Optimizing Random Walk Search Algorithms in P2P Networks[*]

Nabhendra Bisnik
Rensselaer Polytechnic Institute
Troy, New York
bisnin@rpi.edu

Alhussein A. Abouzeid
Rensselaer Polytechnic Institute
Troy, New York
abouzeid@ecse.rpi.edu

Abstract

In this paper we develop a model for random walk-based search mechanisms in unstructured P2P networks. This model is used to obtain analytical expressions for the performance metrics of random walk search in terms of the popularity of the resource being searched for and the random walk parameters. We propose an equation-based adaptive search mechanism that uses an estimate of the popularity of a resource in order to choose the parameters of random walk such that a targeted performance level is achieved by the search. We also propose a low-overhead method for maintaining an estimate of popularity that utilizes feedback (or lack there-off) obtained from previous searches. Simulation results show that the performance of the equation-based adaptive search is significantly better than the non-adaptive random walk and other straight-forward adaptive mechanisms.

1 Introduction

Locating a resource or service efficiently is one of the most important issues related to unstructured decentralized peer-to-peer networks. The objective of a search mechanism is to successfully locate resources while incurring low overhead and low delay. In order to fulfill this objective several random walk based search algorithms [24, 15, 2, 29] have been proposed for resource discovery in decentralized peer-to-peer networks. However, no analytical model has been proposed to quantify the effect of the parameters of random walk (such as the number

[*]A part of this work was presented in the Second International Workshop on Hot Topics in Peer-to-Peer Systems, July 21st, 2005, UCSD Campus, La Jolla, California, USA.

and the time to live (TTL) of the random walkers) on the performance of random walk search.

In this paper we present a mathematical model for random walk-based search mechanisms in peer-to-peer networks. Using the model we derive analytical expressions for the performance metrics, such as the delay, overhead and success rate, of the search in terms of the popularity of the resource being searched for and the random walk parameters. We propose an algorithm that uses the analytical expressions in order to adaptively set the parameters of random walk so that it maintains a certain minimum level of performance. The algorithm is referred to as *Equation Based Adaptive Search (EBAS)*[1]. We also propose a method for maintaining the popularity estimates of the resources in the network. The popularity estimator is based on an exponentially weighted moving average that smoothes out the high frequency (noise) components.

The work in this paper is based on an important observation made in [11] regarding the relationship between random walk and uniform sampling. It is shown in [11] that the set of nodes visited during m steps of random walk over the overlay graph of a P2P network has the same statistical properties as the set of m nodes drawn by performing independent uniform sampling over the nodes of the P2P network. In other words, random walk over a P2P network approximates independent uniform sampling. We use this property of random walk to derive analytical expressions for the performance metrics of random walk search in P2P networks. These analytical results are used to set the random walk parameters so that the required performance level may be achieved.

We perform extensive simulations in order to verify the analytical results of our model and compare the performance of EBAS with pure random walk and some simple adaptive search mechanisms. The simulation results show that the performance of random walk is significantly enhanced when its parameters are set in accordance to the derived analytical expressions. The methodology of EBAS may be extended to other random walk based search mechanisms in order to appropriately set their parameters.

The contributions made by this paper can be summarized as follows: (i) Analytical expressions for the performance metrics of random walk as a function of the number and TTL of the random walkers and the popularity of the resource being searched for, (ii) An algorithm, called EBAS, to adaptively adjust the parameters of random walk in order to achieve a desired performance, (iii) A feedback-based algorithm for maintaining popularity estimates of the resources in the network, and (iv) Simulation results that evaluate the proposed adaptive search methodology and compare it against other search mechanisms.

