average (b) CDF popular (1) (c) CDF non-popular (2-10)

Fig. 2: Download time in the static, steady-state scenario.



(a) Popular (1) (b) Non-popular (2-10)

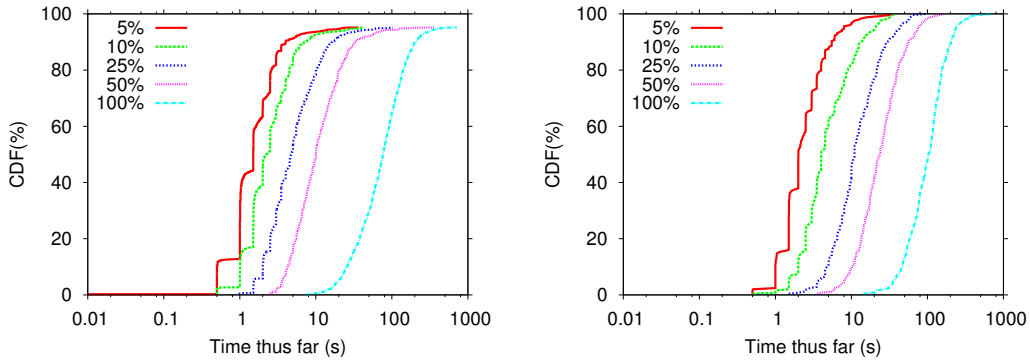
Fig. 3: Download progress without bundling. Download progress measured by the time it takes to download a certain percentage of the file content.

bundling is applied (see Figure 3(b)) are much more squished together for the less popular files, as peers initially may not have any peers to download from, and the peers may have to wait for the seed to help them out. The differences in download progress for the popular file (see Figures 4(a) and 3(a)) are not so obvious, due to the abundance of resources in the swarm.

E. Upload and Download Rates

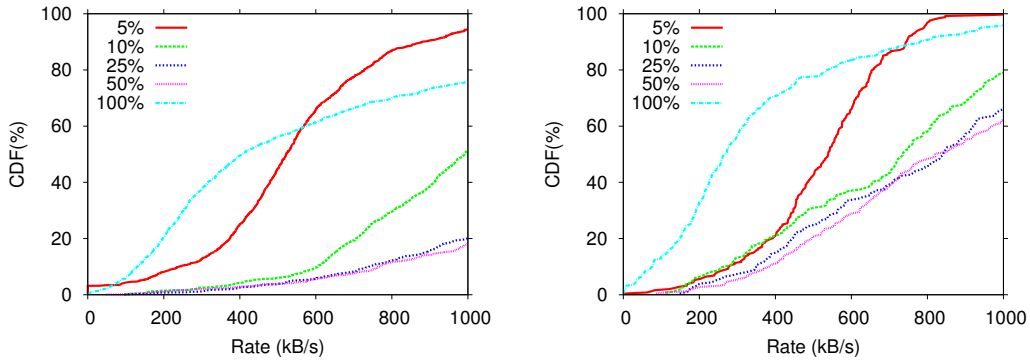
For a closer look at what happens at the peers, Figure 5 shows the download rate as observed when using the random bundling policy. While the download progress looks fairly smooth, it is interesting to note that we still observe the *first piece problem* and the *last piece problem*, as exemplified by the lower download rate for peers in the beginning (5%) and the end (100%) of their download, respectively [14], [22].

The first piece problem is due to peers at the beginning of their download not yet having obtained pieces to exchange with others, and that therefore must rely on optimistic unchoking. For example, when a new peer participates in a swarm, a purely rate-based unchoke algorithm will never choose the new peer (without pieces to share). Thus, the new peers largely rely on optimistic unchoking. With only a small fraction of



(a) Popular (1) (b) Non-popular (2-10)

Fig. 4: Download progress using the random bundling policy. Download progress measured by the time it takes to download a certain percentage of the file content.



(a) Popular (1) (b) Non-popular (2-10)

Fig. 5: Download rate after a peer has made a certain download progress. (Random bundling policy)

the upload connections allocated for optimistic unchoking this naturally leads to the low download rate at the beginning of the download. As we can see, the 5% line in Figure 5 has much lower download rate than others.

