The Effect of Peer Selection and Buffering Strategies on the Performance of Peer-to-Peer File Sharing Systems

Li Zou   Ellen W. Zegura   Mostafa H. Ammar
College of Computing, Georgia Institute of Technology
Atlanta, GA 30332
{zou, ewz, ammar}@cc.gatech.edu

Abstract

Peer-to-peer systems contain a collection of equivalent entities that form an application-layer overlay network. Each entity acts as client, server, and overlay network router. Peer-to-peer systems enable the rapid development of distributed applications; in short order, peer-to-peer file sharing has become one of the most popular Internet applications. There are two primary functions in a peer-to-peer file sharing application: content discovery (determining which peers have the desired content) and content retrieval (transferring content from a peer to the requester). We focus on two aspects of content retrieval that have received limited attention, namely peer selection and request buffering. Using a combination of analytic modelling and simulation, we find that a simple greedy scheme that uses only local information to select a peer has performance comparable to alternative schemes, including those that make more globally-informed decisions. We also find that the optimal request buffering strategy depends on the system load and availability of peer information. For example, when peer information is scarce, download buffering works well; when peer information is plentiful, no buffering is preferable for better throughput. Our results have implications for the design of high-performance, low overhead peer-to-peer file sharing systems.

1 Introduction

In traditional client-server systems, a functional distinction is made between clients (initiating requests) and servers (responding to requests). In contrast, peer-to-peer systems contain a collection of *equivalent* entities that run on end systems and communicate using the underlying network to form an application-layer overlay network. Each entity acts as a client (initiating requests), a server (responding to requests) and an overlay network router (forwarding requests to other peers). Peer-to-peer systems enable distributed, "grass-roots" applications without the deployment of distinguished servers. In short order, peer-to-peer file sharing applications, such as Napster [1] and Gnutella [2], have become extremely popular. In these applications, users provide keywords and the system identifies peers with matching contents.

There are two primary functions in a peer-to-peer file sharing application: *content discovery* and *content retrieval*. Content discovery consists of identifying the peers that have the desired content. Content retrieval consists of selecting a peer and transferring the content to the requesting node[1]. Much research has focused on content discovery [3] [4] [5] [6] [7], with the aim of developing efficient methods to locate peers with the desired content. We focus on two aspects of content retrieval that have received limited attention, namely *peer selection* and *request buffering*. In the area of peer selection, we consider the overall system performance under various schemes for selecting among peers with the desired content. Using a combination of analytic modelling and simulation, we find that a simple local greedy scheme for selecting a peer has performance comparable to alternative schemes, including those that make more globally-informed decisions.

We then turn to two refinements of the basic system. In one refinement, we consider schemes for *buffering requests*. A peer can search for and retrieve multiple files concurrently. Typically peers will limit the number of concurrent download and upload sessions due to resource constraints. In this case, peers might choose to buffer requests to be served when other requests have completed. We develop and evaluate a set of buffering strategies using simulations. We find that the optimal buffering strategy depends on system load and on the desired tradeoff between success rate and response time. In a second refinement, we consider the issue of *par-*

[1] One can envision systems that would identify multiple peers and perform content transfer in parallel. We restrict the focus of this paper to single peer systems, as both the most prevalent current approach and as the foundational starting point for concurrent peer delivery systems.

tial peer information. While systems (such as Napster) that use a centralized directory for peer information can potentially provide information about all peers, a more scalable system will offer a tradeoff between the amount of peer information provided and the overhead for content discovery. We use simulations to study this tradeoff.

Our research is mostly orthogonal to work on content location in peer-to-peer environments. Instead of assuming a specific content location scheme, we abstract away the details of content location and focus on retrieval. In particular, we assume that the search process results in a list of peers that have the desired content. Our work on peer selection is related to prior work in traditional server replication and selection[8][9].

Peer selection differs from server selection in several important ways, however. Peers are more dynamic than replicated servers. Hence, the results returned by the location process are potentially incomplete and less accurate. Peers are potentially more heterogeneous and limited in their processing and access network capacity than replicated servers. Peers are potentially located in more topologically diverse locations in the Internet, hence the difference between good and poor peers may be more extreme than in the server case. We therefore conclude that retrieval from peers warrants systematic study and new solutions.

