Automatic Transition From Client-Server To Peer-to-Peer Download

A Thesis Proposal

Presented to the

Department of Computer Science

Brigham Young University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Roger Pack

November 2006

I. Abstract

In traditional web client-server downloads, when the number of clients increases, the available server bandwidth decreases. This causes servers to become overloaded or to serve files slowly. This proposal addresses this problem by integrating peer-to-peer swarming with traditional client-server downloads. Clients initially attempt to download the original file. If this is too slow, they ask their peers for pieces of the file, and then begin to offer those pieces to other peers.

II. Problem Introduction

In the traditional Internet architecture, a server handles incoming client requests. This is referred to as the client-server paradigm. The bottleneck in such systems tends to be the bandwidth at the server [16], as this bandwidth is split N ways among the accessing clients (see Fig. 1). When an increasing number of users access a web page, that page therefore loads more and more slowly. This predicament is common, and occurs whenever server bandwidth is low, a server experiences a spike in load, or when clients download very large files. A dramatic example of this problem is the "slashdot" effect, in which a sudden "flash crowd" of unanticipated clients may suddenly request certain pages, overwhelming the unexpectant server. This can happen when sites are suddenly linked to by popular sites, such as slashdot.com. These situations cause web servers to become overwhelmed and web users to have a less than favorable experience.

This problem is most commonly solved by distributing the load among a set of servers that mirror the content of the original server (see Fig. 2). A company establishes mirrors by locating a set of servers in many different places on the Internet, then automatically redirecting clients to these servers thus distributing the load to the various edge servers to provide better performance for users. An example of this is the commercial CDN run by Akamai [10], which is employed by Apple to distribute their popular program iTunes™. This solution requires a dedicated pool of servers to provide the extra bandwidth, which can be expensive. Not all sites can afford a CDN, and mirrors require cooperation, configuration, maintenance, and bandwidth.

As an alternative, many web sites are relying on *peer-to-peer content distribution*, or *swarming*, with BitTorrent. In BitTorrent, clients download only some blocks of a file from a central server, and get the other blocks from their peers in the system. While a client participates in BitTorrent, it is both downloading the blocks it needs and uploading the blocks it has to other peers (see Fig. 3). Peer-to-peer content distribution enables a web server to serve a large file to large numbers of users without overloading

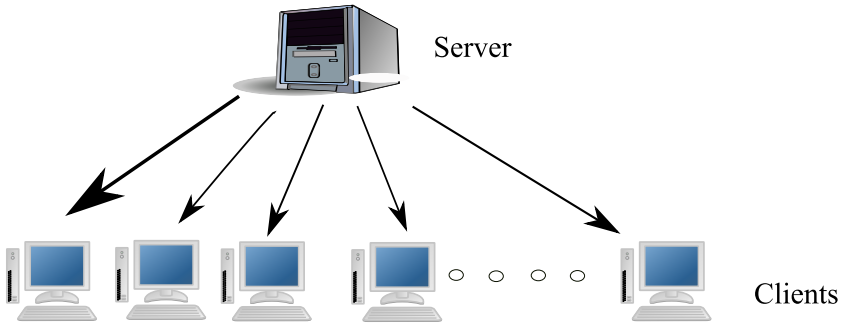


Fig. 1. Traditional HTTP download

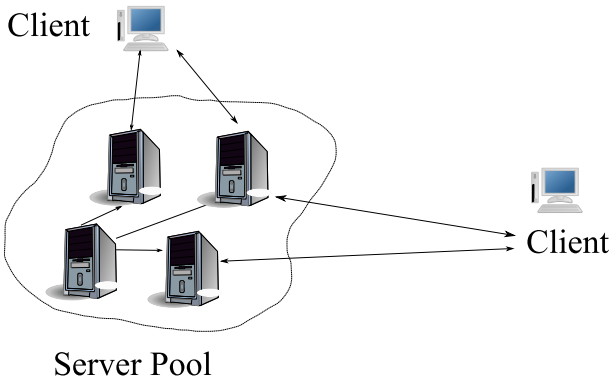


Fig. 2. Server CDN example

its server bandwidth [25]. Swarming protocols like BitTorrent [7] or Slurpie [22] have been shown to even decrease download time per peer as the number of peers increases.

BitTorrent splits files into blocks and connects peers sharing them in a mostly distributed fashion. To accomplish a BitTorrent download, peers must first locate a file with a ".torrent" extension from an out of band source such as a web server. This file lists a target file's size, MD5 hashes of the different blocks of the file (for integrity verification), and the IP address of a tracker for that file. This tracker is a machine which helps connect the peers to each another. Peers contact the tracker and receive a random list of other peers who are currently downloading the file. They then establish connections and begin sharing

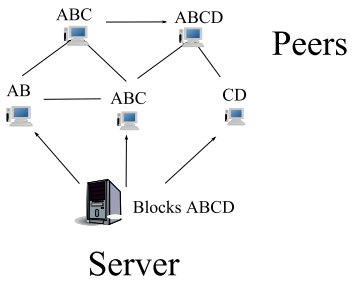


Fig. 3. Swarming example. Arrows represent block exchanges.

and receiving blocks with their new neighbors. Peers in BitTorrent tend to share with peers that also share with them, which creates an incentive for cooperation in the system, referred to as Tit-For-Tat. Finally, when a peer has downloaded the entire file, it may linger in the system, acting as a "seed" and uploading file blocks altruistically.