We now summarize the terminology used in this paper. A “search” is *successful* if it results in the discovery of

[1]This concept is somewhat similar to “equation based congestion control” [10].

at least one node that has the resource being searched for. The *success rate* is defined as the fraction of successful searches given that the resource being searched for is present in the network. The *popularity*, p, of a resource is defined as the probability that a randomly chosen node has the resource i.e. it is the ratio between the number of nodes that have the resource and the total number of nodes in the network. The search *delay* is the duration of time between the start of a search for a resource and the time when either a copy of the resource is found or all the random walkers terminate. The search *overhead* is the total number of bytes transmitted in the network for locating the resource. A *resource* may be a media file, an e-book, storage space or idle processor cycles. The performance of a search method is measured in terms of three metrics; (i) success rate, (ii) overhead and (iii) delay. It is desired that a search method has high success rate, low delay and low overhead.

The rest of the paper is organized as follows. In section 2 we discuss the advantages of random walk and highlight the importance of adaptively choosing its parameters. Section 3 positions our work in context with other related research in this area. Section 4 presents the network model and the analytical results for the random walk performance metrics as a function of the random walk parameters. Section 5 presents the EBAS algorithm. Simulation results and comparisons are presented in Section 6. Section 7 concludes the paper and outlines future work.

2 Background

Gnutella [1], a popular unstructured and decentralized P2P application, employed flooding of search queries in order to locate files in the overlay network. This resulted in huge overhead search overhead. In August 2000 the Gnutella community grew to such enormous size that the entire bandwidth of Gnutella users connecting to the Internet using dial-up connection was "chocked" because of the large amount of queries they forwarded [8]. Although the unstructured P2P architecture is very appealing due to its properties of fault tolerance, self-organization and low overhead associated with node arrival and departure, efficient search methods are very important for the scaling of such networks.

Random walk is a popular alternative to flooding for locating resources in P2P networks. A node, which we call the "querying" node, that needs to locate a resource, sends k queries (i.e. packets) to randomly selected neighbors. Each of these k queries is often called a *random walker*. Each random walker has a time to live (TTL) field that is initiated with some value $T > 0$ that limits the number of times the random walker is forwarded. When an intermediate node receives a random walker, it checks to see if it has the resource. If the intermediate node doesn't have the resource, it checks the TTL field, and if $T > 0$, it decrements T by 1 and forwards the query to a randomly chosen neighbor, else if $T = 0$ the query is not forwarded. On the other hand, if the intermediate node has the resource, the query is not forwarded and a reply is sent to the querying node.

Unlike flooding, the overhead of random walk is independent of the underlying topology. If the overlay network has an average degree d, then flooding the network with a query having TTL T will produce $O((d-1)^T)$ packets on the average. Studies in [16] have shown that the average node degree of Gnutella network is 3.5 and on the average 95% of all nodes are within 7 hops. If flooding is used for querying in this network, with TTL set to 7, the average overhead will be approximately 2.5^7. On the other hand the overhead of a search involving k random walkers, each with TTL T, is bounded by $k \times T$ packets irrespective of the underlying topology. Specifically the overhead of random walk varies linearly with k and T while the overhead of flooding grows exponentially with T.

The performance of random walk largely depends on the choice of k and T. Intuitively, the average number of nodes required to be probed for discovering a resource is inversely proportional to the popularity of the resource. Choosing low values of k and T for searching for a resource with low popularity would result in low success rate and high delays while choosing high values of k and T for searching for a resource with high popularity would result in excessive overhead. Thus, the parameters of random walk must be chosen according to the popularity of the resource being searched for. The popularity of a resource may not be known a-priori at the querying node. In addition, the popularity may change due to the arrival/departure of nodes, replication/deletion/exhaustion of resources or other random changes in the network. Thus the parameters of random walk must be set in an adaptive manner.

Some examples of P2P applications where adaptive search based on the popularity of resources would be most effective are:

1. Applications where the same resource is searched for multiple times by a node. For example a P2P application for distributed computing may search for a certain size of storage space or a certain number of available processor cycles whenever more data is ready for storing or processing. In this case a popularity estimate of a resource can be maintained using feedback from previous search results. Notice that the popularity of a resource may change not only due to nodes joining/leaving the overlay network but also due to the exhaustion of the resource over time.