The remainder of the paper is organized as follows. In Section 2 we describe a limited theoretical model of file sharing systems and propose several peer selection strategies. Section 3 presents a simulation model for a more realistic system and reports results for the peer selection strategies. Section 4 introduces the concepts of request buffering and peer information and investigates their effects on system performance. The paper is concluded in Section 5.

2. Peer Selection Model and Strategies

To gain insights regarding the peer selection process, we develop and study a simplified model of a general peer-to-peer file sharing system. For this model, a simple greedy peer selection scheme yields good system throughput. We then propose several centralized and distributed peer selection heuristics for more general systems.

2.1 A simple model

Usually a user initiates interaction with a file-sharing system by submitting keywords to the system. After the system returns a name list for the files containing the keywords, the user can select the desired file to download. The centralized or distributed peer selection procedure will choose the "best" serving peer for the user. Let $r =< h, f >$ represent the request for peer selection generated from host h for the file f. Considering a snapshot of the global state at the time, a peer-to-peer file sharing network can be represented as a graph $G = (H, U, E_h, E_u)$, where H and U denote hosts and routers, respectively. An edge $e \in E_h$ represents an access link, which has the form $e = (h, u)$, where $h \in H$ and $u \in U$. Each edge $e \in E_u$ represents an inter-router link. We assume one peer per host and symmetric links[2]. We denote the set of files in the system as F. For each file $f \in F$, there is a subset of hosts $Srv(f) \subset H$ that serve this file.

Generally, a file sharing system needs to compute an allocation $A(r) =< h_s, t_s >$ on-line for each request r, according to the global state of the system, specific file searching algorithm and peer selection strategy of the system. $h_s \subseteq Srv(f)$ is one of the hosts that serves file f, t_s is the waiting time before starting the file transfer after the serving peer is assigned. Once assigned, the requesting host can not be switched to another serving host during the file transfer. An optimal peer selection algorithm should compute a set of allocations that yields the best value for a given objective function. In this section, we define the objective function as the system throughput (i.e. number of bytes transferred per second). Alternative objectives might include success rate and file transfer time, etc.

Finding an optimal algorithm for this general problem is difficult because of its complexity. We simplify the problem to focus on the key elements. In particular, we formulate a static assignment problem assuming max-min fair bandwidth sharing with bottlenecks restricted to the access links. Specifically:

- We consider the static form of the problem in which all requests and are known and all allocations are made at once.
- We assume the access links e_h are the only bottleneck links of the network. Hence we can abstract the network to a single router to which all the end hosts are connected and compute bandwidth allocations based solely on the max-min fairness [10].
- We establish as the objective to maximize the throughput of the allocation, ignoring issues of differences in transfer completion times.

After these simplifications, the problem is still interesting. It isolates and emphasizes the peer selection part of the general allocation problem. The solution to this simplified problem could serve as the theoretical foundation of peer selection strategies in real systems. The simplified problem can be restated as follows:

There is a server set S and a client set C. For each client $c \in C$ the algorithm needs to assign a server

[2] Extension to include asymmetric links is straightforward.

$s \in S$ which will serve c's request such that the system throughput is maximal.

It can be proven that this problem is NP-hard by reducing SUBSET-SUM[11] problem to it[3]. However if we allow only a set of discrete bandwidth values and require that larger values can be divided evenly by smaller ones (divisible bandwidths), then a simple greedy algorithm will give the optimal result. An example is to assume all the links have bandwidth $64K$bps, $1M$bps or $10M$bps. Under this configuration, we prove that the greedy algorithm (i.e. in descending bandwidth order, assigning a client to the server with maximum residue bandwidth) will give an optimal result.

Though the greedy algorithm is optimal in the simplified model, it is not at all clear how well it will perform in a more realistic environment. One might conjecture that more complex algorithms, using either local or global information, might outperform the greedy scheme. We turn to this question next.