BitTorrent is inappropriate for serving all the files in a web site. It is designed for large files and is not easily used to serve all files from a web server. For every file that is to be shared a .torrent file must be created, and a tracker established for that file. This type of manual configuration is difficult to repeat for an entire web site. Also, swarms are formed on a per file basis, so it is not possible for a peer in one swarm to find a different file among its peers in the swarm. BitTorrent also uses Tit-For-Tat incentives to share blocks of the same file, which is not appropriate for multiple small file downloads, and causes a longer boot-strapping time.

Ideally, a web server could automatically serve all its files using peer-to-peer file transfer and would automatically transition to peer-to-peer downloads based on the load it experiences. This has been proposed before, but would require that all web servers and clients would use a defined peer-to-peer protocol. Instead, this thesis proposes modifying only web clients so that they can automatically transition to peer-to-peer download for all files on a web server. They will be able to communicate with other modified web clients to get faster downloads, which in turn will provide an incentive to adopt this protocol.

III. Thesis Statement

A system of cooperating web clients can reduce load on a origin web server by automatically switching from client-server file transfer to peer-to-peer content delivery. This system can provide fast download times for objects of many sizes, especially small files. It should be equivalent in speed to BitTorrent for larger files, without requiring any manual configuration or participation from central web servers.

IV. Related Work

Much work has been done to distribute file downloading. Solutions fall into three basic categories: client-side protocols, client and server-side cooperative protocols, and server-side protocols. Each category has certain trade-offs in terms of how ideal it is, in our context.

A. Client-Side Only Protocols

Protocols that use only client-side protocols can help alleviate flash crowds. Client-side interaction typically involves peers self-organizing to help speed downloads. These solutions require no change in

the server software, which allows them to be implemented in participating clients without requiring web servers to be aware of the changes. However, this requires that clients must self-organize without the help of a centralized server.

Shared web caches are an example of this type of system. They provide stable solutions for peers by allowing them to download files from a shared cache [8] [27], though they may require some type of dedicated infrastructure of cache members. Squirrel [11], for instance, is a shared web cache designed for use on a local LAN. Squirrel clients join a DHT and cache any files that map to their location in the DHT. When a user requests a file, that user contacts the member of the local DHT who should be caching that file, and requests it. If the member does not have a copy of the file, it first downloads it from the origin web server, then caches a copy and returns the result to the requesting user. Squirrel does a good job of sharing the caches of participating members on a local LAN. It does not offer an algorithm for transparent transition to p2p download, however, always relying on the proxy for static content.

CoDeen [27] [17] is a distributed proxy, where participating members of the system act as a large web cache which is accessible to outside users. It is the equivalent of a globally accessible, wide-scale version of Squirrel. The load of caching objects is distributed among the different nodes so if one node goes down the entire system is not dramatically affected. It works as follows:

1) Users set their Internet browser proxy to a nearby high bandwidth participant of CoDeen and request all files from it.

2) The proxy, upon receiving requests for a file, either returns the file from its cache, or retrieves it and returns it. To retrieve a file, the proxy requests it from a live member of CoDeen that is the ’owner’ of that file. If the owner has it, it returns the file, otherwise it contacts the origin server and downloads, it then returns it. Codeen also provides load balancing by performing heartbeat tests of its neighbors before requesting files from them.

Similarly, in Coral [8], participating members of a DHT act as a web cache. Users who access this system are redirected (via HTTP redirect) to a close Coral member, which then searches for the file on the DHT, retrieves it, and forwards it on to the user. Coral has a central point of failure in its DNS redirection system, in that it requires a central server to administer all requests for files.

These solutions have some advantages and disadvantages. The advantages are that they reduce the load on the origin server, they load balance among cache members, and they are distributed over a wide area. The disadvantages are that they are constrained to the number of participating proxies. They require an

infrastructure of hosts who are willing to altruistically provide bandwidth, which is a rarity. This means that for these systems, in extreme situations (load surpassing the bandwidth of the combined sum of computers), they would still become overburdened. With our system this should not be a problem, as more peers in the system means that more peers will be available for uploading. This also holds as in traditional web browsing much time is spent only reading pages, which time can be dedicated to uploading content to others at no cost. Having an infrastructure also requires members of the DHT cache files for which they are not directly interested, and clients must choose a static proxy that can become overloaded or go off-line. These systems also lack an algorithm for transition from traditional to peer-to-peer download. They always return the cached copy, even if a request from the origin server would have been the fastest way to download the file.

The effect of flash crowds may also be alleviated by searching among a random subset of Internet peers for desired files. PROOFS [21] (P2P Randomized Overlays to Obviate Flash-crowd Symptoms) uses flooded search to allow peers to locate ’popular’ or ’flash crowded’ files from other peers who have previously downloaded the same. The creators conjecture that files that are the subjects of a flash crowd are likely to be locatable among random sets of peers, since they are popular, hence their use of a random flood search. The benefit of this system is that nodes need not maintain a structured search system (such as a DHT). However, not having a DHT for lookup makes searches more random and slightly less accurate, with potentially higher overhead. PROOFS also does not include parallel downloads nor offer a protocol for automatic transition to a peer-to-peer solution.

