The Scalability of Swarming Peer-to-Peer Content Delivery

Daniel Stutzbach[1], Daniel Zappala[2], and Reza Rejaie[3]

[1] University of Oregon, Eugene, Oregon
{agthorr, reza}@cs.uoregon.edu
[2] Brigham Young University, Provo, Utah
zappala@cs.byu.edu
[3] University of Oregon, Eugene, Oregon

Abstract. Most web sites are unable to serve content to a large number of users due to the inherent limitations of client-server file transfer. Recent peer-to-peer content delivery protocols have demonstrated the feasibility of spreading this load among the clients themselves, giving small web sites the possibility of serving large audiences with very low cost. In this paper we use a simulation-based performance evaluation to study the fundamental question of the scalability of swarming peer-to-peer content delivery. Our results demonstrate the superior scalability of swarming with respect to load, file size, block size, and client bandwidth.

1 Introduction

While the Web has transformed the way we use the Internet, it can still be very slow when large numbers of users try to access a single web site at the same time. Access can be particularly slow when users are trying to search a database or download a large file (such as a software update). Today's state-of-the-art in this area is a Content Distribution Network (CDN), which replicates a web server's content at various locations in the network, then provides a mechanism to redirect users to a local replica.

Although a CDN is feasible for large content providers who expect high access rates, the cost is often not justifiable for what we call "ordinary users" – organizations with unpredictable demand or smaller budgets, such as schools, governments, small companies, and individuals with personal web sites. Several alternatives are available to these users, but none are satisfactory. Buying more bandwidth is certainly one possibility, but most users are limited in the amount they can pay and cannot afford to always provision for peak demand. More commonly, users can recruit volunteers to setup a mirror of the web site so that load is spread among several servers. Ultimately, there are far too many "ordinary users" compared to the pool of volunteers willing to help. Another potential solution is proxy caching, but this is useful primarily from the perspective of an individual client for whom the cache is available. From the perspective of the web server, caching must be deployed at a wide number of sites in order to be effective at reducing load. One last alternative is to multicast content from the web server to a group of clients [1], but multicast is not widely deployed, requires loose synchronization among clients, and additional mechanisms to accommodate heterogeneous client bandwidths.

It is becoming increasingly clear that the best way for this majority of "ordinary users" to serve a large audience is through the use of peer-to-peer technology. A new type of data transfer called *swarming* leverages the cooperative nature of peer-to-peer networking to serve large numbers of users without placing a heavy burden on a centralized web server. With swarming, any user that has downloaded some piece of content from a server can then itself act as a server to other peers for that content. Because no single peer has the entire content, nor a high amount of bandwidth, peers download content from each other in parallel [2, 3], constructing the larger file from the pieces they collect. This frees the web server from having to deliver the entire file to all users; instead it gives a piece of the file to some users and then relies on those users to exchange data among themselves. This technology is most commonly used by BitTorrent clients [4], though other variations have also been proposed [5].

In this paper, we focus on the fundamental question of the scalability of swarming peer-to-peer content delivery. While existing systems such as BitTorrent indicate the feasibility of swarming in general, the load observed on these systems (for a single file) is still small enough that their scalability have not yet been demonstrated.[1] Accordingly, we have designed a simple swarming protocol and examined its scalability through a comprehensive, simulation-based performance evaluation. Our study demonstrates for the first time that peer-to-peer downloading can scale with offered load far beyond what client-server file transfers can deliver. We also show that swarming can smoothly handle flash crowds, with minimal effect on client performance. Finally, we demonstrate that swarming can scale to a wide range of file sizes, block sizes, and client bandwidths.

2 Simple Swarming Protocol

In order to study swarming performance, we have designed a simple swarming protocol. We believe this protocol generalizes the basic content delivery mechanism used by existing peer-to-peer software such as Gnutella, BitTorrent, and Slurpie [5]. Our design divides peer-to-peer content delivery into four key components: *swarming initiation*, *peer identification*, *peer selection*, and *parallel download*. While there are many design choices for each component, our goal is to use a simple yet effective design for each component. This enables us to study the performance of swarming delivery while minimizing complex dynamics and interactions among the components of the system. This makes it easier to correlate an observed behavior to a particular mechanism or parameter.