2.2 Peer selection strategies

The strategies can be classified into two categories; the first category requires global state information of the peer-to-peer overlay network to compute an assignment for a request, while the other only needs the state information related to the request, such as the potential serving peers of the request. We call these two types of algorithms "global" and "user" algorithms, respectively.

2.2.1 User algorithms

The user algorithms run on each peer in a peer-to-peer system. For each request, the peer utilizes the searching mechanism to discover the serving peers and related information, such as the uplink bandwidth and number of current uploading sessions of the peers. For simplicity, we assume a requesting peer can find all its potential serving peers. We investigate the effect of partial information in the next section.

Greedy The algorithm selects the peer that has the maximal value of $\frac{b}{n+1}$ among candidates, where n and b are number of uploading sessions and the uplink bandwidth of a serving peer, respectively. This is a slightly modified version of the greedy algorithm we theoretically studied above.

Fit Let b_0 be the downlink bandwidth of the requester. If all serving peers have the value $\frac{b}{n+1} < b_0$, the algorithm selects the peer that gives the maximal value of $\frac{b}{n+1}$. Otherwise, choose the one that yields minimal positive value $\frac{b}{n+1} - b_0$. The idea of this algorithm is to match the link speed of the requesting peer and the residual bandwidth of serving peers and save fast peers for later requests.

Fastest link This algorithm selects the peer with largest uplink bandwidth regardless of the number of current sessions on that peer. It emulates some file sharing systems where the only information available from the searching processes is the peers' speeds. In this case, it may be reasonable to choose the fastest serving peer.

Random This algorithm randomly chooses one serving node regardless of bandwidth and other state information. This algorithm serves as a baseline for performance evaluation.

2.2.2 Global Algorithms

In reality, the global algorithms could only be run by the central directory server in a Napster-like peer-to-peer network since only the server is in the position to collect global state information. Nonetheless, it is interesting to determine whether algorithms with more global knowledge offer benefit over local algorithms. All global algorithms keep a simplified star-shaped network model as we described in last section, and use max-min fairness to compute the approximate throughput for all transfer sessions. This computation is performed whenever the state of the system changes, i.e. when a transfer finishes or a new connection is placed. The global algorithms select a server on behalf of the requesting peer and only return this serving node to the requester.

Greedy The global greedy algorithm works like the user greedy algorithm except that it computes the bandwidth values using the simplified topology and the global state information of the overlay network.

Fit The global fit algorithm uses the same criteria as the user fit algorithm and the bandwidth value derived from the global state information.

Batch greedy This algorithm accumulates incoming requests for a constant time period before returning results to requesting peers. Then the global greedy algorithm is used to allocate the requests in the descending order of the residual bandwidth of the requesting peers. The motivation is to tradeoff response time for potential better allocations.

Batch fit The algorithm works the same way as the "global batch greedy" algorithm except it uses the criteria in the "user fit" algorithm.

3 Simulation Study of Peer Selection

In this section we design an extensive series of simulations and run them with each of the peer selection strategies proposed in last section. To this end, we first

[3] All proofs are provided in the tech report version of this paper[12].

describe our simulation model and then report simulation results of various peer selection algorithms.

3.1 Simulation model

The simulation model consists of three parts: a network topology model, a file system model and a user request model. The main goal in designing this model is to capture the critical properties of a complex peer-to-peer system. To this end, we choose to use the ns-2 network simulator over a moderate sized network topology and use file distributions derived from a measurement study of real peer-to-peer applications[13]. We simulate a static case where no peers join/leave and no files are added to/deleted from the system.

Network Topology The simulation topology includes 400 peer nodes and 100 router nodes. We use GT-ITM [14] to generate a transit-stub router network. It consists of one transit domain with four transit nodes, an average of three stub domains per transit node, and an average of eight stub nodes per stub domain. The bandwidth of the transit-transit links is 1.6Gbps; the bandwidth of the transit-stub links is one of 155Mbps, 45Mbps and 1.5Mbps with equal probability; the bandwidth of the stub-stub links is either 10Mbps or 1.5Mbps with equal probability. The 400 peer nodes are randomly connected to the 96 stub router nodes.The access links are asymmetrical and their speeds range from 56Kbps modem links to 4.5Mbps T3 links according to the distribution shown in the measurement study by Saroiu et al.[13].