Overall, client-side solutions alleviate flash-crowds well, though they are limited by the bandwidth of contributing members in certain cases. There also seems to be no automatic transition to peer-to-peer download.

B. Server-side and Client-side Cooperative Protocols

Many file distribution protocols include some cooperation and previous knowledge between the server and client. In these protocols clients typically access blocks of a file from the server and also from peers. Server-side and client-side cooperative protocols basically fall into two categories: those used for web redirection, and those used to download single objects. For web redirection, servers typically redirect clients to former clients that have downloaded files previously [12], [16]. This allows a server to ’meter’ its upload speeds and redirect peers, when appropriate, thus providing a backup strategy for over-loaded servers. An example of this is the pseudo-server system [12]. This style of protocol typically requires

changes to both the server and client software, however, and the server could still become overloaded in extreme cases.

Some proposals incorporate the above web redirection, but also include swarming [18], [23], [3]. Swarming provides the benefit of scalable downloads of large files. These protocols basically redirect peers to each other and also hand each a block of the file, then let the peers trade amongst themselves to create the entire file. OnionNetworks, for instance, proposes an extension to HTTP to allow HTTP response messages to include hashes of files and a list of peers from to which other peers may connect and download in a swarming fashion [3]. Several protocols simply initiate block-wise file distribution for arbitrary objects [25], [7], [22], [14], [4], [9], [13]. BitTorrent, the most commonly used swarming protocol, was developed in 2001 by Bram Cohen [7]. Its major contribution (besides popularizing swarming) is the use of Tit-For-Tat to encourage sharing and discourage free-loading. Swarming protocols are highly effective in response to flash crowds, effectively serving loads orders of magnitude higher than an ordinary web server [25].

Shark [1] is a file system prototype that allows clients to download blocks of files from nearby neighbors who have blocks already on their local machines. Shark is similar to our proposed system, except applied to files in a file system, not web objects. It requires a custom DHT for peer localization and proximity estimation, and a central server for hash values of files. It also lacks HTTP integration and a switching mechanism.

These systems typically are used for downloading single, predetermined, static files. They all therefore do a straight p2p transfer, without a transition mechanism from normal web download.

Some solutions seek to integrate HTTP clients with BitTorrent itself. In these solutions a user is presented the option of downloading via HTTP or via BitTorrent (swarming), depending on which they think will give them the quicker download. This provides a backup for overloaded servers. To the users this requires a manual transfer from normal to swarming download. The Osprey system [19], for example, is an ftp server that automatically generates .torrent files for all files in an ftp sub-directory, and integrates a BitTorrent tracker into the server software to handle the two types of requests. Several other sites also offer .torrent files alongside normal (typically large) files, to allow users to manually use swarming in cases of high load. These systems allow peers to manually switch to swarming if the server load becomes too high. Dijjer [26] is a tool that automatically performs swarming downloads of any file. If passed a url like http://dijjer.org/get/http://mysite.com/video.mov the request is intercepted by the Dijjer software,

which performs a distributed download of the file. One can also right-click on any arbitrary file and select 'download via Dijjer' for the same effect. Dijjer contacts a quasi-DHT (similar to Freenet [6]) for block hashes and downloads the blocks from random peers who cache these blocks. Unfortunately, Dijjer lacks automation in the transition to download, and, as the Dijjer software is currently implemented, requires peers to cache material in which they were never interested.

Overall, client and server side cooperation protocols work well at alleviating flash crowds, and some provide automation for transition to p2p delivery. However, they require both the server and client to understand the protocol, which hinders their adoptability. The biggest drawback of these protocols is that they are not transparent to the server.

C. Server-side only Protocols

Server-side only solutions typically create a pool of servers that mirror each others' content, helping to alleviate the impact of flash crowds [24], [28]. In these systems, servers will cover for each other, so that if one becomes very crowded then the others will share the load with it until load decreases. Backslash [24] is one example. In Backslash, when one server becomes overloaded it redirects requests and uploads necessary files to other servers for them to act as temporary mirrors, therefore relying on the volunteer hosts in the system.

Overall, server-side systems still tend to be susceptible to flash crowds (albeit to a lesser extent), because of their collaboration, and are built to serve clients that are not p2p enabled, so offer no transition.

V. Proposed Solution

Our proposed solution emphasizes a client-only system that automatically switches from client-server to a peer-to-peer content delivery without any manual configuration for either clients or servers. The goals for our system include:

1) It should be transparent to servers. This allows the origin servers to remain unchanged, so end users can benefit immediately from the system.
2) It should appear transparent to users by automatically transitioning to peer-to-peer content delivery when the server is slow.
3) It should not require a dedicated special-purpose infrastructure. Using a general-purpose infrastructure, coupled with a dependence on the clients for transferring blocks, makes this system easier and cheaper to deploy.