The last piece problem is due to peers at the end of their downloads less likely to be unchoked, as few peers have pieces they want. The closer to the end, the more evident the problem becomes. We note that these differences are observed for both types of peers. In contrast, if we take a closer look at the upload rates of these peers, the upload rate with which the peers can contribute, as expected, is monotonically non-decreasing. This is illustrated in Figure 6. As the download percentage increases, the peer upload rate gets higher accordingly.

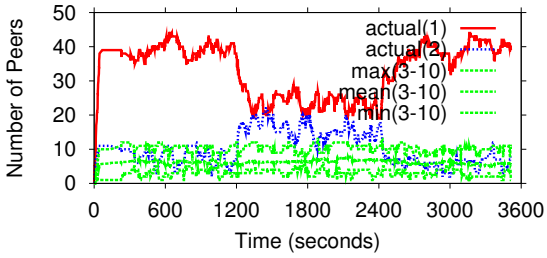
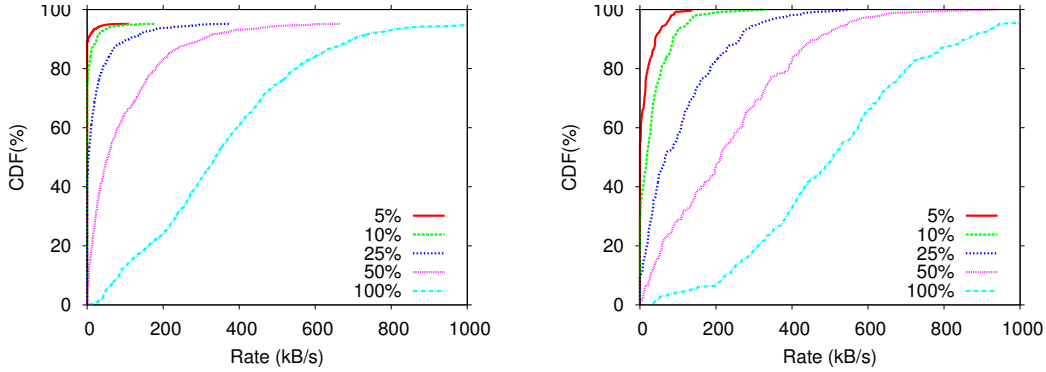


Fig. 7: Number of active downloaders of each file type, using the random bundling policy for the dynamic experiments.

F. Dynamic Transient Experiments

To capture how the policies adapt to changing conditions, we use a set of dynamic experiments, in which the file popularity changes with time. As in our steady-state experiments, we use two scenarios: one with purely random file requests, in which the popular file is requested with higher probability than the less popular ones, and one in which the files requested by each node are statically assigned. The results for both cases are similar, and for the purpose of evaluation, only results for the random scenario are presented here.

For this scenario, we use a total of 69 PlanetLab nodes. We modify the script such that the request rate is time varying, with a time granularity of the same order of magnitude as the download time. In particular, we let the probability of selecting the most popular file (file 1) be 60/69 with the exception of the time interval between 1,200 to 2,400 seconds, during which it is 40/69. During the corresponding time interval, the probability of selecting file 2 increases from 1/69 to 21/69. For



(a) Popular (1) (b) Non-popular (2-10)

Fig. 6: Upload rate after a peer has made a certain download progress. (Random bundling policy)

all other files, the probability remains fixed at 1/69. As a result, there is a higher request rate during the time interval between 1,200 to 2,400 seconds for file 2. This allows us to see how the download performance for file 2 changes over time, as the popularity as well as the assignment of helper peers changes. We keep the experiment running for at least 3,600 seconds before killing the scripts. As before, the bundling policies select files to help from the set of files that have between 1 and 10 peers currently downloading the file.

Figure 7 shows how the number of peers changes over time for the random bundling policy. We note that our tracker-based bundling policies are adaptive, and ensure that file 2 has a reasonable number of downloaders both when it is popular (between 1,200 - 2,400 seconds) and unpopular. While the request rate changes by a factor of 20, the number of peers changes by only a factor of 2-4.

