SeCond: A System for Epidemic Peer-to-Peer Content Distribution[◊]

Ali Alagöz[*], Öznur Özkasap[°] and Mine Çağlar[^]
[*] Department of Computational Sciences and Engineering
[°] Department of Computer Engineering
[^] Department of Mathematics
Koc University, Istanbul, Turkey
{alialagoz|oozkasap|mcaglar}@ku.edu.tr

***Abstract*—We propose an efficient cooperative content distribution protocol in which the cooperation among participants is based on a peer-to-peer (P2P) paradigm. Our main contribution to P2P content distribution is the use of an epidemic communication approach. Since epidemic algorithms are easy to use, robust and adaptive to dynamic conditions, they have found several application areas in distributed systems such as failure detection, data aggregation and database replication. However, they have not been used in cooperative content distribution before. In addition to the use of epidemic algorithms for state exchange among peers, we propose some methods in order to increase utilization of system resources during distribution of the files. We demonstrate effectiveness and scalability of the protocol through our simulation model.**

I. INTRODUCTION

As the usage of the Internet grows up, the number of people preferring to share their contents increases. However, accessibility of the content is affected badly whenever there are lots of end systems retrieving that content simultaneously. It is well known that traditional client-server based solutions are not appropriate for distribution of popular files such as software updates or CD images. When several peers strive to achieve a file at the same time, the file server hence the overall system may fail easily. To avoid such failures, well designed protocols addressing content distribution should have the following properties:

- *Scalability:* As the popularity of the released content increases, the number of users trying to achieve the file simultaneously also increases. Hence, a well designed content distribution protocol should be able to handle large set of users at the same time.
- *Adaptive to dynamic arrivals and departures:* During distribution of the content, for most of the cases users' arrival rate and arrival times may not be anticipated before. Similarly, a user may leave the system without notice. An efficient protocol should be able to operate under dynamic conditions.
- *Easy to deploy:* Although some of the protocols seem to operate well in theory, it is hard to deploy them in real life. That might be due to the requirement of router support or difficulties in the implementation of protocols.
- *Heterogeneity:* Among millions of geographically distributed users, download and upload bandwidths, hardware properties (such as CPU speed) differ from one user to another. Similarly, different network conditions may be observed at different locations. In order to operate efficiently, the platform for content distribution has to take these differences into account.

In this study, we propose a protocol that alleviates the load imposed on the file server while minimizing download times of users at the same time. Our cooperative content distribution protocol, called SeCond, disseminates the load of distribution among all peers in the system. Cooperation among system participants is based on a P2P paradigm similar to BitTorrent [1] and Slurpie [2]. Namely, while peers are downloading blocks of the file, they also upload the blocks they have downloaded before. However, informing other peers about available blocks is based on epidemic information dissemination. Moreover, partial views of the peers are continuously updated in order to increase utilization of system resources. By use of epidemic algorithms, dynamic arrivals and departures can be easily handled. Epidemic or probabilistic dissemination algorithms have many advantages. They are easy to implement and inexpensive to run. During propagation, load on the links and users is distributed uniformly. In case of dynamic user arrivals and departures, it does not require an extra effort in order to reconfigure itself. Most importantly, they are inherently scalable.

Remaining part of the paper is organized as follows. Related work and comparison with our study are described in Section II. Details of our model SeCond is explained Section III. Simulation model and simulation results are given in Section IV and V, respectively. Section VI concludes the paper and states future work.

II. RELATED WORK

We group existing solutions for alleviating the load on the source of the file in a distributed system as follows:

A. Infrastructure-based Solutions

One type of solution in this category is mirroring or replication of the server. Akamai [3] is the best example deployed in the Internet for years which runs tens of thousands of servers. Although mirroring seems to be the best solution for increasing server's resources, in order to reply large number of requests in flash-crowded scenarios,

[◊] This work is supported in part by TUBITAK (The Scientific and Technical Research Council of Turkey) under CAREER Award Grant 104E064.

serving capacity of the file server should be increased linearly with the number of requests which is impossible in most of the cases.

Another approach is caching hierarchies. One example to this kind of solutions is Squid [4]. It stores the requested Internet objects on a system closer to the requesting site than to the source in order to reduce load on the server and client return time.