4) It should be non-intrusive, in that peers should not be required to cache blocks of files in which they were never interested. Peers will avoid caching files they never downloaded, and will not be responsible for content they don't anticipate. Users would also only be using heir upload bandwidth for files in which they are interested, encouraging participation and adoption.
5) It should be fast for small files.

Future goals might include ensuring the validity of files and providing explicit incentives.

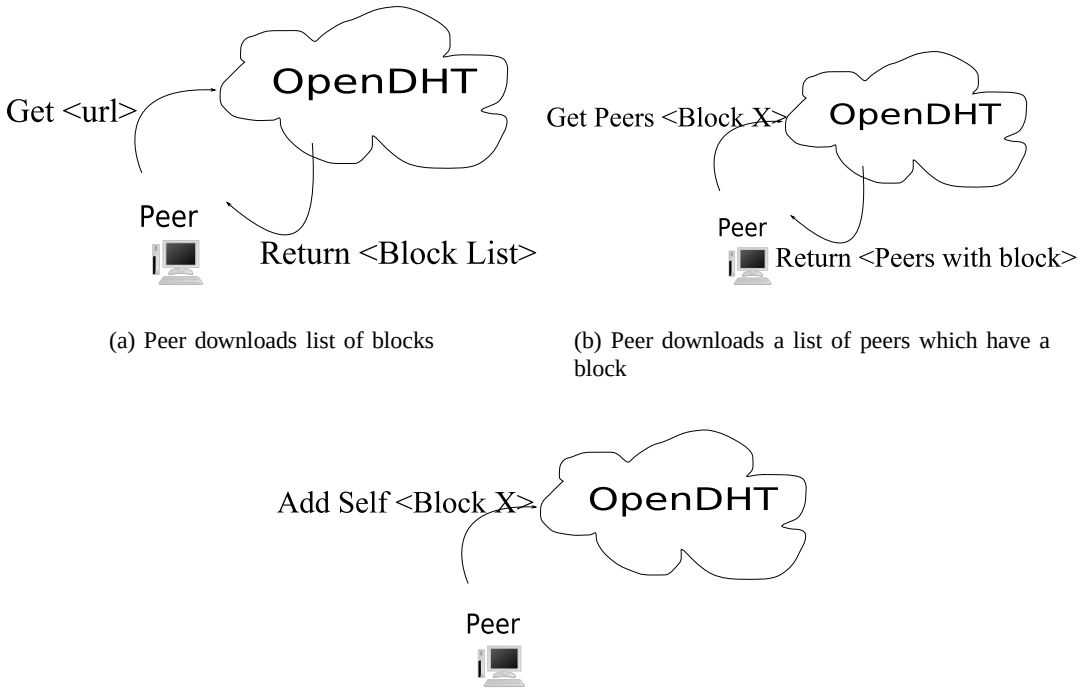
The basic system will be to connect interested peers with one another via a DHT that stores peers lists. A fundamental design decision is also whether to download the contents of blocks from peers or from the DHT itself; peers could store the block contents, and have the DHT serve as a lookup of peers, or could store the block contents on the DHT itself. The trade-off is that having the blocks on the DHT puts more stress on the DHT and relies on members of the DHT for contributing bandwidth, which is intrusive. Imagine for instance caching portions of a file on your personal computer which you never downloaded for yourself, and which you are then sharing with others. Members of the DHT would thus intrusively cache information for which they are not interested. We therefore choose the case of having peers register themselves on a DHT as being willing to serve blocks they own, and have clients download directly from peers. This seems more indicative of a realistic web experience.

A. Basic Algorithm

In our protocol, a peer first tries to download the file from an origin web server. If at some point one of the following conditions occurs, the download will switch to a peer-to-peer swarming download:

1) First the client waits a maximum amount of time T after the start of a normal HTTP download for the first byte of data to arrive. This allows the system to decide quickly whether the origin server is over-burdened and switch to peer-to-peer if needed.
2) Once the client gets some data from the server, then it monitors whether the download rate falls below a certain fixed threshold R bits per second over a window of time w. If the origin server ever becomes slow, the client switches to peer-to-peer delivery.

Once a client decides to switch to peer-to-peer downloading it will perform two steps. First the client will calculate a hash value for the filename. It will use this as a key value in the DHT to retrieve a list of the blocks of the file. The peer will then take each of the blocks' respective hash value as a key to retrieve a list of peers who have the block and are willing to upload it. The peer will choose one peer at



(a) Peer downloads list of blocks

(b) Peer downloads a list of peers which have a block

(c) Peer adds itself to list of peers who have that block

Fig. 4. Steps to accomplish a peer-to-peer-web download.

random from the list, download the block from that peer, and then add itself to the list of those containing the block (see Fig. 4). If there are no peers listed for a block, it will make a note in the DHT that it is accessing the origin server for that block, and download it using an HTTP Content-Range request from the origin server. Peers will initiate this block-wise transfer for some b blocks at a time. If peers download a file previously unknown to the system, then the hash values for each block will be unknown. In this case a peer will first make a note that they are accessing the origin server, for that block, then they will download it, calculate its hash value, and add themselves as the first peer in the list of those having that block. This should accomplish a distributed download without any server knowledge or manual configuration.