2. Applications where the same resource is not searched for multiple times by a node but the resources may be categorized such that resources belonging to the same category are likely to have the same popularity. For

example, music files may be categorized on the basis of artists or genre (or both). In this case the popularity estimate of the category can be maintained based on feedback from previous search results of resources belonging to that category. Notice that the popularity of a resource may change not only due to the arrival and departure of nodes but also due to the replication or deletion of the resource over time.

3 Related work

Studies aimed at modeling P2P networks have revealed that the overlay network has characteristics similar to small world networks [17, 16]; high clustering coefficient, power-law distribution of node degree and small average path lengths. A method for growing a connected graph that has the properties of small world networks has been described in [13]. This method is used in our paper to generate overlay P2P topologies for validation of our analytical results and performance evaluation of our proposed algorithm. In [11] the authors show that random walk on a peer to peer network has statistical properties similar to independent sampling from a uniform distribution. This result is used in the analytical derivations in Section 4.

Several search algorithms for unstructured P2P networks have been proposed. These algorithms vary in terms of the kind and amount of state information maintained at the nodes to assist in forwarding the search queries. Some algorithms maintain little or no state information while other algorithms maintain large amount of state information and use complex algorithms to update it. Flooding of search queries was used by initial versions of Gnutella [1]. In iterative deepening [31], the querying node performs flooding with increasing depth until the search is successful. The search query is forwarded to randomly selected subset of neighbors in modified BFS [18]. In k-random walk [24], the querying node employs k random walkers to search for the desired resource. A two level random walk is proposed in [15]. All the above mentioned algorithms require that the nodes store negligible amount of state information.

In contrast some algorithms attempt to improve the performance of random walk by maintaining state information at the nodes in order to direct the search queries. They are expected to improve the performance (in terms of higher success rate) at the cost of higher complexity. Forwarding random walkers to neighbors with probability proportional to their degree is proposed in [2]. Directed BFS [31] seeks to improve performance by forwarding queries to neighbors that are more likely to return successful results. Intelligent BFS [18] uses a peer ranking mechanism to forward queries to "good" neighbors. The local indices technique [31] seeks to improve performance by maintaining indices of data stored by all nodes within k hops at each node. Adaptive Probabilistic

Search (APS) is proposed in [29]. APS employs k random walkers to search for the required resource. Each node maintains an indices table whose (i, j) entry corresponds to the probability with which the node will forward a random walker searching for resource i to neighbor j. The probabilities are updated after each query. Such search mechanisms, that maintain large amount of state information, are referred to as *state-full* search algorithms.

Our work differs from the existing research primarily because of its analytical approach. Random walk has been used as a basic component of most of the probabilistic search mechanisms proposed in the literature. It is commonly known that delay decreases with increase in k while success rate and overhead increase with increase in k and/or T [24]. However, to the best of our knowledge, quantifying the effects of the parameters k and T on the performance of random walk search has not been previously studied.

EBAS, the search algorithm proposed in this paper, lies in the category of state-full search algorithms. However, compared to other state-full search mechanisms EBAS takes a very novel approach. All the proposed state-full search mechanisms use the state information for deciding how to direct the search queries while little attention has been paid to optimizing the TTL and the number of random walkers deployed. As discussed earlier, these parameters must be set according to the popularity of the resource being searched for. If these parameters are not properly set, then even the state-full search mechanisms might perform badly. For example, the use of large k in searching for a highly popular resource would lead to a large overhead even if good indices are maintained by the nodes in APS. With EBAS, we introduce a framework for adaptively setting the parameters of random walk based on analytical expressions of the performance metrics, rather than using fixed values. Unlike most of the state-full search algorithms, EBAS does not maintain state information about the outgoing links or immediate neighborhood. Instead, each node maintains estimates of the popularity of the resources in the network.

4 Analytical model

In this section we first present the underlying assumptions of our model used for analysis. We then derive analytical expressions for the success rate, expected overhead and expected delay of (stateless) random walk in terms of the popularity of the resource (p), number of random walkers (k) and TTL (T).

