A File-Centric Model for Peer-to-Peer File Sharing Systems*

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

Abstract

Peer-to-peer systems have quickly become a popular way for file sharing and distribution. In this paper, we focus on the subsystem consisting of peers and their actions relative to a specific file and develop a simple theoretical file-centric model for the subsystem. We begin with a detailed model that tracks the complete system state. To deal with the large system state space, we investigate a decomposed model, which not only greatly reduces the complexity of solving the system, but also provides a flexible framework for modeling multiple classes of peers and new system features. Using the model, we can study performance measures of a system, such as throughput, success probability of a file search, and number of file replicas in the system. Our model can also be used to understand the impact of user behavior and new system features. As examples, we investigate the effect of freeloaders, holding-enabled downloading and decoys in the paper.

1 Introduction

Peer-to-peer (P2P) systems [1][2][3] have become a popular way for file sharing and distribution. In such systems, a file is replicated among participating peers by propagating from serving peers to requesting peers. Compared to the traditional client/server architecture, Peer-to-peer systems are more resilient to network problems and provide higher content availability due to file replication. The peers act as both clients and servers of a file. Typically, a user session begins with a search for a certain file, which returns a list of peers that possess the file. This is followed by a process during which the peer chooses one serving peer from the list and directly downloads the file from it. Once the file is retrieved, the peer can serve it to other peers until it goes offline or removes the file.

*This work is supported by the AFOSR MURI Grant F49620-00-1-0327 and NSF Grant ANI-9973115

The degree of replication of a file affects the system performance related to the file. A higher degree of replication increases both the chance to find the file and the number of serving peers to choose from. While content replication is widely used in content distribution networks (CDN) [4], the replication process in a peer-to-peer file sharing network is inherently different. A CDN usually has direct control of the replication process, e.g., the location of the server and the life time of the replicated content. In contrast, file replication in a file sharing system is a side-effect of the user request and downloading process. The degree of replication and the lifetimes of replicas of a file are governed by the collective user behavior towards the file.

Extensive research has been conducted on the fundamental issues of content replication and server selection in traditional client/server environments [5][6]. Lv et al [7] study the file searching problem using multiple random walk strategy and investigate the optimal file replication in peer-to-peer networks. The server selection problem in peer-to-peer environments and several peer selection heuristics are discussed in [8]. However, relatively little work focuses on understanding the basic replication processes of file sharing systems and their impact on the system performance. It is difficult to study the performance issues and dynamic properties of real deployed file sharing systems due to their distributed nature and lack of centralized management. Clearly, a theoretical model will help to study these problems.

The goal of our work is to develop a simple Markovian model that captures some fundamental properties of file sharing systems. Although it is not a very accurate description of the real system, the Markovian model is theoretically tractable and allows meaningful relative comparisons among possible scenarios and provides significant insight into the performance of a P2P system. We base the model on a detailed analysis of various behaviors of multiple classes of peers toward a certain file. Files are the central entities of a file sharing system and peers' preferences for a file collectively determine the request pattern for the file. Combined with peers' other

Proceedings of the 11th IEEE International Conference on Network Protocols (ICNP'03)
1092-1648/03 $17.00 © 2003 **IEEE**

behavior patterns, such as their willingness to share the file and the percentage of time they stay on line, the peers' preferences for the file also determine the number of replicas of the file in the system and the lifetime of the file at a peer's local storage, which in turn determine the performance measures related to the file. By solving the model, we can investigate the performance of the system, such as the system throughput, the probability of a successful search and the degree of replication. It can also be used to understand the effect of multiple classes of peers on the system.

The model is also versatile. It not only can be used to derive performance measures from existing peer-to-peer architectures, but also can be used to analyze and design new file sharing protocols. As examples, we investigate two phenomena of recent peer-to-peer file sharing systems, *holding-enabled download* and *decoys*. Holding-enabled download is a countermeasure to freeloaders in which peers are forced to keep the file online for a small fraction of time to avoid its immediate delete by freeloaders. A decoy is a fake file with the same name as the real one. It can be injected into the file sharing system deliberately to confuse the users who search for the file.

The rest of the paper is organized as follows. In section 2, we discuss the background of peer-to-peer networks and related works. Section 3 presents our file-centric model. In section 4, we develop models for a variety of peers according to their behaviors, which are used in the experiments later. Section 5 performs a series of experiments using the file-centric model and discusses the results. The paper is concluded in section 6.

2 Background

Using the terminology introduced in [7], peer-to-peer networks can be categorized into three types: *centralized*, *decentralized but structured* and *decentralized and unstructured*. The first peer-to-peer system, Napster [1], used the centralized architecture. In centralized systems, central index servers are responsible for replying search requests from all peers. Most recently introduced peer-to-peer architectures, such as Chord [9], CAN [10] and Tapestry [11] fall into the second category. However, the only widely deployed file sharing system which is loosely structured is Freenet [12]. In this category, the interior structure of the system quickly routes the search request to its destination. As a result, the file searching time can be ignored when compared with the file downloading time. The vast majority of deployed file sharing systems, including Gnutella [2] and Kazaa [3], adopt various flavors of decentralized and unstructured architecture. Generally, in these systems, a request is flooded to neighbors. A time-to-live (TTL) scope is used to limit the request propagation. In this paper, we only study the decentralized and unstructured systems, though the model can accommodate the other two categories with some extensions.