B. Further Optimizations

The algorithm described above may not be the most efficient. To improve performance several optimization may be necessary.

First, this protocol may not be resilient against accidentally selecting and downloading from 'slow' peers (which could cause a slow last block problem, as well as being generally detrimental to speedy download). This will be overcome by using an algorithm similar to that discussed for the origin server.

Peers will be given a timeout to connect to them of a few seconds, and will have a ”lowest allowable rate” hard-coded. This will also happen to accomplish a form of load balancing, in that slower peers are dropped more frequently, so fast peers tend to upload more.

It is also possible to establish a load control to allow only a certain number of peers to access the origin server simultaneously. This will be accomplished by limiting the number of peers accessing the origin server, per block, to 3 (a good number, according to the Slurpie paper [22]). Having a small number of peers contact the origin server allows the server to more quickly upload blocks to those peers. These peers can then begin to distribute the blocks.

VI. Methodology

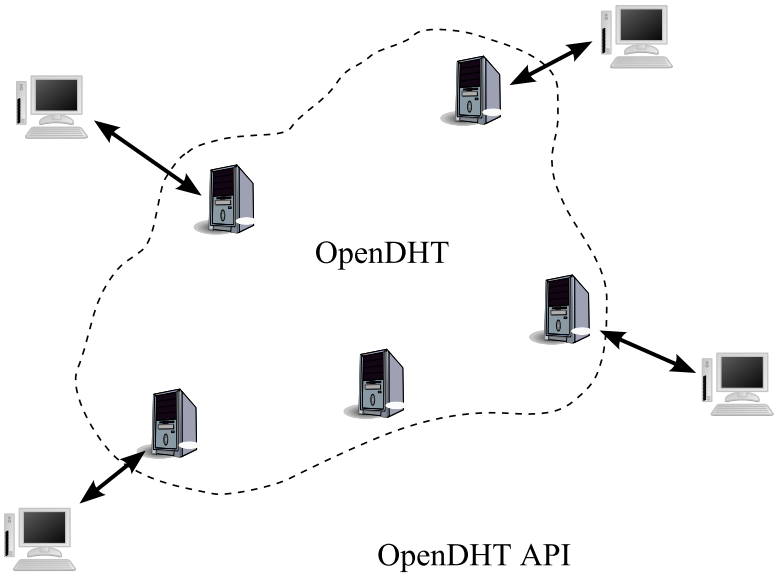
We will build our client and test perform experiments to assess its ability to meet the design goals and satisfy the premise of our thesis. The experiments will be run on the PlanetLab testbed [5]. PlanetLab comprises 600+ computers in 400+ differing global locations with a single sign-on for each researcher. Each PlanetLab computer has the ability to limit its outgoing bandwidth, which will allow us to experiment with different scenarios. Using PlanetLab will give us a good outlook on how this protocol would perform in the Internet.

For a general purpose DHT, we will use the pre-built OpenDHT [20]. This is a DHT already deployed on PlanetLab for general use. To access it, users contact any member and have that member perform queries and return the results (see Fig. 5). Using OpenDHT will allow us to divorce study of the DHT from the study of this protocol, which protocol uses a DHT, as OpenDHT has already been tested and optimized. In a production-style environment the DHT would itself be hosted on each client running the protocol, but for now using the pre-existing framework will speed development and testing. OpenDHT will likely be running on many of the same computers we use as clients; we will attempt to quantify the effect this has on our measurements.

This differs from other systems in that the lookups will be generated for slow content, the queries are using a generic DHT (easier to configure), and the queries are block-based (which is rare for a DHT built sharing system).

A. Workload

In most experiments we will choose a single file or a set of small files to download, then will vary the rate of clients requesting the file.



OpenDHT API

put(k,v,H(s),t)	write *(k,v)* for TTL *t*, can be removed with secret *s*
get(k) returns *{(v,H(s),t)}*	read all *v* stored under *k*
remove(k,H(v),s,t)	remove *(k,v)* put with optional secret *s*, *t* > TTL remaining for put

Fig. 5. OpenDHT Description.

B. Metrics

We will run experiments and log all events and messages, then process logs to calculate the appropriate metrics. Specific measurements will be client download and upload rates and times, server upload rates, total throughput of the system, total bytes uploaded by clients, and total bytes uploaded by the server. For all experiments we will examine the distribution of these metrics across the peers, using averages and percentiles to analyze overall behavior of the system.

C. Experiments

First we will do a proof-of-concept experiment to see if this system effectively meets the premise of the thesis: automatically switching to peer-to-peer content delivery results in improved download time. We will use a 100KB file and hold other variables constant (with hard-coded, reasonable values), then exert an increasing client load on a server, up to the rate of saturation for the system. We will compare this with the same load on a traditional client-server system. We will use Ruby for the implementation to speed development. We will consider ourselves successful if we are able to gain two times the speed of the traditional server.