4.1 Modeling and assumptions

The P2P network is assumed to have small world properties [2] [17, 16] (see Section 3). The power-law exponent of the node-degree distribution of the P2P graph is assumed to be greater than 2. This is observed, for example, in Gnutella snapshots [8]. This assumption implies that the difference between the $1^{\rm st}$ and $2^{\rm nd}$ eigenvalues of the probability transition matrix, known as *spectral gap*, of the P2P graph is bounded by a constant [25].

In order to keep the mathematical treatment simple enough for efficient real-time implementation, we assume that all query packets have the same size and we measure the overhead in terms of number of packets. We also assume that the time required to transmit a query packet over each hop (i.e. from a node to its neighbor) is constant, denoted by ρ. In each time step of duration ρ, one node is probed by each random walker. Delay is expressed in terms of number of time steps elapsed before a random walker visits a node that has the resource. This is also equal to number of nodes visited by the random walker that is first to discover the resource.

We present analytical results for "*pure*" random walk. By "pure" random walk we mean that state information is not maintained by any node. This implies that each node forwards the random walkers without any bias towards any particular neighbor, and that multiple random walkers visiting the same node are forwarded independently. A random walker is terminated in two cases: 1. If the random walker visits a node that has the resource (*successful termination*), and 2. If TTL of the random walker expires (*unsuccessful termination*).

4.2 Analytical results

In the analysis below, we invoke the following result [11]: For families of graphs where the spectral gap is greater than some constant, consecutive steps of random walk are excellent candidates to approximate independent uniform sampling. A large spectral gap corresponds to good global connectivity in the graph. Many existing complex graphs, such as Internet topologies (AS topology, router topology, etc.) and P2P overlay graphs, have been found to have a constant spectral gap [25]. Also, dynamic networks with large spectral gaps may be maintained by incurring a constant overhead for each node join/leave operation [11, 21].

Intuitively this means that on a graph with good connectivity, a random walk quickly looses its memory (i.e., knowledge of previous nodes visited), such that the sequence of nodes visited by it in k steps approximates, in the statistical sense, the sequence of nodes selected by randomly drawing k nodes using uniform sampling. Since the spectral gap of the class of P2P graphs (power law exponent ≥ 2) considered in this study is bounded by a constant,

[2] This assumption implies that the network has no bad cuts or bottlenecks.

we may approximate random walk with independent uniform sampling. In what follows, we first summarize [11] which shows the correspondence between random walk and uniform sampling. Then we derive the analytical results for the performance measures of interest in this paper.

4.2.1 Correspondence between random walk and uniform sampling

Two abstractions of independent sampling are used in [11] to compare uniform sampling and random walks: *coupon collection problem* and *independent Bernoulli trials*.

Consider n distinct kinds of coupons and an infinite population. Each member of the population is handed out exactly one coupon, which is equally likely to belong to any of the n kinds. Now the probability that a randomly selected person from the population has the coupon of type i is $1/n$. The coupon collection problem is to collect at least one coupon of each kind by randomly selecting people from the population and collecting coupons from them. Let T_n be the time it takes to collect a coupon of each kind and $T_{\gamma n}$ be the time it take to collect γn distinct coupons for $0 < \gamma < 1$. Then from [26]

$$E[T_n] = O(n \log n) \tag{1}$$

$$E[T_{\gamma n}] = \frac{1}{1-\gamma} O(n) \tag{2}$$

The time taken by the coupon collector to collect all the coupons by uniform sampling is analogous to the cover time[3] of the random walk. Consider random walk over an undirected connected graph $G(V, E)$, with $|V| = n$. Let $d_{\min} = \min_{v \in V} d_v$ be the minimum vertex degree in the graph. Random walk over G has transition probability matrix A, where $a_{ij} = \frac{1}{d_i}$ (d_i is degree of vertex i). From [6], A has n eigenvalues values $\lambda_1, \lambda_2, \ldots, \lambda_n$ such that $1 = \lambda_1 > \lambda_2 \geq \lambda_3 \geq \ldots \geq \lambda_n > -1$. For a graph with good global connectivity properties (no bad cuts/high graph conductance), $\lambda_2 << 1$ which is a reasonable assumptions for all communication networks.