File system properties In the simulation we distribute 800 files over all the peer nodes. The file lengths are sampled from the measurement study by Saroiu et al. [13]. It is long tail distribution ranging from 1KB to 50MB with a peak around 5MB due to the domination of MP3 files on file sharing networks. According to the measurement study [13], the number of files shared by a peer is independent of the bandwidth of the node. We adopt this assumption in the simulation. It is well known from web server studies that the number of servers serving a given file falls into a Zipf distribution [15]. Due to the lack of real measurement data, we use a synthetic Zipf distribution to generate the number of replicas of each file in the system. The distribution is independent of the file size distribution.

User Request Model We assume the probability that a file is requested is proportional to its popularity, thus it is also Zipf distributed. Each peer issues requests independently. The inter-arrival time of requests at a given peer is exponentially distributed with the mean proportional to the anticipated file download time (file length divided by downlink bandwidth of the requesting peer). We use this type of process to objectively compare the performance of different heuristic algorithms since the request arrival process is the same across the simulations.

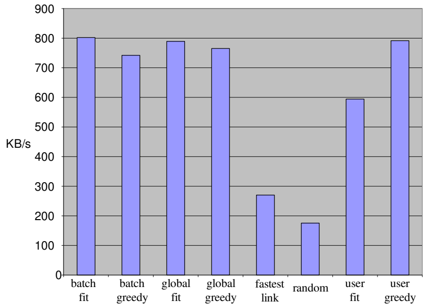

Figure 1. System Throughput

One aspect that we do not simulate in detail is the file searching process since our main focus is the retrieval process. To compensate for the search delay, which has impact on the system throughput, we add a uniform random delay between 2 and 20 seconds to each request and reply.

3.2 Simulation results

We perform a series of simulations using all eight heuristics. The performance data we consider include system throughput and average file turn-around time (the time from issuing a request to completion of the transfer session). All the simulations are run for more than 14,000 seconds of simulation time. To reduce the effect of transient behavior, the file transfers in the first 6,000 seconds are not used. The batching time in the batching algorithms is 10 seconds.

Figure 1 presents the system throughput achieved by using the peer selection algorithms. There are several interesting points shown in this graph. The first is the poor performance of the "random" and "fastest link" selection algorithms. The low throughput in the random case is due to the dominance of modem users among peers. For the case of fastest link, the bad performance is because all requests are directed to the few fast peers without considering their loads. Because all peers can see these few fast peers, they overwhelm these nodes with requests. The second interesting observation is that the simple greedy algorithm performs as well as other more sophisticated heuristics.

Figure 2 illustrates the break-down of average file turn-around time according to the downloading peers' access link bandwidth. Similar to the results of the throughput, the "random" and "fastest link" algorithms incur longer turn-around time whereas the other algorithms have approximately same turn-around time. Another observation from the graph is that the faster the access link, the shorter the turn-around time, which is intuitive. However, the turn-around time is not inversely

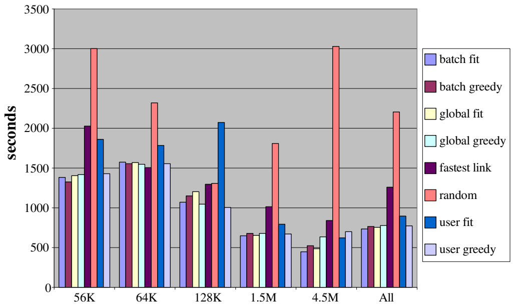

Figure 2. Average turn-around time

proportional to the link speed of the downloading peers. This is partially because of the asymmetry of the links and the fact that the throughput of a session is determined by the link speeds of both downloading and uploading peers.

The simulations in this section show that despite it simplicity, the user greedy algorithm performs as well as other more sophisticated centralized or distributed algorithms. In next section, we mainly consider the case of user greedy algorithm and no longer simulate other seven algorithms.