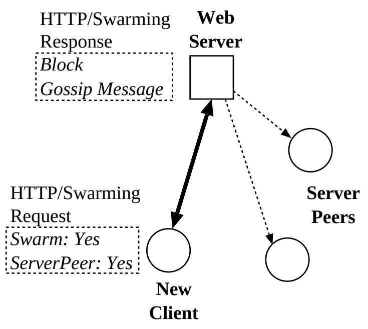
Our design also integrates the swarming protocol into a standard web server. The system uses client-server communication to bootstrap peer location and to

[1] One study [6] measured 51,000 peers in 5 days, but this is only 7 peers per minute.

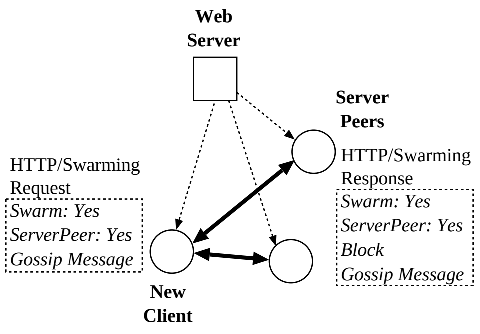
serve as a fallback in case a client's known peers all leave the network. The system uses peer-to-peer networking to deliver content and to discover additional peers through gossiping [7, 8]. Swarming is implemented on top of HTTP, providing backward compatibility and allowing for incremental deployment. It is important to note that the client may be either a web browser or proxy server. It should be simple to integrate swarming into existing proxies because they already include server functionality. We also note that swarming protocols can be used in a wide variety of content delivery systems, and are not limited to the web.

2.1 Protocol Overview

Swarming clients send regular HTTP requests to web servers, along with two additional headers. The Swarm header indicates that a client is willing to use swarming, and the Server Peer header indicates that a client is willing to provide the requested file to other peers (Figure 1(a)). Web servers that are not capable of swarming will simply ignore the unrecognized headers.



(a) A new client gets a block and a gossip message.



(b) The client finds and downloads blocks in parallel.

Fig. 1. Protocol Example

To initiate swarming, a web server gives clients a single *block* of the file and a *Gossip Message* (Figure 1(a)). A block is a portion of the file, typically tens or hundreds of kilobytes. The server determines the block size on a per-file basis; our performance evaluation looks at a range of file and block sizes. A gossip message contains a list of peers that are willing to serve portions of the same file. For each peer, the message lists the peer's IP address, a list of blocks the peer is known to have, and a time stamp indicating the freshness of this information.

When a client receives a swarming response (partial content plus a gossip message), it invokes a *peer selection* strategy to determine the subset of peers from which it will download content. A client's primary concern is to locate blocks of the file that it has not yet received. The client then begins downloading blocks from both the web server and the peers in parallel. In this study we are not concerned with fairness; thus, unlike BitTorrent, we do not include incentives to encourage clients to upload data.

Each transaction between a client and the web server or a peer includes a single block and a two-way exchange of gossip messages (Figure 1(b)). Requesting a single block at a time will naturally lead to faster peers delivering more blocks, resulting in proportional load balancing. Exchanging gossip messages with peers enables progressive *peer identification* – clients gradually learn about other peers in the system. This is needed because the initial pool of peers identified by the web server may refuse to serve the client, may disconnect from the network, may have low bandwidth connectivity, or may simply not have all of the content the client needs. Gossiping provides a low-overhead mechanism for peer identification while distributing this load away from the web server and among all peers.

When a peer receives a request for a block, it determines whether it will accept the connection based on its configuration or capabilities. The peer then delivers the requested block and exchanges gossip messages with the client. Once the peer has itself downloaded the entire file, it may decide to leave the system immediately or it may choose to linger and help additional peers. In our study we use a *lingering time* of zero in order to model the case where users are selfish and automatically exit after completion of the download. Existing studies on BitTorrent assume that clients linger for a longer time; one measurement study found that approximately 40% of the file was provided by clients who had already downloaded the entire file [6]. We do not want to assume that such charitable behavior will continue. When a peer disconnects from the system, it does not wait for any ongoing block downloads to finish. To reduce the amount of bookkeeping that is required, clients discard partially downloaded blocks.

As large numbers of clients attempt to download the same content, they form a dynamic mesh or swarm of peers. We can view this mesh as a *collaborative delivery system*, where peers with larger portions of the file or higher bandwidth will tend to serve greater numbers of clients.