Our work is most closely related to the work by Ge et al [13]. They model a peer-to-peer file sharing system as a multiple-class closed queueing network. The queueing network consists of four parts: common services, file download services, off-line and think time. The common services is a single server queue that models the file search process, whose service rate is a function of the numbers of on-line peers of all classes. For each file, the authors used a single server queue for the download service. The service rates of the download queues are determined by the number of on-line peers of all classes as well as the replication ranks of files. The off-line and think queues are both modeled as infinite server queues whose service rate depend only on the class of the peer. After going on-line, a peer goes in the common service queue waiting for the end of the search phase. After a successful search, the peer goes through the download queue and then returns to either the think queue or the off-line queue.

The model has some limitations. A peer can only go to the off-line state immediately after a successful file download, which is not true in a file sharing system. The model also assumes a static scenario where the number of file replicas is constant over time, thus the service rates of the download queues depends only on the number of on-line peers. However, in a real system, the number of file replicas increases as a peer acquires the file either from a successful downloading in the system or from the injection of the file into the system. The number of replicas decreases as a peer with a replica goes offline or by user deletion. The number of replicas of a certain file should be determined by the peer behavior relative to the file.

Our file-centric model focuses on the subsystem consisting of peers and their actions relative to a *specific file*. The model is based on the detailed analysis of a single peer and derives system performance measurements from the aggregation of individual peer states. It not only addresses the aforementioned limitations straightforwardly but also provides a tool to describe complex interactions between peers and files.

3 File-centric model of peer-to-peer file sharing systems

A file sharing system can be seen as a content delivery system where consumers of a file also act as providers of the file. The capacity of the system to serve the file depends on the number of on-line peers that possess the file and are willing to share it with other peers.

It also depends on the ability of the system to route a searching request of the file to a serving peer. For a specific file, its consumers and providers form a subsystem that collectively decides the availability of the file. The peers that are outside of the system make up the "background" for the subsystem, which indirectly affects the performance of the subsystem by participating in the distributed search process. In order to understand the fundamental characteristics of a general file sharing system, we target the properties of these subsystems associated with unique files.

In this section we develop a file-centric model that deals with one particular file in the system. The model is based on the analysis of peers' life cycle and their behavior relative to the file. We first study the interactions between a peer and the file and identify the possible states of the peer. We then propose a detailed Markovian model that tracks the state of every peer in the subsystem corresponding to the file. To deal with the large state space, we investigate an approximation by using a decomposed model.

In the remainder of the paper, we consider the interaction relative to one file in the system, thus the words "system" and "subsystem" will be used interchangeably. We assume that there are n peers that are relevant to the specific file and $N-n$ background peers. Thus there are N peers in the system that potentially participate in the search process.

3.1 Peer state space

Peers have different behaviors depending on users' choices of how to utilize the file sharing system. For instance, after a successful request and download, a well-behaved peer would keep the file for a while to share with others. A freeloader, on the other hand, would immediately put the downloaded file off-line or deny its existence to the incoming requests. Peers also have different preferences for the given file depending on the content of the file. A certain class of peers may have a high probability to request the file whereas some peers may never search for it. To study a file sharing system consisting of various classes of peers, we put these different classes into a common framework by abstracting their fundamental properties. We identify the possible states of the model based on the observations of interactions between a peer and the given file. The states are defined to capture the complete life cycle of a peer and the file in the system. The behavior of the peer is modeled using a Markovian state transition diagram. At any state in the model, the peer waits for a period of time before changing to the next state according to the transition diagram. The parameters of the waiting time distribution reflect the behavior characteristics of the peer. Different classes of peers have their own sets of parameters and/or their own transition diagrams.

A peer may request the same file multiple times. In order to capture the properties of the file propagation process, we specifically distinguish the states before a peer gets the file for the first time from other states. These states are transient and do not affect the stationary distribution of the system.

We first present a representative example that illustrates a state transition diagram of a typical peer, as shown in Figure 1. Other types of peers can be modeled in a similar fashion. Generally, the peer can be either on-line and participating in the file sharing system or off-line. Also, it either has the specific file or not. The combination of these two factors produces four states. An on-line peer without the file will generate a search request for the file at a certain rate depending on its preference for the file. The search takes some time and may succeed or fail. After a successful search, the peer will initiate file downloading directly from a serving peer. We assume for this peer, the file could also be injected from outside of the system at certain rate. After keeping the file for a while, the file is deleted from the peer regardless of whether the peer is off-line or on-line. Like in a real system, we assume the peer can go off-line at any time, including during the searching or downloading processes.