4 Peer Information and Request Buffering

In simulations performed in the last section, we assume that every peer finds all serving peers. In this section we investigate the effects of partial peer information, as might arise in a real system. Another aspect we do not study in the last section is the request buffering policy. A file transfer session consumes resources, hence, many file sharing applications limit the number of concurrent downloading and uploading sessions (called slots). In this case, request buffering is commonly used to handle the requests when the downloading or uploading slots are full. In this section we investigate the role of buffering on performance.

Buffering and partial information interact in complex ways, hence our simulations consider both together.

4.1 Simulation model

The simulations in this section use the same network topology, file system model and request model as in the last section. However, we slightly modify the original simulation to incorporate the new features of peer information and buffering. We describe the new features of the simulations in the following paragraphs.

Peer information To simulate the case of partial information, a probability value is assigned to every peer to denote the chance that the peer is contacted or considered during the search phase. File searching is performed on the overlay network, which has no obvious relationship with topology of the Internet, thus it should be reasonable to assume that every serving peer has the same chance to be discovered[7]. When a request is issued, a random subset is chosen from all the peer nodes according to the given probability. Then the peers in the subset that actually serve the requested file are returned to the requesting peer. If none of the peers in the subset serves the file, the request is considered failed and dropped.

Buffering strategies We study three buffering strategies: no-buffering, download-buffering and upload-buffering. In all three configurations (including the no-buffering case), every peer has a request queue that also acts as download queue. In the upload-buffering case, every peer is also equipped with an upload queue. The following pseudo code describes how the queues work when a user issues a file request:

Algorithm 1 on-request

```
if requesting peer has no free download slot then
    put the request in the peer's download queue
else
    s ← serving peers returned from file searching process
    if s == φ then
        request failed due to no-server-found
    else case buffering
        no-buffering:
            if no peer in s has a free upload slot then
                request failed due to all-server-busy
            else
                select peer
        download-buffering:
            if no peer in s has a free upload slot then
                put the request back into download-queue
            else
                select peer
        upload-buffering:
            p = selected peer
            if p has a free upload slot then
                connect to p
            else
                put the request to p's upload-queue
```

When a file transfer finishes, the requests at the heads of the queues of involved peers are processed to use the newly released upload and download slots.

Algorithm 2 on-transfer-completion

```
dr ← transfer.destination.download-queue.pop()
resubmit the request dr to the file sharing system
if upload-buffering then
    ur = transfer.source.upload-queue.pop()
    if ur.requester has free download slot then
        connect ur.requester to transfer.source
    else
        push ur back to ur.requester.download-queue
```

Cost function Our earlier study determined that the greedy algorithm has good performance. In this section we generalize the greedy algorithm to add a component that estimates the network traffic generated by a request using the file size and hop count of the peer-to-peer path. Specifically, we define a general cost function as the linear combination of two optimization goals: turn-around time and network traffic. We denote the cost function as: $C = (1 - \alpha)T + \alpha N$, where C is the cost, T and N are estimated turn-around time and network traffic, respectively, and α is a value between 0 and 1 denoting the weight of network traffic in the cost function. The network traffic associated with a file transfer session is $N = S \times H$, where S and H are the size of the requested file and the hop-count from requesting peer to the serving peer, respectively. Estimated turn-around time T includes two parts, waiting time T_W and transfer time T_T. In the estimation, waiting time is only meaningful for the uploading-buffering case when a serving peer has no free upload slots. For the other two buffering strategies, a request that can not be immediately fulfilled is either dropped or queued on the download side again waiting for its next chance to be assigned. When upload queueing occurs, assuming the file transfer is half done, the estimated queueing time is: $T_W = S_W/B + S_T/2/B$ where B is the uplink bandwidth, S_W is the total size of queued files and S_T is the total size of the files being transferred. Assuming uploading sessions at a peer share the bandwidth equally, the estimated transfer time is:

$$T_T = \begin{cases} S/(B/(N+1)) & N < M \\ S/(B/M) & N = M \end{cases}$$

Where N and M are number of current upload sessions and the number of upload slots, respectively.