2.2 Protocol Algorithms and Parameters

Our swarming protocol uses the following algorithms and parameters:

Swarming Initiation. We have chosen the conservative approach of swarming at all times. This enables the web server to be proactive with regard to load, so that it doesn't react too late to a flash crowd.

Peer Identification. Our goal for this component is to discover recent peers, since peers may leave the system at any time. Each client caches a record for the N_c peers with the most recent time stamp, then includes in its gossip messages the most recent N_g peers, where $N_g \leq N_c$. Clients must also gossip about peers that are no longer available, to minimize time wasted trying to connect to these peers. Accordingly, disconnected peers are represented as peers without any blocks of the file, and these records are shared in gossip transactions like any other. Eventually the record of a disconnected peer is discarded because its time stamp will never be renewed.

Peer Selection. We use a simple strategy that emphasizes content availability. Each client limits itself to N_d concurrent downloads. When choosing a new peer, clients choose the peer that has the most blocks that it still needs.

Parallel Download. Each client chooses N_d peers for parallel download, using the peer selection component, then continues to use this set unless a peer disconnects or runs out of blocks that the client needs. In either of these cases, the client drops the peer and then immediately invokes the peer selection component to choose a replacement. If none of the peers in the client's gossip cache have blocks that the client needs, then the client contacts the web server for some additional peers. When requesting blocks, a client selects a block randomly from those available when connecting to both the web server and peers. This ensures some amount of diversity in the content that is available, and increases the chance that a client can find a peer with content that it needs.

3 Performance Evaluation

We have conducted an extensive performance evaluation of swarming using our own peer-to-peer simulator built on top of some of the original *ns 1.4* code. We summarize our results here; for more details see our technical report [9]. Unless otherwise mentioned, we use the default swarming parameters given in Table 1.

Similar to congestion control studies, we use a simplified topology in which we model the Internet as a single router, as shown in Figure 2. This topology helps us to focus on the places where bottlenecks are likely to occur under high load – at the web server and peers. In order to focus on transmission delay, we set the propagation delay of all links to 1 *ms*. Most of our simulations use the basic scenario shown in Table 2, which models a server at a small company and clients with broadband access. We use a 1 MB file for most simulations because this provides a good comparison with a standard web server. Swarming can be used for much larger files, but a larger file in this case would render the web server completely useless.

We control the workload by varying the arrival rate of clients requesting the same file from the web server. For a given arrival rate, we randomly generate

Table 1. Default Swarming Parameters

Parameter	Value
N_d (Concurrent downloads)	4
N_c (Size of gossip cache)	64
N_g (Peers in gossip message)	10
Block Size	32 KB

Table 2. Basic Swarming Scenario

Parameter	Value
File size	1 Megabyte
Server bandwidth	$1Mbps$
Client bandwidth (down/up)	$1536Kbps$ / $128Kbps$

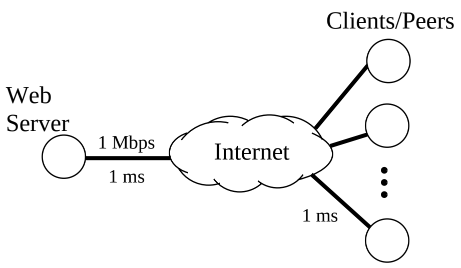


Fig. 2. Simulation Topology

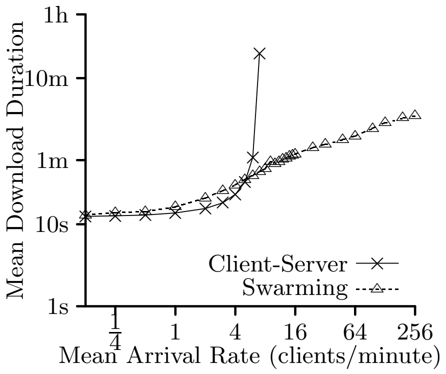


Fig. 3. Scalability with Load

client inter-arrival times using an exponential distribution. We also simulate a flash crowd by abruptly increasing the arrival rate for a given period of time.

When a client arrival occurs, we create a new client and it immediately begins its download. During the download, the client also serves content to other peers, then it leaves the system once its download is complete. While in the real world clients may be somewhat more polite, we opt for a conservative approach and hence underestimate the benefits of swarming.