1) Automatic Transition: After proving that the system is viable, we will then run tests to vary parameters and see the effect this has on the protocol. We will thereby determine the 'best' values for T (the time before switching to peer to peer content delivery), R (the rate at which we will spontaneously decide to give up on the origin server because it is too slow), and W (the time slot of recency in which to calculate R). These basic experiments will determine settings for parameters that will be used for the rest of the experiments. We will use a fixed file size of 100KB, and a fixed client arrival rate. We will fix b, the number of neighboring peers from whom to download, at a max of 20, which is shown to be reasonably good [2]). Block size will always be fixed at 32KB, which is shown to be reasonably effective in [25]. Lingering time will be set at 0s (no lingering). These experiments will be run for 30 minutes, or until 1000 downloads complete, whichever comes first. We will use an Apache server to distribute the original files. We will begin by holding all other variables constant and varying T from .5s to 5s. Similarly, we will hold all variables constant and vary R from 50Kbps to 1 MBps, and vary W from 1s to 10s (possibly using an estimated mean weighted average EMWA for download speeds).

We will next test the system with varying server bandwidths to ensure these values are appropriate for a variety of loads. We will use the original experiments and vary server bandwidths of 32Kbps, 256KBps, 1Mbps and 2Mbps.

2) Entire Web Site: We will next examine the ability of our system to serve an entire web site. We will examine effectiveness with a typical web sized files by using a copy of the BYU home page and its associated objects to examine performance downloading an entire web site. Block size might make a big difference for small files. We will experiment with different block sizes to determine the impact of this. The supposition is that too small of a block size will be detrimental, as will too large.

3) Optimizations: Next we will test the effect of lingering times on system performance. These tests will be with a fixed file size of 100KB, server bandwidth of 256Kbps, and a request rate of 20/min (10x bandwidth). We will perform the original experiments and vary lingering time from 0s to unlimited. We expect lingering time on the order of a minute or two will give most of the useful benefit to the system.

After these we will examine whether optimizations seem necessary, from the above experiments. If needed, we will repeat these experiments with these features turned on and examine whether they provide a performance improvements. If necessary, we will experiment with imposing a load control of 3 peers per block accessing the origin server (shown in [22] to be efficient), and avoid the slow last block problem.

Finally, we will compare the performance of our protocol with BitTorrent [8]. We will repeat the basic experiment with BitTorrent and compare it with our own.

VII. Proposed Thesis Schedule

Proposal: Nov 12 or so

Basic implementation: Dec 31

First experiments completed Feb 1

Optimizations completed March 1

First draft March 15

Final draft April 1

VIII. Contribution to Computer Science

The major contributions of this thesis will be that it creates what we believe to be a unique client-side system of cooperative web clients that automatically transitions from client server to peer-to-peer delivery as needed. It is transparent to both the server and the user and is non-intrusive in that users do not download files they do not want. It will be appropriate for smaller files and will not require a special purpose DHT. This contribution could dramatically increase the utility of swarming for everyday web browsing. This thesis could serve as a useful landmark for examining the scenarios when this tool is helpful, and provide hints for best-practices should it be developed by industry.

Another interesting feature of the system is that it has the potential to be a kind of ‘backup’ for servers, if peers in the system have downloaded files. Since we only lookup files by their URL, it is possible to download files for servers that have crashed or deleted the original files. This therefore provides redundancy and reliability for servers that go off-line (see Resurrect [15]) (OpenDHT keeps entries for up to a week).

There has been some concern about the legality of peer-to-peer protocols in the past. This algorithm, however, represents a generic peer-to-peer protocol, which will tend to be used with normal web downloads, which tend to be legal, so represents a way of using peer-to-peer downloads for a typically legal end. File sharing and downloading of movies have given p2p a bad name, so this represents a break from that trend, though could still be used for illegal content.

It therefore provides an automatic transition, and also will help us understand how to combine a larger system (BitTorrent style swarming) with a system suitable for small files.

IX. Delimitations of Thesis

We do not plan on running tests in a mix of normal peers and ‘swarming-enabled’ peers, which would be indicative of a real world trial. We are also not looking at the possibility of a server redirecting ”non-aware” clients to peers that would be willing to serve it to them (similar to pseudo-servers [12]). Nor do we plan on using traces of real world traffic, as the tests listed should be descriptive enough of the protocol.

We also plan on running experiments mostly on clients that tend to have high upload links–which is not truly indicative of a real-world experience (where most links are assymetric).

A means of verifying that content is not corrupted or maliciously modified is also lacking. A server that is aware of the protocol could potentially sign checksums and place them in the DHT, or, alternatively, clients could form a reputation system, such as ’voting’ on the contents of a block. Analysis of this is not included in this work.

References

[1] S. Annapureddy, M.J. Freedman, and D. Mazieres. Shark: Scaling File Servers via Cooperative Caching. *Proceedings of the 2nd USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), Boston, USA, May*, 2005.

Shark uses block distribution through a DHT named Coral [8]. It basically uses coral to lookup local peers who have a copy (locally) of a file and downloads it from them. It requires a custom DHT for this, and has a central server which is also cognizant of the protocol.

[2] A.R. Bharambe, C. Herley, and V.N. Padmanabhan. Analyzing and Improving BitTorrent Performance. *Microsoft Research, Microsoft Corporation One Microsoft Way Redmond, WA*, 98052:2005–03.

This paper analyzes some aspects of BitTorrent, such as the fact that outward-degree, or number of peers, increases up to 20, then decreases, though this seems like a relatively little studied aspect.