The transitions reflecting these properties are depicted by the state transition diagram shown in Figure1. The states that are denoted with apostrophes represent the transient states that the peer has never had the file before. Table 1 and Table 2 explain each state and transition rate. Most transition rates in the same row should have the same value except for the search request rates. We can assume that a peer has a higher initial request rate when it requests the file for the first time. By studying the effect of the initial request rates, we could investigate the process of file propagation in the system. Generally, we assume the rates are time-independent but they could depend on the states of other peers. Specifically, in the diagram shown in Figure 1, the rates that depend on other peers' states are the successful/failed search rate and the downloading rate. The probability of finding the file, P_{find}, depends on both the number of replicas in the system and the number of peers the request visits. Assuming uniform distribution of the file over the peers, $P_{find} = 1 - (1 - m/N)^k$, where m is the number of on-line peers that have the file (i.e. in State ON/HAS), N is the total number of peers in the system and k is the number of peers visited by the request. The downloading rate λ_d not only depends on the characteristic of the peer and the file, e.g, the peer's link bandwidth and the file length, but also could depend on the number of potential serving peers. The other rate

parameters are mostly a peer's or a file's intrinsic properties; the rates of going online and offline is determined by the user's online pattern, the rate of request is determined by the preference of the peer to the file, the inject and delete rates are related to both the peer's behavior and the file's nature.

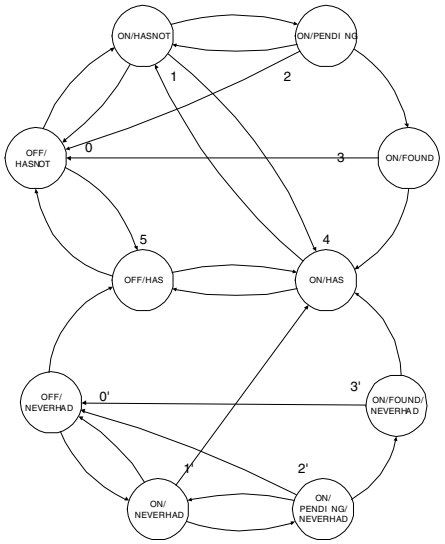

Figure 1. Single Peer State Transition Diagram

By introducing new states into or removing existing states from the diagram in Figure 1 and/or changing the rate parameters we can model a variety of peer classes. For instance, some peers act like traditional file servers, that is, they never delete files and never request and download from other peers. In this case, we can remove the states related to the searching and downloading. Another instance is the freeloader. We can either assign zero file inject rate and very high delete rate to the corresponding transitions or remove the states that the peer has the file. A more complicated example is what we called "decoys". The concept is to inject fake files into a file sharing system to confuse and discourage downloaders. To investigate the effect of decoys, the model needs to be changed to represent the existence of false files in the system. Specifically, each of the "ON/HAS", "OFF/HAS" and "ON/FOUND" states needs to be split in two states, one for the real file and the other for the fake file. Accordingly, the search and download rates also need to be changed. We will discuss various classes of peers in Section 4.

3.2 Detailed model

One modeling approach is to treat the whole subsystem as a single entity for content delivery, where the peers contribute to service capacity and request resolving. In contrast, our approach is to model each peer in detail and deduce the global properties from the individual states of the peers. This approach more closely reflects how a peer-to-peer system works and provides some insights that other models can not achieve.

In the detailed model, we define a state of a file sharing system as a collection of individual states of peers, which are discussed above. Formally, we define $\gamma = \{s_1, s_2, \ldots, s_n\}$, where γ is a system state, n is the number of peers in the system, $s_i \in S_i$ denotes the state of peer i where S_i is the state space of peer i, as described in Section 3.1. Thus, the state space of the system, $\Gamma = S_1 \times S_2 \times \ldots \times S_n$ is the Descartes product of individual state spaces. Since a system state gives a full description of the system relative to one file, global measures, such as the number of file replicas and the number of on-line peers, can be derived. Hence, the peer-state-dependent transition rates can be determined.

We use the detailed model in two different ways. If we intend to study the transient behavior of the system, we could use the full individual peer transition diagram shown in Figure 1. Otherwise, if one is only interested in the stationary distribution of the system, the upper part of the diagram, which represents the recurrent states, can be used.

By constructing the system transition matrix and solving the corresponding linear equations, we can get the stationary distribution of the system and deduce the desired performance measures from it. The transition matrix has a dimension of $|\Gamma| = |S_1| * |S_2| * \ldots * |S_n|$. In theory, we can compute the transition matrix M for this detailed model and find the stationary distribution vector x by solving a set of linear equations. Transition matrix M is extremely sparse. However, for a 6-state transition diagram, even using specialized numerical sparse matrix inversion algorithms, the maximum number of peers we could handle was 7. Thus we need a viable model to deal with large file sharing systems. However, since the detailed model is the more accurate description of the system, we use the stationary distributions of small detailed models as the basis to evaluate the accuracy of other models.

3.3 Decomposed model

The detailed model is most expressive since it keeps the state of the whole system but it is computationally expensive to solve. However, obtaining the state of every single peer usually is not necessary. The state-dependent rates actually depend only on the number of peers at the states. To find the performance measures of the system, one also only needs the aggregated state information, such as the number of peers that occupy a given state.