Our primary performance metric is client download time. In particular, we are interested in how download time changes in response to increased load, rather than the absolute value of download time. We also measure the packet loss rate, the number of clients served by each peer, the number of blocks served by each peer, and a variety of other swarming-related metrics. Unless otherwise indicated, we begin each simulation with a warm-up period of 500 download completions, allowing the system to reach steady state behavior. We then collect data for 5500 download completions. For each experiment we conduct multiple runs of our simulations, average the results, and compute the 95% confidence interval. We do not include confidence intervals here because they are very small.

3.1 Scalability: Load

Swarming has excellent scalability with respect to load. In Figure 3, we plot the mean time a client takes to fully download a file versus the client arrival rate on a log-log scale. Client-server is unable to handle load beyond about 7 clients per minute; at this rate the arrival rate exceeds the departure rate. Naturally, this point will vary, depending on server bandwidth, file size, and load. Swarming, on the other hand, can serve at least 192 clients per minute, which translates to serving the one megabyte file to more than a quarter million people per day. This is an impressive feat for a $1Mbps$ access link. To serve an equivalent load using a client-server protocol would require, at a bare minimum, $28Mbps$. This could cost thousands, perhaps tens of thousands, of dollars per month!

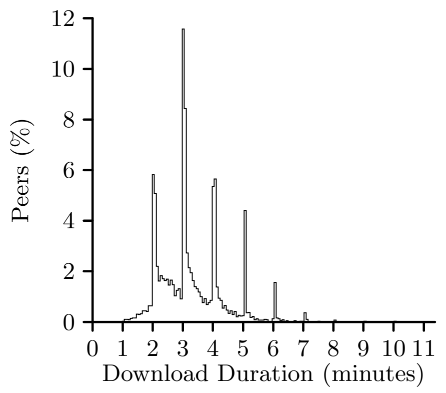


Fig. 4. Download times: 192 clients/min

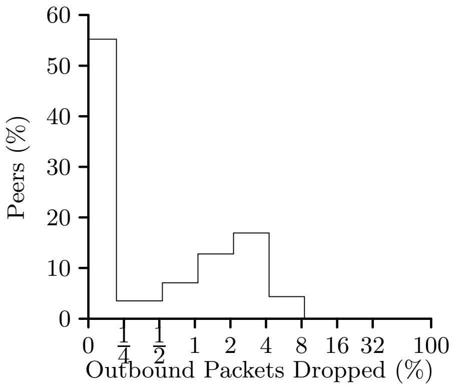


Fig. 5. Packet loss: 192 clients/min

Due to memory limitations we were unable to simulate higher loads, but a quick calculation suggests that any bound on swarming performance will not occur for at least an order of magnitude further increase in arrival rate.[2] This limitation exists for any scheme that relies on contacting a known, central point to initiate a download. At extremely high loads, swarming can incorporate a decentralized method for locating peers, such as PROOFS [10].

We observed several performance limitations with our swarming protocol, each of which could be fixed with further optimizations. First, our protocol imposes a small performance penalty under light load because the web server delivers all blocks, but with the added overhead of gossip messages. This could be eliminated by designing a dynamic server initiation component that uses swarming only when needed. Second, under very high load – 192 clients per minute – the server experiences severe packet loss. This is due to the web server providing at least one block to every client. This could be relieved by having the server only give a subset of the clients a block and require the rest to get the entire file from their peers. Finally, under high load a disproportionate number of download times are close to multiples of 60 seconds (Figure 4). It is important to note that the download times are measured individually from the start of each client; thus, this pattern does not indicate synchronization of flows within the network. This behavior is caused by our use of a 60 second time out to detect dead connections, and the high congestion at the server causes many clients to time out. This suggests that alleviating server congestion will also result in significant improvements for the peers.

Our swarming protocol otherwise performs very well. Even at the highest load we simulated, peers experience very little packet loss (Figure 5). Furthermore, during high load the burden of content delivery is spread evenly among the peers. At the highest load we simulated, roughly 60% of the clients serve less than one

[2] We assume a single 1500-byte packet is used to transmit the referral information. The $1Mbps$ server can transmit $2^{20}/(1500 * 8)$ of these per second, or 5242 per minute.

megabyte. Nearly all of the clients upload less than two megabytes. Re-serving the file once or twice is fair, so this behavior is quite good.