[3] J. Chapweske. HTTP Extensions for a Content-Addressable Web, May 2002. http://www.open-content.net/specs/draft-jchapweske-caw-03.html.

OnionNetworks proposes that HTTP headers be extended to include the lists of file block hashes and peers who have recently downloaded it, allowing a distributed download of files seemlessly.

[4] L. Cherkasova and J. Lee. FastReplica: Efficient Large File Distribution within Content Delivery Networks. *4th USENIX Symposium on Internet Technologies and Systems*, 2003.

See mutualcast [14]. FastReplica performs a similar efficient block wise transfer as it.

[5] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

PlanetLab is a global distributed test bed available to researchers for internet wide experiments, via using slices of computers which researchers may control.

[6] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Workshop on Design Issues in Anonymity and Unobservability*, 320, 2000.

Freenet is an anonymous lookup for web pages. It uses a DHT style lookup where queries tend toward peers that tend to have the block, and uses intermediate caching.

[7] B. Cohen. Incentives build robustness in bittorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, 2003.

BitTorrent's makers recently came out with the capability to search for files without contacting the central server (via a DHT search), bringing that product one step closer to automatically downloading any file. BitTorrent motivates peers' cooperation with a 'Tit For Tat' incentive policy. Peers which upload most quickly to others have a higher chance of being in turn uploaded to. Peers accomplish this by choosing four neighboring peers which upload more quickly to it as those to which it will then choose to upload to.

[8] M.J. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with Coral. *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI 2004)*, pages 239–252, 2004.

The search avoids hot spots and leverages locality by creating concentric DHT's. Each node is a member of several DHT rings representing nodes who are within certain proximity of it (and to each other). It creates these rings with expanding size–i.e. DHT for nodes within 100ms, and one within 500ms, and one with global scope. Members first query the DHT of nodes close to them, then the next ring up, then the next, until they find a node that has cached the file, or it is not found. This allows them to use DHT queries for 'close' nodes, first (reducing latency), and to contact members who are close. Coral does well in finding local copies of files quickly. Coral relies on a central access point for redirection, and does not have automatic fall over to swarming download, though it does have excellent locality properties. Coral pages are 'cached' by accessing a url like http://outside.page.nyud.net:8090/subdir/pagename.

[9] C. Gkantsidis, P. Rodriguez, et al. Network Coding for Large Scale Content Distribution. *Proceedings of IEEE Infocom*, 2005.

Avalanche is a BitTorrent like protocol that uses Forward Error correcting codes to make it so that clients only need to download a certain percentage of the total blocks to then be able to recreate the entire original file. They show this helps with some rare block problems, especially for faster peers.

[10] A.T. Inc. Akamai. *URL http://www.akamai.com/en/html/services/edgesuite.html*, October 2006.

Akamai provides a large scale Content Distribution Network that consists of 20,000 servers in 71 countries, allowing corporations to effectively avoid flash crowds, who use it.

[11] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized, peer-to-peer web cache. *Proceedings of the 21st Annual PODC*, 2002.

Squirrel networks the caches of computers residing on a LAN to allow them to lookup files in this shared cache and thereby save on bandwidth, if they have been accessed recently by users. It does not provide a distributed download, however.

[12] K. Kong and D. Ghosal. Pseudo-Serving: A User-Responsible Paradigm for Internet Access. *WWW6 / Computer Networks*, 29(8-13):1053–1064, 1997.

In pseudo serving clients may agree to act as serving backup agents for a server, which then pawns off future requests to them, if it is hammered like a hammer shark.

[13] D. Kostic, R. Braud, C. Killian, E. Vandekieft, J.W. Anderson, A.C. Snoeren, and A. Vahdat. Maintaining High Bandwidth under Dynamic Network Conditions.

Bullet Prime, described here, is a protocol for downloading a large file–it does peer location using a random percolation through the peers of their neighbors, then has each peer dynamically choose an appropriate number of neighboring peers from which to download. It involves a little bit of overhead with its use of the RanSub routine to connect peers, however.

[14] J. Li, P.A. Chou, and C. Zhang. Mutualcast: An Efficient Mechanism for Content Distribution in a Peer-to-Peer (P2P) Network. Technical report, MSR-TR-2004-100, Sept. 2004, 2004.

Mutualcast connects peers downloading a file with one another. Each block is assigned to exactly one peer, which redistributes that block to all others. Those peers who are more 'productive' are given more blocks to distribute. Mutualcast uses a simple queue at each peer of 'blocks completed' which, when empty, is filled by the server, to make use of all available bandwidth. The lesson we learn from this protocol is the importance of giving high bandwidth peers more blocks to share.

[15] Anthony Lieuallen, October 2006. https://addons.mozilla.org/firefox/2570/.

Resurrect is a FireFox extension which allows users, upon accessing a currently dead link, to search coral CDN [8], google, and Yahoo (etc.) caches for the same file.

[16] V.N. Padmanabhan and K. Sripanidkulchai. The case for cooperative networking. *Proceedings of IPTPS '02*, 2002.