The peers in a file sharing system usually can be categorized into classes. The peers in the same class can

0, 0'	OFF/HASNOT, OFF/NEVERHAD: The peer is off-line and does not have the file
1, 1'	ON/HASNOT, ON/NEVERHAD: The peer is on-line and does not have the file
2, 2'	ON/PENDING, ON/PENDING/NEVERHAD: The peer is on-line and waiting for the search result
3, 3'	ON/FOUND, ON/FOUND/NEVERHAD: The peer found the file and is downloading it
4	ON/HAS: The peer is on-line and has the file
5	OFF/HAS: The peer is off-line and has the file

Table 1. Single Peer States

$\lambda_{01}, \lambda_{54}, \lambda_{0'1'} = \lambda_{on}$	The rate of going from off-line to on-line
$\lambda_{10}, \lambda_{20}, \lambda_{30}, \lambda_{45}, \lambda_{1'0'}, \lambda_{2'0'}, \lambda_{3'0'} = \lambda_{off}$	The rate of going from on-line to off-line
$\lambda_{05}, \lambda_{14}, \lambda_{0'5}, \lambda_{1'4} = \lambda_{inj}$	The rate of injecting the file to the peer
$\lambda_{50}, \lambda_{41} = \lambda_{del}$	The rate of deleting the file from the peer
$\lambda_{12}, \lambda_{1'2'} = \lambda_{req}$	The rate of issuing search requests for the file
$\lambda_{20}, \lambda_{2'0} = \lambda_{abort1}$	The rate of aborting a search request
$\lambda_{21}, \lambda_{2'1'} = \lambda_s(1 - P_{find})$	The rate of failed search
$\lambda_{23}, \lambda_{2'3'} = \lambda_s P_{find}$	The rate of successful search
$\lambda_{30}, \lambda_{3'0} = \lambda_{abort2}$	The rate of aborting a downloading
$\lambda_{34}, \lambda_{3'4} = \lambda_d$	The rate of file downloading

Table 2. Single Peer Transition Rates

be thought of as identical. In this case, each class corresponds to a different transition diagram. Due to the symmetry of identical peers, for any system state, swapping any two peers in the same class will not change the stationary state probability. That is, $P\{s_1 = a_1, \ldots, s_i = a_i, \ldots, s_j = a_j, \ldots, s_n = a_n\} = P\{s_1 = a_1, \ldots, s_i = a_j, \ldots, s_j = a_i, \ldots, s_n = a_n\}$ if peer i and j belong to the same class. Hence, by aggregating these states and considering the marginal distribution of the peers at each state, we define new system states as $\gamma = \{n(i,j) | i \in [1,k], j \in s_i\}$, where k is the number of classes, s_i denotes the state space of peer i, $n(i,j)$ denotes the number of peers at state j of class i. $n(i,j)$ satisfies $\sum_{j \in s_i} n(i,j) = n_i$ and $\sum_{i=1}^{k} n_i = n$ where n_i is the number of peers of class i. The new transition matrix can be derived from the original one. The dimension of the new matrix is polynomial in n instead of exponential as in the detailed model, thus greatly reducing the computation complexity of solving the linear equations.

To reduce the complexity further, we investigate the property of a class of peers. Since all the peers in the same class are identical and each peer changes states independently. Hence, we can obtain the average number of peers at a state from the stationary state distribution of a single peer in the class. If all the transition rates were state-independent, then this method will yield accurate results. In our model some transition rates depend on aggregated state and do not directly depend on the system state. However, because the model targets the file sharing systems that contain a large number of peers, we anticipate that the aggregated state should have small variance. Hence, we can use this method as an approximate algorithm. The technical report version of this paper [14] provides some intuition into why we believe this approximation is accurate. In Section 5, we use simulations to verify this.

For a general case that involves multiple peer classes, we obtain numerical results using the decomposed model using the following iterative procedure:

1. For each class state transition diagram, set up flow balance equations with state-dependent rates
2. For each set of equations, assume reasonable initial rates for the state dependent rates.
3. For each set of equations iterate on the state-dependent rates until the desired convergence precision is obtained. This step deals with the rates that depend on the states of the same class.
4. Repeat the last step until all state-dependent rates achieve the desired convergence precision, this step deals with the rates that depend on the states of the other classes.

The following algorithm describes the procedure more formally, where d is the number of classes:

Algorithm 1

1 **for** each class C_i, $i \in [1,d]$, assume a set of state dependent rates r_i
2 Repeat until $r_1, \ldots, r_d$ all converge
3 **for** each class C_i, $i \in [1,d]$, compute state transition matrix $M_i(r_1, \ldots, r_d)$
4 **for** each class C_i, $i \in [1,d]$
5 Repeat until r_i converges
6 Solve linear equations $P_i = P_i M_i$ to get stationary distributions P_i for class C_i
7 Compute class C_i's state dependent rate $r_i(P_1, \ldots, P_i, \ldots, P_d)$ using the new P_i

4 Model Parameters

In this section, we discuss several peer classes and their corresponding parameters. These classes can be used together to describe a complex subsystem associated with a given file. The peer classes we discuss here only serve as examples and are used in the evaluation

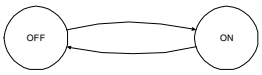

Figure 2. Transition Diagram of Contributor and Decoy

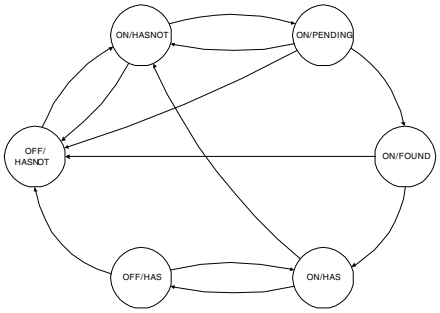

Figure 3. Transition Diagram of Normal Peers in a System without Decoy

part of the paper. When needed, other peer classes can be modeled in a similar manner