We simulate the cases where $\alpha = 1$ and $\alpha = 0.0001$. The former corresponds to the case of minimizing the turn-around time, and the latter is a version of hop-count based selection, where among all the peers with same hop-count the greedy algorithm will choose the one with the least estimated turn-around time. We call the former "greedy" algorithm and the latter "hop-count" algorithm in remaining part of this section.

4.2 Simulation results

We performed a series of simulations to study the effect of partial information and request queueing on the system performance, specifically, the request success rate, system throughput, request waiting time, file transfer time and network traffic. The simulations use the same topology, peer, file system configuration as before. In addition, each peer has three upload and download slots. We use both heavy and light load for the simulations. The heavy-loaded case has a request arrival rate five times that of the light-loaded case.

Figure 3 shows the system performance under heavy load. In the graphs, UB, DB and NB denote upload-buffering, download buffering and no-buffering, respectively. The x-axis denotes the percentage of information available to a given peer (i.e. the probability that a search request reaches a random peer). Except for the success rate, all other performance metrics are computed based on successful file transfers.

The *success rate* increases as peers gain more information for all three buffering strategies because the peers have more chance to discover a serving peer. Since the requests in the no-buffering case have only one chance to find a serving peer, their success rate is low compared to the other two cases. In a download-buffering system, a request may success (i.e. find a server with a free slot), fail (i.e. no server has free slots) or be dropped (when no serving peers are discovered in this pass). A failed request will be append to the download queue waiting for another chance to go through the location process, thus another possible chance to be dropped. Hence, as shown in the figure, upload-buffering has better success rate than download-buffering. When the peers have complete information, the upload and download buffering both guarantee every request can be fulfilled because the searching process will always return all the serving peers. Another observation from the graph is that if no buffering is used, even with total information, the success rate is low (about 20%).

The *system throughput* of the upload-buffering strategy is the worst of the three buffering techniques. There are two possible reasons for this result; one is that a request is assigned to a serving peer based on results returned by only one search request thus limiting the choices; the other is the inaccuracy of the turn-around time estimation for the queued files in a peer's upload queue caused by the complexity of the system. In the download-buffering case, a request can be queued and submitted to the system multiple times. As a result, though a request has a higher possibility of being dropped, the successful requests have better chance to find a serving peer. The system also has good performance without buffering since the heavy load will keep the system busy and cancel the effect of the dropped requests.

The throughput of the no-buffering and upload-buffering schemes increases with the availability of peer information. This trend is intuitive because the more information available, the more choices a requesting peer could have. However, the behavior of download-buffering case is not so obvious. It is the result of two competing factors, more choices and increasing waiting time. Having more information provides a peer more candidates to choose from, thus increasing the system throughput. But the benefit rapidly diminishes due to fast-growing waiting time when the amount of the information exceeds certain limit (40% in this case).

All three buffering techniques queue requests at the download side when there is no free download slot.