All the solutions under this category require infrastructure support which is usually an expensive supply. Moreover, demand on the content should be predictable so that content providers can make provision against bottlenecks in data distribution. However, it is not possible to anticipate the demand accurately all the time. Misconstructions may lead to waste of resources.

B. Cooperative Content Distribution Solutions

Instead of only requesting blocks of files from server, nodes may also be involved in the distribution of a popular file. Although the idea lying behind the cooperative solutions seems very simple, implementing such a protocol organizing nodes effectively in a dynamic system where arrival and departure of nodes occurs very frequently is a challenging topic. Cooperative solutions can be classified as follows:

1) *Multicast:* Multicast solutions strive to deliver the information to a group of destinations using the most efficient strategy to deliver the messages over each link of the network only once and only create copies when the links to destinations split. A criterion to classify multicast schemes is the degree of reliability they offer. First group of protocols offering strong reliable multicast schemes such as atomicity and message delivery ordering suffer from scalability. Moreover, their performance can become unstable under stressed conditions [5]. Other group of protocols offers better scalability, but best-effort reliability. For instance, Bimodal Multicast is a protocol offering scalability and probabilistic reliability as stated in [6]. Although it provides a high level of reliability and scalability, it is not adaptable to dynamic systems. In fact, most of the multicast solutions do not intend to operate well under conditions where node arrivals and departures occur frequently. Moreover, nodes do not decide from where they download a block dynamically taking network conditions into consideration.

2) *Peer-to-Peer Cooperative Protocols:* P2P systems create a platform where people find lots of files to transfer, but generally they do not intend to disseminate a popular file. Popular file sharing applications such as KaZaA [7], Gnutella [8], and e-donkey/e-mule [9] are good examples of this kind of systems where peers are organized together so that they can exchange different files. However, the main goal of these applications is locating sources for the desired files. Two important examples of P2P protocols whose main goal is organizing the peers sharing and requesting the same file into an overlay network are BitTorrent and Slurpie. Peer-to-peer cooperative protocols let the end systems decide the source for the data they strive to achieve locally. This locality increases the utilization of the system resources and enables parallel downloading [10].

Similar to SeCond, both BitTorrent and Slurpie construct meshes among the end systems trying to obtain copies of the file. While peers continue the download, they also upload the blocks of the file they have obtained. This cooperation alleviates the load on the original source for the file.

BitTorrent executes a rarest-first block selection policy in order to decrease the sparsity of blocks in the system. Whenever, a block cannot be obtained from the other peers in the system, peers try to get that block from the original seed. However, while peers are approaching the end of the download, simultaneous requests for missing blocks may cause a bottleneck on the seed. Slurpie avoids this problem by using a random back off policy. Whenever a peer cannot obtain a block from the other peers, it goes to the server with some probability. Our SeCond algorithm deploys a different back off policy. If a peer has initiated a block download from any source in a given time interval recently, it does not strive to obtain a block from the server. This allows the other peers, which cannot obtain a block for the same time interval, to initiate a block download from the server with an acceptable download rate. Moreover, since the main goal of our protocol is to make copies of the blocks available in the system as fast as possible, we limit the number of parallel downloads from the file server also. Otherwise, especially for flash crowded scenarios, download rates of transmissions for the blocks obtained from the file server may decrease below an acceptable lower bound. As a result, replication of the blocks in the system may take longer times.

Another point that SeCond differs from Slurpie and BitTorrent is the propagation of states during the dissemination of the file. Instead of multicasting available blocks whenever a new block is obtained, we deploy a gossiping mechanism to propagate peers' states during the dissemination of the content.

BitTorrent deploys a rate based "tit-for-tat" mechanism to avoid free riding [11]. However, as stated in [12], that policy is not effective in preventing unfairness especially for heterogeneous systems. In SeCond, instead of forcing peers upload to peers from where they can download, each peer utilizes its upload bandwidth independently. This leads to an increase in utilization of system sources. In return, it increases the performance of the file distribution. However, peers give priorities to peers from where they have downloaded more. The details of this procedure are given in the description of our model in the next section.