We model four types of users and two types of downloading applications. The four types of users are *contributors* who inject the file into the system and never delete it, *normal* class users who search and download the file and share it with others and delete it after some time, *freeloaders* who only search and download the file and immediately put the file off-line after downloading, *decoy* class peers who inject fake file into the system and never delete it. In addition to the normal file sharing application, we also study a type of application that keeps the file for a short amount of time after a successful downloading and before yielding control to the user. We call this type of application as *holding-enabled* downloading. During this short holding time, the file can be served to others. Since usually the majority of peers are freeloaders, using holding-enabled downloading could increase the availability of the file. We do not show the transient states in the following transition diagrams for simplicity. The transient states can be added to the diagrams easily in the same manner shown in Figure 1.

Figure 2 shows the transition diagram for both contributor and decoy classes, which inject authentic or fake files into the system, respectively. They never remove the file from system but could be taken off-line. They are the sole sources for the files and decoys in the system. Figure 3 presents the transition diagram for normal peers if there are no decoys in the system. Figure 4 shows the case with decoys in the system. In this case, we need to keep the state of whether or not the file is authentic.

Figure 5 shows the transition diagram of freeloaders. Freeloaders immediately put the files off-line after a successful downloading. In the diagram the rate from ON/FOUND state to ON/HASNOT state reflects the downloading rate. The version in the presence of decoys can be constructed similarly.

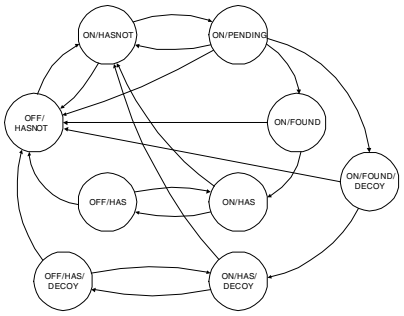

Figure 4. Transition Diagram of Normal Peers in a System with Decoy

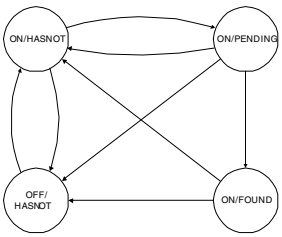

Figure 5. Transition Diagram of Freeloaders in a System without Decoy

Figure 6 illustrates the transition diagram for a normal peer that is running a holding-enabled download in the absence of decoys in the system. The case for freeloaders and/or with decoys is similar to Figure 6.

Hence, in a system with all four types of users and two types of applications, there would be six different classes in the model. The transition rates in these diagrams are similar to those described in Table 2. If there

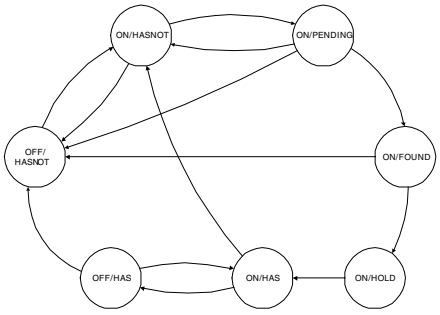

Figure 6. Transition Diagram of Normal Peers Running Holding-Enabled Applications in a System without Decoy

are l copies of the file and l' copies of the decoy in a file sharing system with N peers and a search request visits k peers, then the probability that the search finds a real copy is $(1-(1-(l+l')/N)^k)l/(l+l')$ and the probability of finding a decoy copy is $(1-(1-(l+l')/N)^k)l'/(l+l')$.

To assume a reasonable downloading rate, for simplicity, we consider an optimal system that is capable of matching up downloading peers and serving/holding peers perfectly. In other words, the system will assign an incoming downloading peer to a serving/holding peer when it is available, otherwise the downloading peer will share a serving/holding peer with others. We denote the number of downloading peers as k, the number of online peers that have the file as l and all peers have the same upload/download capacity at rate λ_c. In a system that does not utilize holding-enabled download, if $k < l$, each downloading peer can be assigned to a serving peer, thus has a downloading rate $\lambda_d = \lambda_c$. If $k \geq l$, each of the l serving peers will upload at rate λ_c, thus on average, the downloading rate is $\lambda_d = l\lambda_c/k$. That is:

$$\lambda_d = \begin{cases} \lambda_c & l > k \\ l\lambda_c/k & otherwise \end{cases}$$

In the presence of holding peers, we assume that the number of holding peers is m and each one holds the file for α fraction of the downloading time. For a downloading peer that is assigned to a holding peer, if it has not retrieved the full file before the holding period expires, the system will reassign it to another serving/holding peer for the reminder of the file. Thus, it needs $1/\alpha$ holding peers to finish the downloading fully. Hence, we can treat the m holding peers as αm serving peers. In this case:

$$\lambda_d = \begin{cases} \lambda_c & l + \alpha m > k \\ (l+\alpha m)\lambda_c/k & otherwise \end{cases}$$

5 Numerical results

In this section, we present the results of our models. Depending on the different aspects on which we focus, we use various combinations of the peer classes discussed in the last section. First we investigate the transient behavior of the system via simulations of the model. Next we evaluate the effectiveness of the decomposed model by comparing its results to that of the detailed model and simulation. We then use the decomposed model to study some performance issues of file sharing systems.