3.2 Scalability: Flash Crowds

While good steady-state behavior is important, web servers must also be able to cope with extreme bursts of activity called flash crowds. We simulate the effect of a flash crowd by abruptly increasing the arrival rate for a fixed period of time. The steady state load in this section is 6 clients per minute. For client-server transfer we introduce an impulse of 12 clients per minute, lasting for one hour. For swarming, we provide a more challenging flash crowd by increasing the flash crowd rate to 120 clients per minute![3] In both cases, after the flash crowd passes, we simulate the steady state load until the web server is able to recover. The results are presented in Figure 6 and Figure 7, where each data point represents the mean download time for all downloads finishing in the previous 1000 seconds.

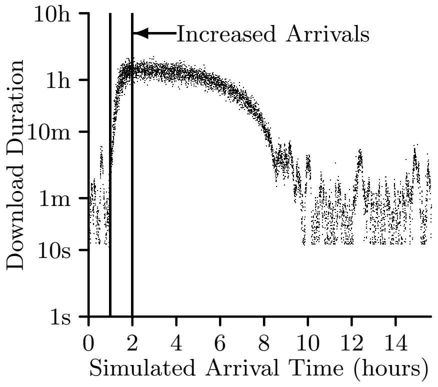


Fig. 6. Client-Server flash crowd

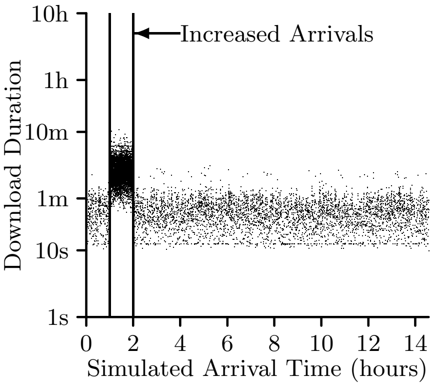


Fig. 7. Swarming flash crowd

As can be seen from these figures, swarming enables a web server to smoothly handle large flash crowds that would otherwise bring content delivery to a crawl. It maintains reasonable response times as the crowd arrives, and dissipates the crowd quickly. With the traditional client-server approach, the arrival rate of the clients quickly exceeds the service rate, and the server does not recover until long after the arrival rate decreases. Note that we intentionally do not model clients giving up during a long download period so that our model for the web server matches that of swarming. This is not unrealistic, as Kazaa users have been observed to wait for a day to download MP3 files [11].

It is particularly impressive that we achieve this result using a conservative and unoptimized swarming protocol. With additional optimizations, the server

[3] In order to fill out the graph, we ran this simulation for 12,000 completions.

should be able to handle even larger flash crowds. In fact, by utilizing Gnutella or PROOFS [10] to locate peers under extremely high loads, *swarming can be made effectively immune to flash crowds.*

3.3 Scalability: File and Block Size

Swarming also scales well with the size of the file, allowing a small user to easily serve large files (e.g. multimedia). We demonstrate this result in Figure 8, which shows the mean download time for both swarming and client-server as a function of the file size. For this simulation we use an arrival rate of 4 clients per minute, with the same basic scenario given in Table 2. Varying the file size is similar to varying the arrival rate in that both cases increase the load on the web server. Swarming exhibits only a linear increase in download time as the size grows; note that a linear increase is the best it can do because the file size is growing while the bandwidth at the client stays constant. For file sizes of two megabytes or larger the client-server protocol is unable to enter steady-state.