There are studies examining the performance of BitTorrent-like systems. In [13], the performance of BitTorrent is studied by representing the system with a fluid model. The study given in [14] investigates real life performance of BitTorrent by collecting data for a five month period. Another analysis for BitTorrent [15] shows how the performance of the system is affected in case of exponential decrease in peer arrival rate. Similar to [11], it is stated that fairness cannot be achieved for heterogeneous systems.

Another protocol using epidemics in order to disseminate information and manage the membership is Newscast [16]. In Newscast, caches of peers hold neighbors' descriptors. A descriptor for a peer consists of peer address, creation time of that descriptor and information of the corresponding peer. At predefined time intervals, each peer creates its own descriptor. Since cache has a fixed size, that descriptor is replaced with the oldest one. After that, a peer is selected from cache and an

exchange of states occurs with the selected peer. Then views are merged and old entries are removed to keep cache fresh. By this way, dynamic joins and leaves are handled easily by the protocol. However, this protocol does not directly target distribution of large files especially in flash-crowd scenarios.

III. MODEL DETAILS

There are two servers in SeCond, namely index server and file server. Index server is responsible for holding Ids of peers currently downloading the file. File server is the place where the complete file is stored initially. To initiate a download, a peer should register itself to index server. After registration, index server returns a list of peers currently downloading the file which is called *view* of the peer. At predefined time intervals, peers send gossip messages containing Ids of available blocks to randomly selected peers from their view. By the help of these gossip messages, peers upload and download blocks of the file from other peers and update their views as well. Each peer continues updating its partial view during dissemination in order to maximize utilization of system sources. Below we give details of our protocol.

A. Mesh Construction

As mentioned, index server is responsible for helping peers to find each other. It is similar to the tracker of BitTorrent. The list returned by the index server contains peers *randomly* selected among the ones registered before. To update global view of the index server, active peers report their state to the server periodically. Moreover, if a peer realizes that one of the peers in its partial view is not active, it reports that dead peer also. A peer containing all blocks of the file is called a *seed*. Although a peer has become a seed, it may not continue remaining in the system. In order to alleviate load of the server, we have to utilize resources of existing seeds in the system efficiently. To achieve this aim, upon becoming a seed, peer reports its state to the index server. Since propagating available blocks of other peers to seeds is meaningless, mesh of a recently joined peer is constructed among the ones which are not seeds. However, index server asks the seeds to add new peers to their partial view. Mechanism for adding a peer to the partial view of another peer is explained in the Section III D below.

B. Structure of the File

File server is the original source for the file. In order to enable parallel downloading, the file is broken into blocks. Since small block sizes increase the TCP overhead, the size of a block is set to 256KB. File server refuses uploading if the number of peers served simultaneously reaches a particular value. This value may be adjustable with respect to upload capacity of the file server and average download capacities of the peers. Refused peer does not try to download a block from the server for a given duration.

C. State Propagation

Peers propagate their states via gossiping. A gossip message holds the Ids of the downloaded blocks of the peer. Targets for that gossip message are uniformly chosen from the partial view of the peer. Upon receiving a gossip message, receiver peer looks for the blocks that it does not have but the sender has. If there are such blocks, that peer requests them from the sender. It should be noted that sometimes requesting blocks may not initiate block downloads. Another important point is that gossiping is not reciprocal. Namely, holding some peer in the partial view and sending gossip messages to that peer does not necessitate that destination peer also sends gossip messages to counterpart.

D. View Update

In order to increase the performance of content distribution, cooperation among the peers should be maximized. Hence, peers should be able to upload and download blocks. However, the peers given by index server at registration may have left the system or approximately all of the block transmissions among the peers might have been done. This leads to waste of link capacities of the peers. In such a case, a peer is forced to download blocks from the server. To avoid this situation, peers update their partial views during dissemination. This is done again via gossiping. When a peer receives a gossip message from an unknown peer, it looks for a place in its partial view in order to add sender. If there is a free space, it directly adds the unknown peer to its view. However, if the capacity of the list is full, then it looks for a passive peer. A peer in the partial view that has not downloaded any block in a given time interval is said to be a passive peer. If there is a passive peer, it is replaced with the new one and the partial view is updated.