5.1 Transient Behavior of the System

To study the propagation process of a file, we simulate a subsystem with 2001 peers, of which one peer is a contributor, 400 are normal peers and 1600 are freeloaders. We assume that there are 10^6 "background" peers which are irrelevant to the considered file except for participating in the searching process. All the peers are running the non-holding version of the application. The parameters we use are: $\lambda_{on} = 1.0$, $\lambda_{off} = 5.0$, $\lambda_{del} = 0.1$, $\lambda_{abort1} = 2.0$, $\lambda_{abort2} = 0.2$. The download rate function is the same as discussed in the last section and the constant $c = 4.0$. We assume that the system uses a flooding search architecture and each request visits $2,000$ on-line peers. For the recurrent states, we use $\lambda_{req} = 10.0$. We study the behavior of the system using different transient state request rates $\lambda_{req'}$.

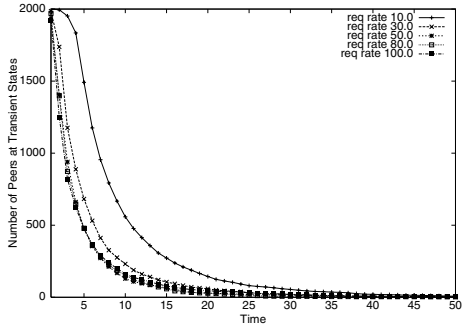

Figure 7. Number of Transient Peers in the System

Figures 7 and 8 show the number of peers that are in transient states and the average number of replicas in the system. The simulation starts with all the peers at transient state OFF/NEVERHAD. As the simulation progresses, a peer leaves the transient states when it gets the file for the first time. Hence the number of peers in the transient states steadily decreases. We can see that higher transient request rates make the number of transient peers drop faster, however, there exists an upper bound on this speed regardless of the request rate, which is determined by other transition rates in the system. In Figure 8, the number of file copies increases very fast in the beginning because of the higher initial request rates. After most peers have left the transient states, the number of copies decreases to the level determined by the stationary process. The horizontal line in Figure 8 denotes the average number of copies in the system derived by the decomposed model. It agrees with the simulation results very well.

5.2 Evaluation of the Decomposed Model

To evaluate the decomposed model, we solve the detailed models for small file sharing systems and compare the results with that of the decomposed model. In this experiment, we assume all the peers in the system are in the same class with the transition diagram shown in the recurrent part of Figure 1. For simplicity, we also assume that the probability of a successful search is pro-

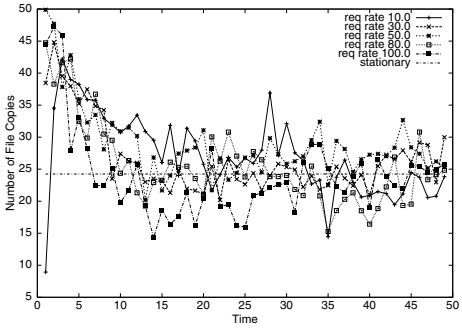

Figure 8. Number of Copies in the System

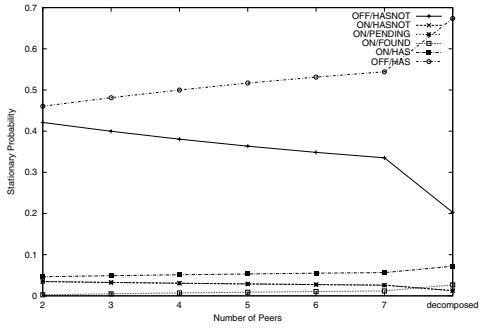

Figure 9. Result of Detailed Model

portional to the ratio of the peers that already have the file among the on-line peers. Due to computational complexity, we can only solve a detailed model with less than eight peers in the system. Figure 9 illustrates the stationary probabilities of each state of the systems with two to seven peers. It also shows the results of the corresponding decomposed model. In spite of the small sample size, we can still see that the results from the decomposed model agree with the general trend of the curves. It supports our conjecture that the decomposed model asymptoticly converges to a system with an infinite number of peers.

5.3 Application of the File-Centric Model

The versatility of the decomposed model provides us a powerful tool to analyze file sharing systems with complex peer-peer and peer-file interactions. From the stationary distribution of the decomposed model, we can derive various performance measures such as throughput and success rate of a file search. The model also enables us to investigate the effect of various peer behaviors and system parameters on the performance. In this part, we study a file sharing system that consists of contributors, decoys, normal peers and freeloaders. Each normal peer or freeloader could run either the unmodified application or the holding-enabled version.