In CoopNet the server assigns peers to others who are 'close in IP' (same prefix at a certain byte range) in an attempt to assign peers to others which are close to them. CoopNet (the Co-operative Network) redirects incoming peers to peers who have recently downloaded the file and have agreed to act as mirrors. They suggest some form of blocking for cached multimedia files but leave the research up to future work.

[17] K.S. Park and V.S. Pai. Scale and Performance in the CoBlitz Large-File Distribution Service. *NSDI '06*, 2006.

CoBlitz is used within CoDeen to allow members to cache large files in a distributed way on the system. CoBlitz distributes large files block by block among the different members participating in the system. Files are thus not saved in the cache of single members of the proxy-system, but are instead saved block-wise by several members, and 'gathered up,' from those, when the fiel is requested.

[18] J.A. Patel and I. Gupta. Overhaul: Extending HTTP to combat flash crowds. *Lecture notes in computer science, WCW '04*, pages 34–43, 2004.

Overhaul is a server system in which the server starts serving only blocks of a file when it becomes overloaded, while connecting peers to each other to download the various blocks. The authors show that 'chunking' (splitting the file into blocks) immensely benefits the system in its strength against flash crowds.

[19] J. Reuning and P. Jones. Osprey: peer-to-peer enabled content distribution. *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 396–396, 2005.

Osprey creates .torrent files for arbitrary files in an ftp server directory.

[20] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: a public DHT service and its uses. *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 73–84, 2005.

OpenDHT provides a globally accessible DHT in which clients can query and set key/value pairs with simple ease of use. It provides set, get, and set with key (which last one is immutable without knowing the key), so it provides a nicely manageable API. It is also stable and good for testing.

[21] Dan Rubenstein and Sambit Sahu. Can unstructured P2P protocols survive flash crowds? *IEEE/ACM Trans. Netw*, 13(3):501–512, 2005.

PROOFS provides a random, unstructured 'backup' net overlay, in which, if a file is not found on the internet, you can run flooded queries through this overlay (which happens to be robust to conniving peers), to find the file.

[22] Rob Sherwood and Ryan Braud. Slurpie: A cooperative bulk data transfer protocol. In *INFOCOM*, 2004.

Slurpie is an alternative protocol to BitTorrent [22] developed at the University of MD. It allows only a few peers to connect to a central seed at a time, allowing those connected to the seed to download unique, rare blocks quickly and begin to share them, thus decreasing the overall download time. Unfortunately they assume well-connected peers and poorly connected seeds, and therefore fail to discover what makes their download system faster than BitTorrent (which it is).

[23] G. Sivek, S. Sivek, J. Wolfe, and M. Zhivich. WebTorrent: a BitTorrent Extension for High Availability Servers.

In Webtorrent Mozilla meets BitTorrent as web pages are packaged into larger files which are then served using BitTorrent. To locate this paper you must use google scholar's cached copy.

[24] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address flash crowds. *1st International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, 2002.

Backslash is a server based system in which servers cache each others content to prevent overload.

[25] D. Stutzbach, D. Zappala, and R. Rejaie. The Scalability of Swarming Peer-to-Peer Content Delivery. *Proc. of the 4th International IFIP-TC6 Networking Conference*, pages 15–26, 2004.

Examines the effectiveness of a swarming protocol against order of magnitude larger crowds–it is deemed to be effective.

[26] Dijjer Development Team, 2006. http://www.dijjer.org.

Dijjer provides distributed download for any file on the internet via intercepting url's of the form http://www.dijjer.com/linkToThisFile. It uses a DHT similar to Freenet to lookup the different blocks, after first getting the hash values of the blocks saved somewhere in the DHT, as well. It unfortunately is invasive as peers need to cache blocks for which they are not directly concerned, and also its lookup is not totally guaranteed to succeed.

[27] L. Wang, K. Park, R. Pang, V. Pai, and L. Peterson. Reliability and Security in the CoDeeN Content Distribution Network. *Proceedings of the USENIX 2004 Annual Technical Conference*, 2004.

CoDeen has heart-beat monitor of its neighbors, so it knows to direct queries only to live neighbors. It accomplishes this by downloading small HTTP files and using pings to determine liveness, among other things. When large files are requested (i.e. many different proxies request the same large file), it uses a kind of 'multi-cast' from one peer through a tree expansion stream to the others (which assumes that the first peer in the stream is fast). This is bad if the tree is non-optimal, but reasonable if the original server is slow so that that won't make a difference. CoDeen at least once in their experiments, ran out of bandwidth (i.e. saturated a single proxy connection which many members were using.

[28] W. Zhao and H. Schulzrinne. DotSlash: A self-configuring and scalable rescue system for handling web hotspots effectively. *International Workshop on Web Caching and Content Distribution (WCW)*, 2004.

The DotSlash system operates by contacting backup servers one and a time and asking each to become a redirection cache for an overloaded server.

This thesis proposal by Roger Pack is accepted in its present form by the Department of Computer Science of Brigham Young University as satisfying the thesis proposal requirement for the degree of Master of Science.

Daniel Zappala, Committee Chair

Mark Clement, Committee Member

Christophe Giraud-Carrier, Committee Member

Parris Egbert, Graduate Coordinator