E. Downloading Decisions

Whenever a peer is able to download blocks from other peers, it prefers downloading blocks from them, not from the server. However, if it cannot initiate any block download for a while, it has to visit the server. Since peers are asynchronous, it is not expected that all peers try to download a block from the server simultaneously. Moreover, similar to a file server, number of peers that a peer serves at the same time is limited.

Whenever a peer realizes a source holding some blocks of the file that it has not, it tries to obtain missing blocks from that source. In order to increase density of the blocks in the system, the source prefers to forward the least uploaded blocks. The decision for the density of the blocks is given locally. Same uploading policy is also applied on the original file server.

Requesting some blocks from a source does not guarantee the initiation of block download. Since the concurrent upload capacity of the peers is limited, the request may be refused. In such a case, not to refuse a request, peers put the received requests into their upload queues. However, these are not first in first out queues. A priority is given to peers that upload more to the owner of the queue.

IV. SIMULATION MODEL

In this part, the underlying model and assumptions that we used to simulate SeCond are described. The code is implemented in Java and it is a time-discrete event-based simulation.

A. Shared File

In our simulation model, the file is stored on a file server and is broken into blocks of size b. Hence, the number of blocks in a file, c, is equal to ceiling of s/b

where *s* is file size. Parameters *b* and *s* are set to 256 KB and 700MB respectively. Thus, *c* is 2800. Whenever, a peer completes downloading of a block successfully, it starts sharing that block. At the beginning of the simulation, there is only one seed which is the file server. It is assumed that peers completing the download do not leave the system. They stay in the system and serve as seeds.

B. Peer Arrival and System Size

In order to download the shared file, a peer should register itself to the index server. After registration process, the server returns a list of peers downloading the file currently. Number of peers requesting the file is set to 1000 in the simulation unless another value is stated. For scalability analysis, system size is increased up to 5000 peers. Peer arrivals occur with constant rate of 1 peer per minute.

C. File Distribution and Propagation States of Peers

In the beginning, file server is the only one containing blocks of the file. As the simulation continues, peers owning blocks of the file increase in the system. In order to download blocks from other nodes instead of the server, peers should be informed about the available blocks. Propagation of available blocks is done by the help of gossiping. At each predefined time interval called gossip interval, each peer sends gossip messages to *f* distinct peers chosen from its partial view. These gossip messages contain Ids of downloaded blocks and *f* is called the fan-out parameter. When a peer receives a gossip message from another peer, it checks whether there is any block that it does not have but the sender has. If it has available download bandwidth, it tries to get missing blocks from the sender. If a peer cannot download any block from a peer or file server for a time interval called *server visit interval*, it requests missing blocks from the server. If server accepts another parallel downloading, it uploads the least served block among the desired blocks. However, if any block download is completed within the server visit interval, counter for the server visit interval is set to zero.

D. Server, Peer and Link Properties

The file server and peers have maximum upload bandwidth capacity that is maximum number of peers served simultaneously, called *upload capacity*. Peers also have maximum *download capacity*. Whenever a peer or server reaches its maximum upload capacity, it does not serve anyone else until a peer completes downloading. Uploader peer shares out its bandwidth *equally* to downloader peers.

An example to restrictions and calculations given above is illustrated in Figure 1. The list indicated as B[*block id list*] represents blocks downloaded by the corresponding peer up to that time, *P* represents the number of peers that can be served additionally, and [*upload capacity, download capacity*] represents the download and upload bandwidth of the peer. T#:[*a*,*c*] represents the transmission of block *a* with transmission rate *c*. Unit of bandwidth capacities is Kbps. In Figure 1, shared file consists of 5 blocks. At round 1, peer 2 initiates transmission of block 1 from peer 1. Bandwidth initially reserved for this transmission is equal to, minimum of upload capacity of peer 1 and download capacity of peer 2, 128 Kbps. Assuming that no block transmission is