Unless specified, we use the following class parameters throughout this part of the experiments. There are

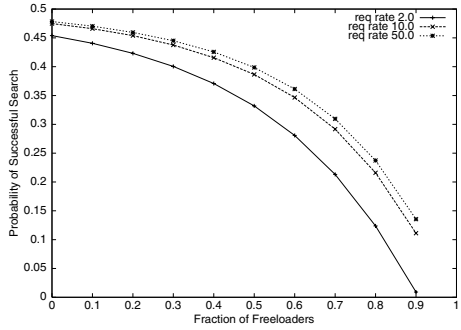

Figure 10. Probability of finding the file with varying fraction of freeloaders

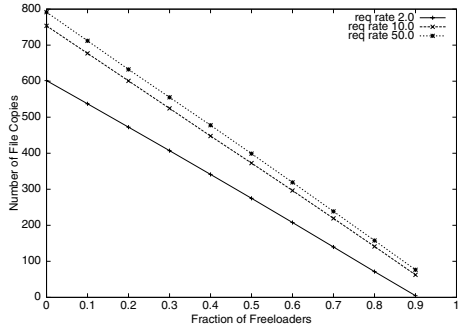

Figure 11. Average number of copies of the file in the system with varying fraction of freeloaders

$10,002$ relevant peers to the considered file in the system, of which, one is a contributor, one is a decoy, 20 percent of the remaining $10,000$ peers are normal peers and 80 percent are freeloaders. Of the $10,000$ peers that actively request the file, 70 percent are using unmodified application and the other 30 percent are using the holding-enable version. The choice of the application is independent of whether the peer is a normal peer or a freeloader. The following are the default system parameters we used. $\lambda_{on} = 1.0$, $\lambda_{off} = 5.0$, $\lambda_{del} = 0.1$, $\lambda_{abort1} = 2.0$, $\lambda_{abort2} = 0.2$, $\lambda_{req} = 10.0$ and $\alpha = 0.3$. We use the same downloading rate function as in the last section and with constant $\lambda_c = 4.0$. A flooding search visits 2000 on-line peers for each file request.

We first study the impact of the freeloaders on the system performance. For this experiment, the percentage of freeloaders varies from 0 to 90. We do not count the throughput and success rate of downloading decoys into the system performance. Figures 10, 11 and 12 show the probability of finding the file, the number of copies of the file and the throughput of the system, respectively for different request rates. Here, the throughput we consider is the number of successfully transfered files in unit time.

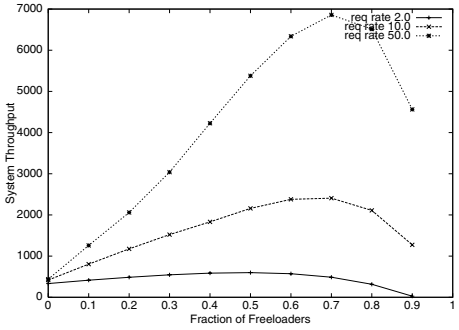

Figure 12. System throughput with varying fraction of freeloaders

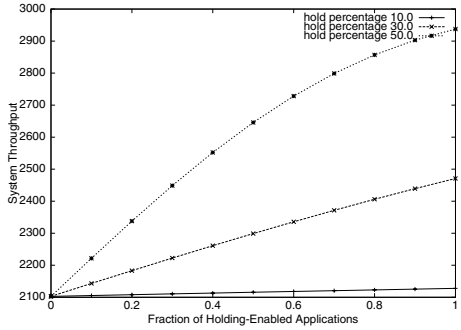

Figure 14. System throughput with varying fraction of holding-enabled download

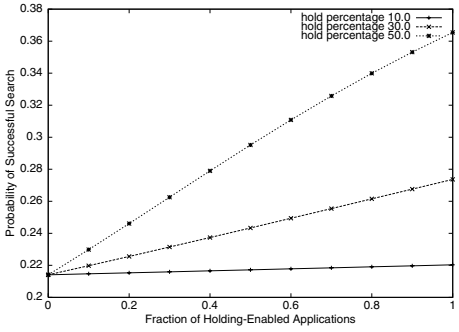

Figure 13. Probability of finding the file with varying fraction of holding-enabled download

Figure 11 shows that the number of replicas in the system decreases linearly with the percentage increase of freeloaders. As a result, the probability of finding the file also decreases. However, Figure 12 shows that the system throughput initially increases with the percentage of the freeloaders and then drops rapidly above a certain threshold. The increasing part of the curve is due to the lower turnaround time of the freeloaders. Since freeloaders do not spend time to share the file with other peers, they can issue more file requests than normal peers in the same amount of time, thus increasing the system load. But beyond a certain threshold, as the number of peers that have the file decreases, all the serving peers will be saturated by incoming requests, thus limiting the system throughput.

The result about freeloaders agrees with that by Ge, et al [13] in that as long as the system has spare serving capacities, more freeloaders will increase the system throughput but lower the request success rate.

Figures 13 and 14 show the effect of the holding-enabled download. Both search success rate and system throughput improve linearly with the fraction of peers that use the holding-enabled version. The longer the holding time, the better the effect. If all the peers use the holding-enabled download and hold the file for half the downloading time, the probability of finding the file almost doubles and the system throughput improves by 50 percent. Although usually it is impractical to ask all the peers to change to a new version of the application at the same time, the linear growth of the performance allows new versions to be deployed incrementally to achieve increasingly better performance.

Decoys have been used to fight piracy on file sharing systems by confusing and frustrating downloaders. The decoys are injected into file sharing systems and are expected to propagate in the same manner as normal files. Usually a peer can not distinguish between a real file and a decoy until the downloading is finished and the file is opened. To investigate the effectiveness of the decoy, we focus on the users' behavior on decoys. In our experiment setup, if users treat the decoy in the same way as a normal file, due to symmetry, the system will have the same number of copies of decoys and files. If users delete decoys at a higher rate than files, we can anticipate that the system will have more copies of the file than that of the decoy.