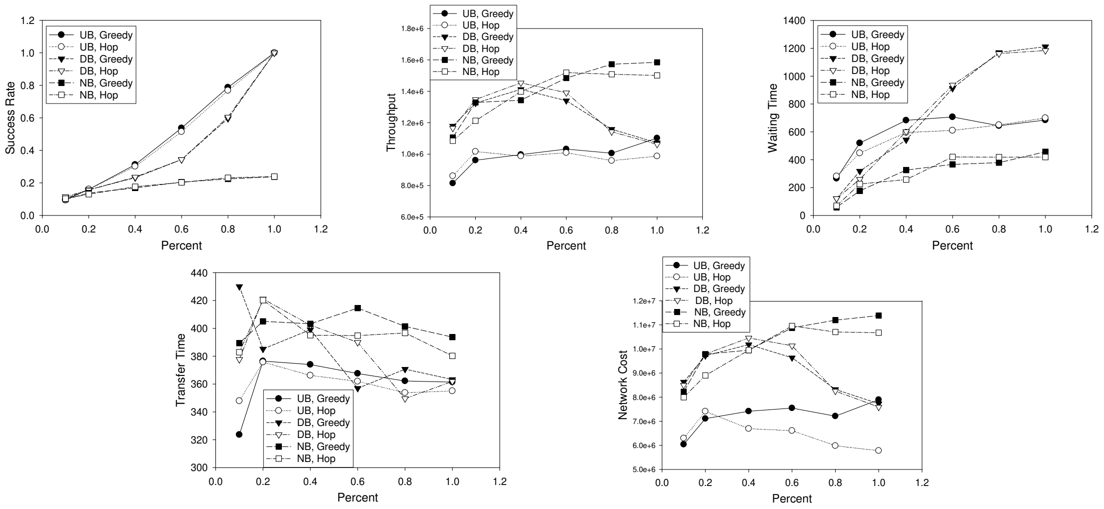

Figure 3. System performance in heavy-loaded system

They behave differently only when there is a free download slot but no serving peer has free upload slot for the requested file. In the download-buffering case, is appended to the download queue of the requesting peer. In no-buffering case, the said request is dropped. In the upload-buffering case, the system selects a serving peer according the cost function and the request is appended to the upload queue of this peer. Thus, no-buffering scheme has the least waiting time since the only queueing involved for a request is waiting for a free download slot at the peer. As the available information increases, the success rate grows and so does the download queue length and waiting time. The download queue in the upload buffering case has the same behavior as the no-buffering case whereas queueing at upload side contributes additional waiting time. The waiting time for the download-buffering strategy increases faster than the upload-buffering case with the amount of information because requests are buffered only at download side causing longer download queues and every request in the download queue will bring extra delay through the searching process.

All the three strategies give similar *file transfer time*. On average, the upload-buffering performs slightly better than the other two methods. The reason could be that the cost function enables the upload-buffering choose to be queued at a fast serving peer instead of immediately download from a slow peer if doing so reduces estimated turn-around time.

The graph of *network cost* shows the same trends as the throughput graph. However, the benefit of the hop-count based algorithm in this category is more evident. Hop-count based method has similar or slightly worse performance in other aspects.

Figure 4 illustrates the performances in a light-loaded file sharing system. The graphs show similar characteristics as in the heavy-loaded system. However, there are several points worth noting. Given same amount of peer information, the light-loaded system has greater success rate since the serving peers are not so busy as in a heavy-loaded system thus the occurrences of dropping requests are less frequent. The throughput of the download-buffering strategy also shows the property that more information does not necessarily bring better throughput. However, the turning point is higher in a light-loaded system than in a heavy-loaded one. As a result, the download-buffering performs better in most cases. Compared to heavy-loaded system, the waiting time in download-buffering strategy increases less rapidly with amount of the peer information as a consequence of slower growth of download queue. As an effect, the upload-buffering becomes the worst one in this category.

Combining all the above observations together, we are able to abstract several performance guidelines for a file sharing system.

- Our hop count-based peer selection algorithm has similar performance to the pure greedy algorithm, but with less network traffic. However, it requires knowledge of hop count, which may be costly to obtain.
- The no-buffering strategy always has least waiting time. It also has good throughput, especially when a large amount of peer information is available. However, its success rate is the lowest of the three buffering techniques.
- The upload-buffering strategy gives worst throughput almost all the time. Under light-load it also results in worst waiting time. Although it has best file transfer time and success rate, its benefit is not enough to offset the bad performance on