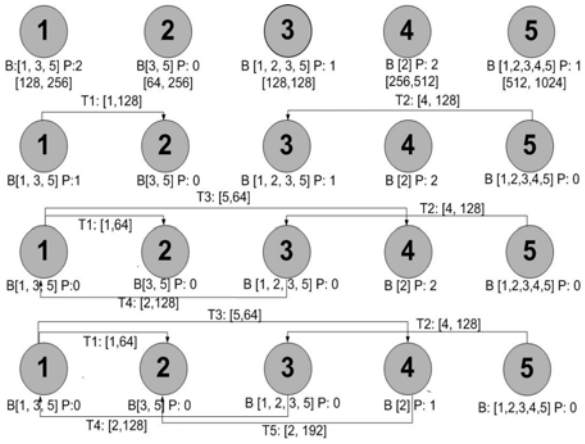


Figure 1 Illustration of block dissemination in SeCond

completed, peer 1 can serve at most 1 other peer in the remaining rounds. Similarly, a transmission of rate 128 Kbps from server is started in round 1. At round 2, peer 1 starts to download block 2 from peer 3 with rate 128 Kbps. However, peer 4 also initiates a transmission from peer 1 at round 2. Since upload capacity of a peer is shared out equally among downloaders, both of the rates reserved for transmissions 1 and 3 are set to 64 Kbps. At round 3, peer 2 initiates a block download from peer 4. Reserved bandwidth for this transmission is equal to minimum of download capacity of peer 2 and upload capacity of peer 4, which is 192 Kbps. Remember that peer 2 uses 64 Kbps of its download capacity for transmission 1. Without completion of any block transmission, no one can initiate a transmission after round 3.

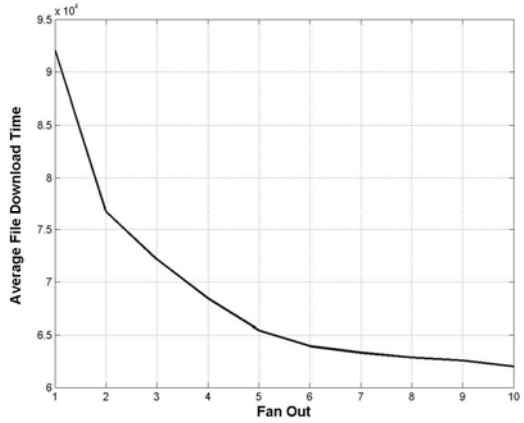
There is a loss probability for a message during transmission over links. That probability is set to different values for gossip packets and block packets. Since it is assumed that block transmission is done over TCP and gossip transmission is done over UDP, the probability of loss in gossip packet transmission is set to a higher value.

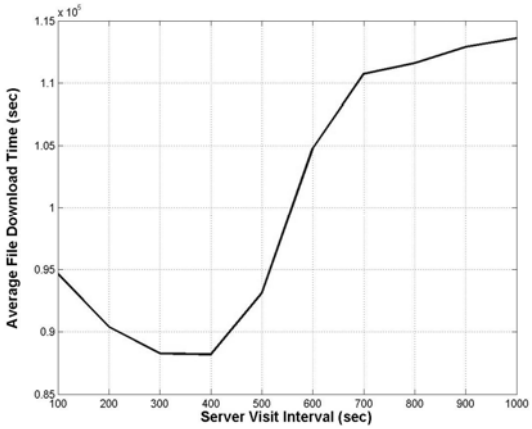
V. SIMULATION RESULTS

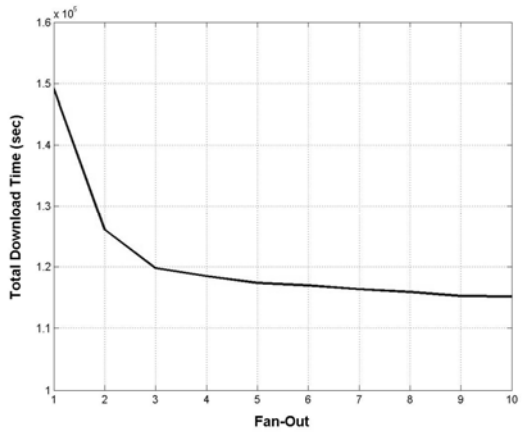
Terminology of simulation evaluation is given below:

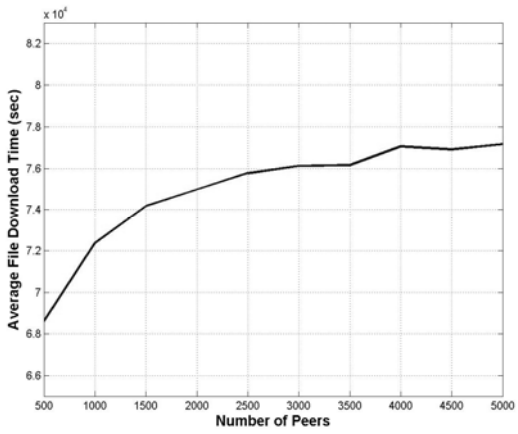
- *Average File Download Time:* File download time of a peer is the time between the initiation of download and completion of the download. Average of all file download times of the peers in the system is evaluated by this parameter.
- *Number of Blocks Downloaded from Server:* A peer obtains blocks of the file either from the file server or from another peer. This parameter counts the number of blocks downloaded from the server.
- *Total Download Time:* Time between arrival of the first peer and completion of last file download is called as total download time.

Although this is a time-discrete simulation, we may think one unit time as one second. System size is set to 1000 peers. It is assumed that all peers are identical. The upload and download rates of the peers are 512 Kbps and 128 Kbps respectively. Upload capacity of the file server is 1024 Kbps. A peer can serve at most 4 peers simultaneously and partial view of the peer is set to 10. In the calculation of transmission delay of a block,







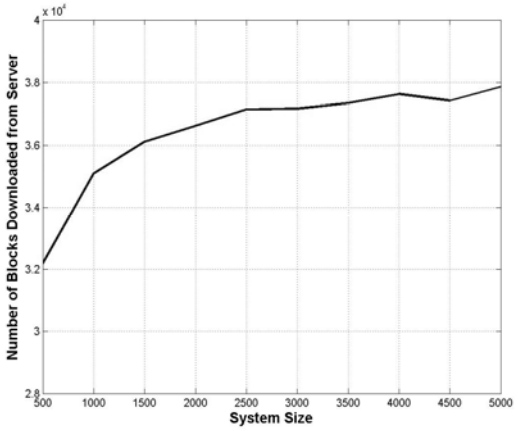


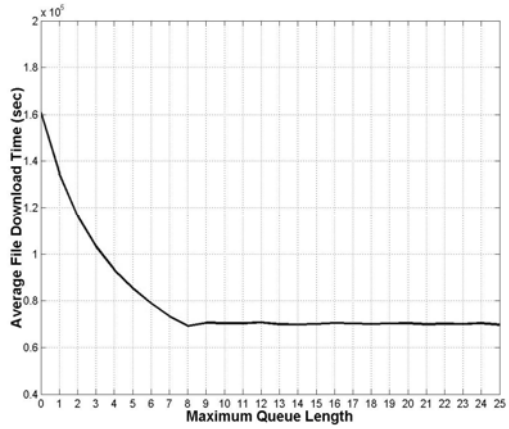
propagation delay is ignored. We believe this assumption has no significant effect on simulation results since dissemination time of the file is dominated by block transfers.

As it is mentioned in the model description, peers propagate their states via gossip messages. At each gossip round, a peer sends its state information to f different neighbors randomly selected from its partial view. This parameter, f, is called as fan-out parameter. To see the effect of fan-out parameter, we have adjusted fan-out parameter and left others unchanged. As shown in Figure 2, forwarding gossip messages to more peers have no continuous positive effect on file distribution. Average file download times stays approximately constant after some point. Similarly, we have not seen any significant decrease in total download times after that point as it can be seen from Figure 3. We have to point out that forwarding unnecessary gossip messages causes an additional load on the network.

Although peers cooperate with each other to obtain blocks of the file, original source for the file is the server. Taking the probability of nonexistence of another seed in the system into consideration, peers have to visit the server. The question is at which periods peers should go to the server. In SeCond, if the time passing through the initiation of the last block transmission from any source exceeds server visit interval, that peer tries to obtain a block from the server. However, since the number of peers served by the server simultaneously is limited, visiting the server frequently may not have additional benefit for the peer. Additionally, these unnecessary requests may lead to a bottleneck in the server. On the contrary, setting the server visit interval to a large number may decrease the utilization of the server resources. To see how the performance of the distribution of the file is affected, we have run simulations for different server visit intervals.