Figure 15 presents the effect of a higher decoy deletion rate. The x-axis denotes the ratio of deletion rate of decoys to that of files. We assume that only 50 percent of all the peers delete decoys at this higher speed and the others treat decoys and files in the same way. The figures show that the use of decoys is less effective as the users delete decoys faster. Hence, in this case, using decoys is not an effective way to hinder the spread of the file.

Lastly, we study the effect of the search scope. A larger search scope leads to a higher search success rate, which in turn leads to a larger number of copies. However, a larger search scope will also result in a longer search time, which reduces the load on the system. In a TTL-scoped flooding search system, the number of searched peers increases exponentially with TTL and the search time increases linearly with TTL. Hence, we

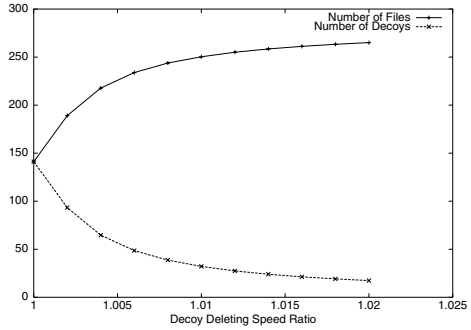

Figure 15. Number of file and decoy copies with varying decoy deletion speed

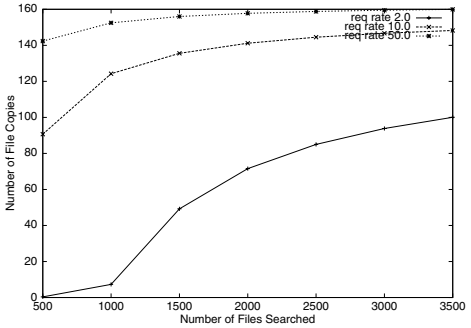

Figure 16. Number of file copies with varying search scope

assume the search time increases logarithmically with TTL. Figures 17 and 16 show the probability of a successful search and the number of copies in the system, respectively.

The figures show that in this setup, the benefit brought by the increasing success rate surpasses the negative effect of the lowered system load. As a result, both the success probability and number of copies increase.

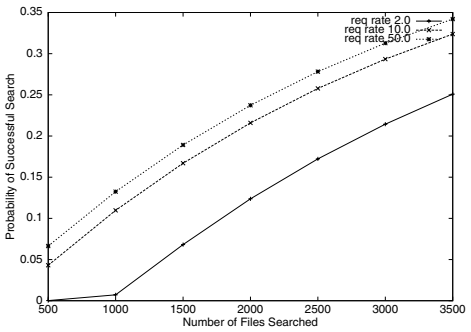

Figure 17. Probability of a successful search with varying search scope

6 Concluding remarks

The work presented in this paper aims to understand the basic performance issues of file sharing systems. To this end, we have presented a theoretical file-centric model that considers a subsystem consisting of peers and their actions relative to a specific file. The decomposed model provides a general, computationally effective and versatile framework for describing system features and peer characteristics. The model is also extensible thus enabling us not only to study the performance measures of existing systems, but also to investigate the effect of emerging features in the system. Using the model, we investigate the file propagation process in a file sharing system, the impact of freeloaders, the effectiveness of using decoys to fight piracy and the performance enhancement by introducing holding-enabled download into the system. Our results highlight several interesting points. The results regarding freeloaders echo a previous study [13]. They also show that using decoys might not be an effective way to fight piracy and incrementally adopting holding-enabled download can improve the system performance. Our future work in this direction will include studying the effectiveness of the Markovian model by simulating low level transactions of a real peer-to-peer system. We also want to investigate other correlation among files and adding peer selection into the model using Markov decision processes.

References

[1] "Napster." http://www.napster.com.
[2] "Gnutella." http://gnutella.wego.com.
[3] "Kazza." http://www.kazaa.com.
[4] S. Gadde, J. S. Chase, and M. Rabinovich, "Web caching and content distribution: a view from the interior," *Computer Communications*, vol. 24, no. 2, pp. 222–231, 2001.
[5] Z. Fei, B. Bhattacharjee, E. Zegura, and M. Ammar, "A novel server selection technique for improving the response time of a replicated service," *Proceedings of IEEE Infocom 1998*, March 1998.
[6] E. Zegura, M. Ammar, Z. Fei, and B. Bhattarcharjee, "Application-layer anycasting: a server selection architecture and use in a replicated web service," *IEEE/ACM Transactions on Networking*, pp. 455–466, August 2000.
[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] 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," in *Proceedings of MASCOTS'02*, 2002.
[9] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *Proceedings of SIGCOMM'01*, August 2001.
[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," *Proceedings of SIGCOMM'01*, August 2001.
[11] 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.
[12] "Freenet." http://freenet.sourceforge.net.
[13] Z. Ge, D. R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley, "Modeling peer-peer file sharing systems," in *Proceedings of IEEE INFOCOM*, (San Francisco, CA), March 2003.
[14] L. Zou and M. H. Ammar, "A file-centric model for peer-to-peer file sharing systems," tech. rep., Georgia Institute of Technology, 2003.