The effect of the changes in popularity on the download times can be seen in Figure 8, where we compare the download time for different files. Results are shown for the two bundling policies, as well as without bundling. The reported download times are the average of the download times of the peers arriving within a 250 seconds wide time window. For the most popular file (file 1; popular/red line), the average download time does not change much with the fluctuation of the number of peers. The high selection probability of the most popular file (40/69 for file 1 during the 1,200 to 2,400 second period and 60/69 otherwise) guarantees its relatively stable download rate. However, the increased popularity of the dynamic file (file 2; dynamic/blue line) can lead to significant changes in its download time. This can be seen without bundling (Figure 8(a)). The file 2 download time during the 1,200 to 2,400 second period (dynamic/blue line) is much lower than that before 1,200 and after 2,400 seconds.

In contrast, with the random bundling policy (Figure 8(b)) and balanced bundling policy (Figure 8(c)) the download time is much more stable. For example, the bundling policy adaptively adjusts the allocation of helper peers for file 2 download during the time interval when the file is more popular, and it does not require such peers. For the cold files 3-10, the download time should not change too much because they have the same selection probability throughout and this can be seen for both bundling policies (Figures 8(b) and 8(c)).

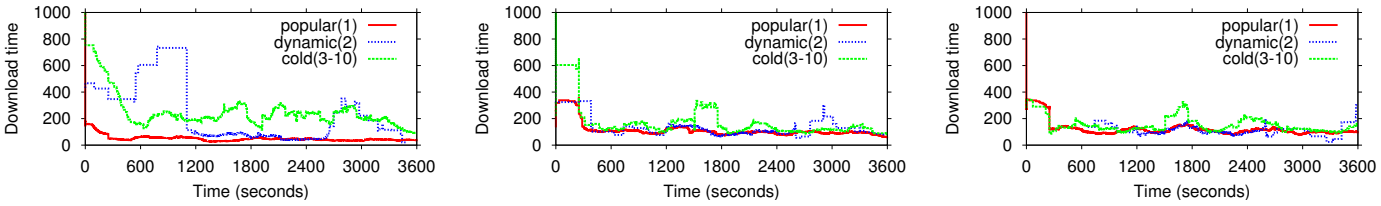
Finally, we note that these dynamic results provide additional examples of how the download times for the less popular files can be improved with the help of bundling. This is particularly evident by comparing the download times of file 2 during the times it is less popular (as well as the download times of the other less popular files) for the cases with and without bundling.

V. Discussion and Conclusions

In this work, we presented the design of a dynamic bundling system along with a proof-of-concept implementation of the design. We create super bundles of many files and let users pick which files they want and request that they also help out in the sharing of one or more additional files within this bundle. The file(s) a user downloads in addition to the user's file of interest are determined dynamically. Dynamic allocation of individual bundles allows the system to inflate swarms based on measured demand for files and the desired tradeoff (for example between server bandwidth and download latency). While our super bundle approach allow for dynamic bundling and adjusting the participation in the sharing of files, we acknowledge that it has some limitations.

One shortcoming of the current implementation is that super bundles must be statically created, and once a super bundle is created additional files cannot be added to it. An ideal implementation would have a single giant super bundle that contains all existing files, and to which files may be dynamically added. For such flexibility to be achieved, a number of issues must be addressed. Most importantly, peers must be able to easily communicate interest and make piece requests among each other regardless of which files they are downloading. Unfortunately, the current mainline software makes this difficult to implement when peers use different .torrent files in which the relative piece indices for the same content may be different.[4] In

[4]As an example, we have found that there are at least nine types of mappings of piece indices in the system. Based on these mappings, we have made initial experiments when peers are only allowed to have at most two files per individual bundle. However, this alternative approach has been difficult to generalize. The super bundle system allows larger individual bundles [21].



(a) No bundling (b) Random policy (c) Balanced policy

Fig. 8: Download time for the random dynamic scenario. (Reported download times are the average download times of the peers arriving within a 250s time window.)

addition to communication between peers, another challenge is the piece disk writing algorithm which constrains the piece writing sequence. Similar challenges are associated with disk reads. In general, the current BitTorrent standard and software are based on the assumption that all the peers in the swarm have the same .torrent file, referencing the same pieces with the same piece index sequence. These assumptions do not fit the design of a fully dynamic bundling system, and hence introduce more difficulties for modifying the source code.