An interesting result that can be observed in Figure 4 is that requesting blocks from the server more frequently decreases the system performance after some point. As we know, in order to make use of system resources, blocks should be uploaded to the system as fast as possible. However, whenever we increase the number of peers downloading blocks from the server simultaneously, average upload times of those blocks may increase. Hence, until peers obtain blocks they can share, their resources may remain idle. The reason for the decrease in average file download time until some point may be the utilization of system resources. On the contrary, we see from the Figure 4 that if server visit interval is increased after a certain point, average file download time increases because the peers wait too long to download a block from the server although they cannot initiate any block transmission for a long time.





For a well designed peer-to-peer file distribution protocol, it is important to handle large number of requests simultaneously. In fact, this issue is the base of our motivation for this work. In order to see scalability, we increase the number of peers requesting the file. As shown in Figure 5, when we increase the number of peers requesting the file from 500 to 5000, average file download time increases only 13% of its initial value. Moreover, if classical client-server based approach was used, number of blocks uploaded from the server would increase linearly with respect to the system size. However, as depicted in Figure 6, there is an 18% increment in number of blocks uploaded by the server as the system size scales up. Another inference is that there is a positive correlation between average file download time and number of blocks uploaded by the server. In fact, this inference has been obtained in all simulation settings.

Initiation of a block download after the arrival of a request message depends on the upload capacity of the source at that time. If the number of peers served by that source has reached its maximum value, any download request will be added to the queue of the uploader peer. Whenever a continuing download is completed, it initiates a block upload selected from the queue. In real life implementation some priorities may be given to some requesters, but in simulation settings we applied first in first out policy. Figure 7 shows that increasing length of queues affects the file dissemination positively until some point. Actually, we may expect that setting maximum queue size to large numbers may have negative affect since a peer may wait too long in the queue to download a block although it can download that block from another source. Moreover, in real life waiting in the queue does not guarantee initiation of the download, since in a dynamic environment the source may leave the system any time.

VI. Conclusion and Future Work

In this paper, we have described our protocol SeCond that that alleviates the load imposed on the file server while minimizing download times of users at the same time. It targets distribution of large contents especially in flash crowded scenarios. We have developed a simulation model of SeCond and analyzed its behavior via several simulations. As ongoing work, we analyze the performance of the protocol for different peer arrival and departure rates. Moreover, we aim to develop an analytical model for our protocol.

References

[1] B. Cohen, "Incentives build robustness in BitTorrent," *P2P Economics Workshop*, Berkeley, CA, 2003.

[2] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: A cooperative bulk data transfer protocol," IEEE *Infocom 2004*, Hong Kong.

[3] Akamai: http://www.akamai.com.

[4] Squid: http://www.squid-cache.org.

[5] R. van Renesse, "Why bother with CATOCS?" *ACM SIGOPS Oper. Syst. Rev. 28*, pp. 22–27, Jan. 1994.

[6] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," ACM Trans. Comput. Syst., vol. 17, pp. 41–88, May 1999.

[7] KaZaA: http://kazaa.com/.

[8] Gnutella: http://gnutelliums.com/.

[9] E-donkey: http://edonkey2000.com/.

[10] Pablo Rodriguez-Rodriguez and Ernst Biersack, "Dynamic parallelaccess to replicated content in the internet," IEEE/ACM Transactions on Networking, vol. 10, no. 4, pp. 455–465, 2002.

[11] E. Adar and B. Huberman, "Free riding on gnutella," First Monday 5, October 2000.

[12] Ashwin R. Bharambe, Cormac Herley, and Venkata N. Padmanabhan, "Analyzing and improving BitTorrent performance", Technical Report MSR-TR-2005-03, Microsoft Research, February 2005.

[13] Dongyu Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," ACM *SIGCOMM* 2004.

[14] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. Al Hamra, and L. Garces-Erice, "Dissecting bittorrent: Five months in a torrent's lifetime," *Passive and Active Measurements (PAM)*, 2004.

[15] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, X. Zhang, "Measurements, Analysis, and Modeling of BitTorrent-like Systems", IMC 2005.

[16] M. Jelasity and M. van Steen. "Large-Scale Newscast Computing on the Internet", Technical Report IR-503, Vrije Universiteit, Department of Computer Science, Oct. 2002.