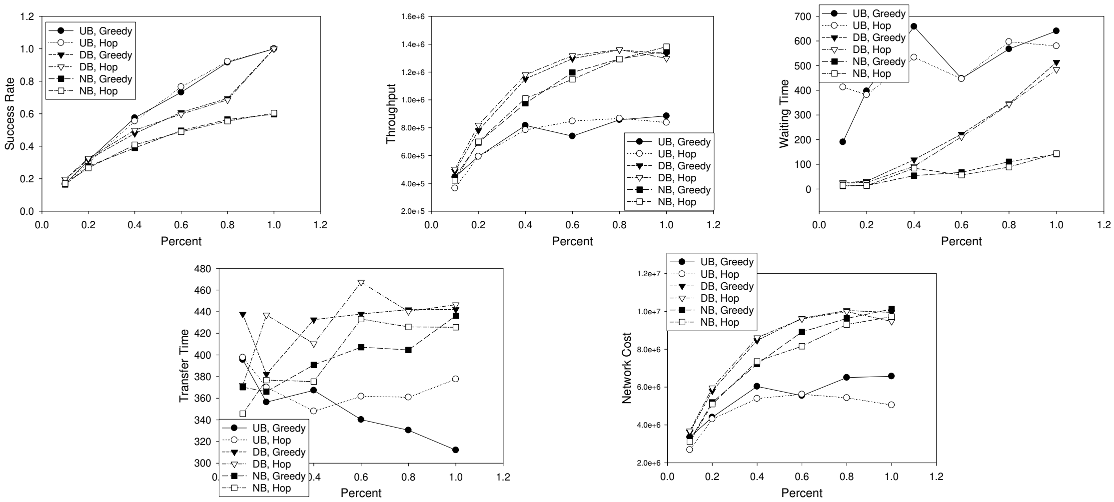

Figure 4. System performance in light-loaded system

throughput and waiting time.

- The download-buffering case leads to best throughput and good waiting time when the peer information is scarce or the system load is light. However, when under heavy load and with large amount of peer information, the performance degenerates rapidly. It also has a good success rate.
- Except for the download-buffering case, more peer information leads to better throughput but also results in longer waiting time. In addition, the rate of increase of throughput tends to slow down as the amount of peer information increases.

When designing a real file sharing system, one could choose a queueing strategy based on the objective of the file sharing system and the amount of peer information that the search process can provide. For example, if the system aims to provide high success rate with small amount of peer information, one could use the upload-buffering strategy. If the system emphasizes throughput and anticipates a heavy load with large amount of peer information, no-buffering strategy is a good choice.

5. Concluding Remarks

In this paper, we have investigated the file transfer aspect of peer-to-peer file sharing systems, and specifically peer selection strategies, queueing strategies and partial peer information. Our work focuses on understanding the interaction of these techniques in order to provide insights for designing file sharing applications. Our work in this area needs to continue with consideration of more elaborate peer selection and request scheduling algorithms. Our simple file location model assumes same peer response probability regardless of the location and uniformly distributed request delay. One could use more detailed model and operations of a specific file sharing protocol. Ultimately, one should consider the design of new file techniques using the results presented in this work.

References

[1] Napster. http://www.napster.com.
[2] Gnutella. http://gnutella.wego.com.
[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the ACM SIGCOMM 2001 Conference*, August 2001.
[4] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," in *Proceedings of the ACM SIGCOMM 2001 Conference*, August 2001.
[5] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Tech. Rep. UCB//CSD-01-1141, U. C. Berkeley, April 2000.
[6] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *Lecture Notes in Computer Science*, vol. 2218, p. 329, 2001.
[7] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proceedings of the 16th ACM International Conference on Supercomputing*, June 2002.
[8] R. L. Carter and M. E. Crovella, "Dynamic server selection using bandwidth probing in wide-area networks," in *Proceedings of the IEEE INFOCOM 1997 Conference*, March 1997.
[9] A. Myers, P. Dinda, and H. Zhang, "Performance characteristics of mirror servers on the Internet," in *Proceedings of the IEEE INFOCOM 1999 Conference*, March 1999.
[10] D. Bertsekas and R. Gallager, *Data Networks*. Prentice Hall, second ed., 1992.
[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. The MIT Press, 1990.
[12] L. Zou, E. Zegura, and M. Ammar, "The effect of peer selection and buffering strategies on the performance of peer-to-peer file sharing systems," tech. rep., Georgia Institute of Technology, 2002.
[13] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," Tech. Rep. UW-CSE-01-06-02, University of Washington, 2002.
[14] K. Calvert, M. Doar, and E. W. Zegura, "Modeling Internet topology," *IEEE Communications Magazine*, June 1997.
[15] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *Proceedings of the IEEE INFOCOM 1999 Conference*, pp. 126–134, 1999.