Given the complexity of a fully generalized dynamic bundling system, the super bundle system is designed such that peers can be assigned individual bundles from a large catalogue of files. The approach provides high flexibility and our experimental performance evaluation is encouraging. To the best of our knowledge this is the first implementation of a dynamic bundling system. It provides concrete validation of dynamic bundling as a solution to the content availability problem in BitTorrent. The design only makes small changes to current BitTorrent systems, makes use of the current system structure and is compatible with the current BitTorrent specification. We leave the fully dynamic bundling system implementation and experiments for future work. Ongoing work includes the mathematical modeling of which contents in a large catalogue to bundle, and which not to bundle [23].

VI. Acknowledgements

We thank the anonymous reviewers. This work was supported by National ICT Australia (NICTA), the Natural Sciences and Engineering Research Council (NSERC) of Canada, and CENIIT at Linköping University.

References

[1] G. Dan and N. Carlsson, "Power-law revisited: A large scale measurement study of p2p content popularity," in *Proc. IPTPS*, Apr. 2010.

[2] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan, "Measurement, modeling and analysis of a peer-to-peer file-sharing workload," in *Proc. ACM SOSP*, Oct. 2003.

[3] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. IEEE INFOCOM*, Mar 1999.

[4] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: A view from the edge," in *Proc. IMC*, Oct. 2007.

[5] S. Mitra, M. Agrawal, A. Yadav, N. Carlsson, D. Eager, and A. Mahanti, "Characterizing Web-based video sharing workloads," *ACM Transactions on the Web*, vol. 5, no. 2, May 2011.

[6] D. S. Menasché, A. A. A. Rocha, E. A. de Souza e Silva, R. M. Leão, D. Towsley, and A. Venkataramani, "Estimating self-sustainability in peer-to-peer swarming systems," in *Proc. IFIP Performance*, Nov. 2010.

[7] N. Carlsson, D. L. Eager, and A. Mahanti, "Using torrent inflation to efficiently serve the long tail in peer-assisted content delivery systems," in *Proc. IFIP/TC6 Networking*, May 2010.

[8] D. S. Menasche, A. A. A. Rocha, B. Li, D. Towsley, and A. V. Taramani, "Content availability and bundling in swarming systems," in *Proc. ACM CoNEXT*, Dec. 2009.

[9] J. Han, S. Kim, T. Chung, T. T. Kwon, H. chul Kim, and Y. Choi, "Bundling practice in bittorrent: What, how, and why," in *Proc. ACM SIGMETRICS/Performance*, June 2012.

[10] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *IEEE Internet Computing*, vol. 6, no. 5, Sept/Oct. 2002.

[11] B. Cohen, "Incentives build robustness in bittorrent," in *Proc. Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[12] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *ACM SIGCOMM*, Aug/Sept. 2004.

[13] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming," in *Proc. IEEE INFOCOM*, Mar. 2005.

[14] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest first and choke algorithms are enough," in *Proc. IMC*, Oct. 2006.

[15] R. Bhagwan, S. Savage, and G. Voelker, "Understanding availability," in *Proc. IPTPS*, Feb. 2003.

[16] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, analysis, and modeling of bittorrent-like systems," in *Proc. IMC*, Oct. 2005.

[17] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The bittorrent p2p file-sharing system: Measurements and analysis," in *Proc. IPTPS*, Feb. 2005.

[18] G. Dán and N. Carlsson, "Dynamic swarm management for improved bittorrent performance," in *Proc. IPTPS*, Apr. 2009.

[19] Y. Yang, A. L. H. Chow, and L. Golubchik, "Multi-torrent: A performance study," in *Proc. MASCOTS*, 2008.

[20] S. Zhang, N. Carlsson, D. Eager, Z. Li, and A. Mahanti, "Towards a dynamic file bundling system for large-scale content distribution," in *Proc. MASCOTS*, July 2011.

[21] S. Zhang, "A dynamic file bundling approach for large-scale content distribution," Master's thesis, University of Calgary, Dec. 2010.

[22] M. Lin, B. Fan, J. C. S. Lui, and D.-M. Chiu, "Stochastic analysis of file-swarming systems," in *Proc. IFIP Performance*, Oct. 2007.

[23] N. Carlsson, G. Dan, D. Eager, and A. Mahanti, "Tradeoffs in cloud and peer-assisted content delivery systems," in *Proc. IEEE International Conference on Peer-to-Peer Computing (P2P)*, Sept. 2012.