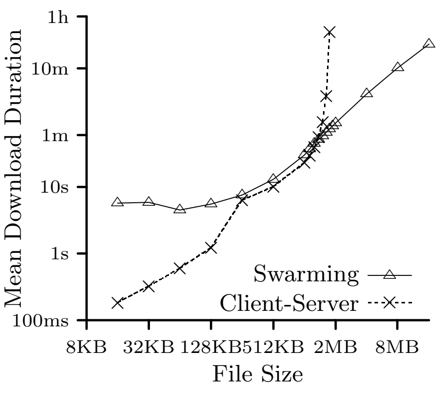


Fig. 8. Scalability with file size

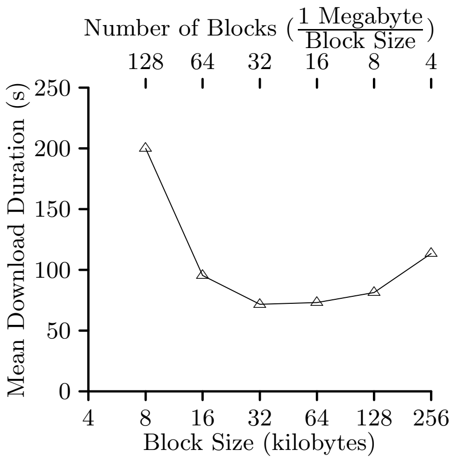


Fig. 9. Scalability with block size

An interesting result from this simulation is that gossiping can impose significant overhead when the block size is small. For this simulation the number of blocks is 32, regardless of file size. Thus as the file size decreases, the gossip message becomes large relative to the size of the data. This is shown in Figure 8, in the region where file size is less than 256 KB; the mean download time never goes below 4 seconds. Despite this overhead, swarming will eventually outperform client-server for small files as the arrival rate increases. Nevertheless, this is clear evidence that swarming can benefit from dynamic server initiation.

To fully explore the effect of block size on swarming performance we conducted a series of simulations with varying block and file sizes, using an arrival rate of 16 clients per minute. Figure 9 shows the results of one these simulations, using a file size of 1 megabyte, from which we can identify two trends.

First, download time increases as the block size decreases. As the block size becomes smaller, the transmission delay incurred by the client transmitting a gossip message becomes a significant part of the overall delay. Second, as the block size increases the download time increases slightly for large blocks. This is a result of the "last block problem", which occurs when the last block to be downloaded is coming from a slow source. Our results show that for a block size of 256 KB the last block consumes 35% of the download time. BitTorrent solves this problem by simultaneously downloading the last block from multiple sources, although this results in redundant data transmission

3.4 Scalability: Client Bandwidths

Finally, swarming can handle a wide range of client bandwidths. This is an important result since some peer-to-peer protocols collapse when too many low-bandwidth users enter the system; the original Gnutella protocol had this flaw. To address this concern, we conducted a variety of simulations using different mixtures of clients drawn from three classes: modem ($56Kbps/33Kbps$), broadband ($1536Kbps/128Kbps$), and office ($43Mbps$). We assign each class of users a different probability, then randomly assign new clients to one of these classes according to these probabilities. Other than client bandwidth, the rest of the scenario is the same as Table 2.

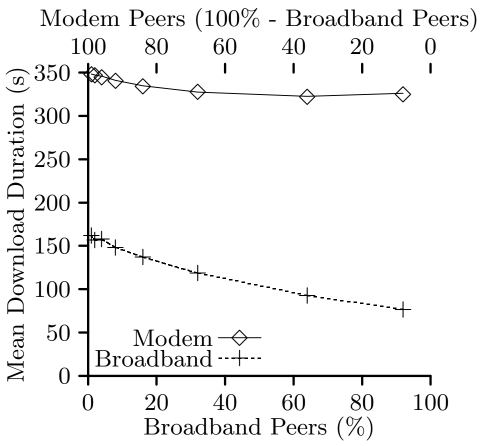


Fig. 10. Impact of low-bandwidth clients

As an example of our results, Figure 10 plots the mean download time for a combination of broadband and modem users. As the percentage of modem users increases from 10% to 99%, the download time for broadband users increases by roughly a factor of two. While this is a significant increase, the system clearly continues to function well despite an overwhelming number of low-bandwidth users. We see a similar result for office users – their mean download time increases by a factor of 3 for the same changes in the mix of broadband and modem users. Note that broadband users do not get a significant benefit from office users because office users do not linger after downloading the file and because we are not using any kind of bandwidth-based peer selection. Finally, the performance of modem users is relatively unchanged by large numbers of higher-speed users because their access link remains a bottleneck.

4 Related Systems

A number of peer-to-peer content delivery systems have been developed recently that use the concept of swarming. BitTorrent [4] is notable as a swarming system because it is currently used to transfer large files, such as new software releases, to hundreds of peers. BitTorrent uses a centralized host for peer identification and includes mechanisms to try to enforce fairness among peers. Slurpie [5] forms a content delivery mesh, with the main goal of trying to minimize client download time. Slurpie clients utilize bandwidth estimation to adapt to changing conditions in the mesh and select peers for data transfer. With both CoopNet [12] and Pseudoserving [13], a web server gives clients a list of possible peers, and the client chooses a single peer from which it downloads the entire content.

Several other peer-to-peer content-delivery protocols take different approaches. Splitstream [14] divides content into several stripes and then multicasts each stripe to a different application-layer multicast tree. Peers join as many trees as they want, and nodes try to spread the load of content delivery among themselves. Bullet [15] organizes peers into a mesh and delivers disjoint sets of data to peers. Peers are individually responsible for discovering and retrieving data.

Other peer-to-peer systems focus on enabling web servers to cope with high load. The Backslash system [16] forms a collaborative network of web mirrors. An overloaded web server then redirects clients to a cached copy of the content located at one of the collaborating sites. PROOFS [10] uses a peer-to-peer network of clients to cache popular content. When a client is unable to download content from a web server, it queries the peer-to-peer network to see if any other user has a copy of the desired content.

5 Conclusions and Future Work

The power of swarming as a file transfer mechanism is that it actually uses scale to its advantage – system capacity increases with the number of peers participating. Peers spread the load of content delivery over the entire network and share the burden of peer identification with the web server; this prevents server overload and avoids network congestion. Clients utilize parallel download to protect themselves against peer instability, which would otherwise hinder a peer-to-peer application. Moreover, swarming is clearly an economical solution for the web server because it does not have to pay for the bandwidth used for peer-to-peer communication. Clients have an incentive to participate in swarming because the alternative is that everyone suffers from a congested server.

Our study lays the groundwork for future research in many interesting areas. A dynamic server initiation component should be able to decide when to use client-server transfer (for small unpopular files) and when to use swarming (for large or popular files). It should also be able to switch exclusively to swarming during high loads. Other avenues of research include applying swarming to dynamic content, bandwidth-based and distance-based peer selection, dynamic adjustment of the number of concurrent downloads, peer performance monitoring, more efficient gossiping, and ways to encourage cooperation.

References

1. Clark, R.J., Ammar, M.H.: Providing Scalable Web Services Using Multicast Communication. Computer Networks and ISDN Systems **29** (1997) 841–858
2. Byers, J., Luby, M., Mitzenmacher, M.: Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads. In: IEEE INFOCOM. (1999)
3. Rodriguez, P., Biersack, E.W.: Dynamic Parallel-Access to Replicated Content in the Internet. IEEE/ACM Transactions on Networking (2002)
4. Bram Cohen: Bit Torrent (2004) http://bittorrent.com.
5. Sherwood, R., Braud, R., Bhattacharjee, B.: Slurpie: A Cooperative Bulk Data Transfer Protocol. In: INFOCOM. (2004)
6. Izal, M., Urvoy-Keller, G., Biersack, E.W., Felber, P.A., Hamra, A.A., Garces-Erice, L.: Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In: Passive and Active Measurement Workshop. (2004)
7. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: ACM Symposium on Principles of Distributed Computing. (1987) 1–12
8. Karp, R.M., Schindelhauer, C., Shenker, S., Vocking, B.: Randomized rumor spreading. In: IEEE Symp. on Foundations of Computer Science. (2000) 565–574
9. Stutzbach, D., Zappala, D., Rejai, R.: Swarming: Scalable Content Delivery for the Masses. Technical Report UO-TR-2004-01, University of Oregon (2004)
10. Stavrou, A., Rubenstein, D., Sahu, S.: A Lightweight, Robust P2P System to Handle Flash Crowds. In: IEEE ICNP. (2002)
11. Gummadi, K.P., Dunn, R.J., Saroiu, S., Gribble, S.D., Levy, H.M., Zahorjan, J.: Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In: ACM Symposium on Operating Systems Principles. (2003)
12. Padmanabhan, V.N., Sripanidkulchai, K.: The Case for Cooperative Networking. In: 1st International Workshop on Peer-to-Peer Systems. (2002)
13. Kong, K., Ghosal, D.: Mitigating Server-Side Congestion on the Internet Through Pseudo-Serving. IEEE/ACM Transactions on Networking (1999)
14. Castro, M., Druschel, P., Kermarrec, A.M., Nandi, A., Rowstron, A., Singh, A.: SplitStream: High-Bandwidth Content Distribution in a Cooperative Environment. In: International Workshop on Peer-to-Peer Systems. (2003)
15. Kostic, D., Rodriguez, A., Albrecht, J., Vahdat, A.: Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In: SOSP. (2003)
16. Stading, T., Maniatis, P., Baker, M.: Peer-to-Peer Caching Schemes to Address Flash Crowds. In: 1st International Workshop on Peer-to-Peer Systems. (2002)