Let C_n be the cover time of a random walk on a graph of n nodes and let $C_{\gamma n}$ be the time taken by the random walk to visit γn distinct nodes. Then for $d_{\min} = \Omega(\frac{|E|}{n})$, $E[C_n]$ and $E[C_{\gamma n}]$ are given by ([6, 3])

$$E[C_n] \leq O(\frac{n \log n}{1 - \lambda_2}) \tag{3}$$

$$E[C_{\gamma n}] \leq \frac{1}{1-\gamma} O(\frac{n}{1-\lambda_2}) \tag{4}$$

[3]Cover time of a random walk on a graph is the time the random walk takes to visit all the nodes of the graph.

. Comparing (1),(2) with (3),(4), it is clear that both uniform sampling and random walk solve the coupon collection problem in the same order of time, provided that λ_2 is small and $d_{min} = \Omega(\frac{|E|}{n})$. Since the overlay network is well connected graph with low diameter, the second eigenvalue is small. The second constraint is also satisfied by most P2P networks since the number of connections with peers established by a new node joining the network is well bounded, by say K. Thus $|E| \leq nK$, which implies that $d_{min} = \Omega(\frac{|E|}{n})$.

Similarly an analogy may be established between uniform sampling and random walk by considering Chernoff bound [7, 12] on a sequence of Bernoulli trials. Consider k independent Bernoulli trials $X_1, X_2, \ldots X_n$ such that $P[X_i = 1] = p$ and $P[X_i = 0] = 1 - p$. Here p is analogous of the popularity of the resource being searched for. Let $X = \sum_{i=1}^{k} X_i$. The mean of X equals kp or mean of X/k equals p. The Chernoff bound for $\frac{X}{k}$ is given by

$$P[|\frac{X}{k} - p| \geq \xi p] \leq 2e^{-\frac{\xi^2 kp}{20}} \tag{5}$$

for $0 < \xi < 0.9$.

Now consider that p fraction of vertices of the graph G have the a resource. Let y_t be the vertex visited by the random walker at time t, after it achieves steady state. We define a random variable Y_t, Y_t equals 1 if y_t has the resource, otherwise it is zero. Similar to Bernoulli trials we define $Y = \sum_{t=\tau}^{\tau+k} Y_t$. According to [11]

$$P[|\frac{Y}{k} - p| \geq \xi p] \leq 8e^{-\frac{\xi^2 kp^2(1-\lambda_2)}{20}} \tag{6}$$

Equations (5) and (6) indicate that for random walk on a well connected graph (low λ_2), using $\frac{Y}{k}$ as estimator of p has same the statistical property as $\frac{X}{k}$.

4.2.2 Derivation of Performance Metrics

Following the above discussion, we approximate random walk on a P2P graph as independent uniform sampling. Each node visited by a random walker may be assumed to be an independent sample from a space of uniform distribution of the nodes of the graph. From the definition of popularity and this result, the probability that a node visited by a random walker has the resource equals the popularity of the resource. The accuracy of this approximation depends on the actual spectral gap of the graph; the higher the gap the better the approximation.

Success Rate: The success rate is the probability that a search is successful in discovering the resource being searched for. A random walker terminates unsuccessfully if none of the nodes visited by it has the resource. The probability that a node visited by a random walker does not have the resource is $(1-p)$. So the probability that a random walker is unsuccessfully terminated is $(1-p)^T$. The probability of unsuccessful termination of k independent random walkers is given by $(1-p)^{k\cdot T}$. Thus the probability of success of a search, employing k random walkers each with TTL T, for a resource with popularity p, is given by

$$p_s = 1 - (1-p)^{kT} \tag{7}$$

Expected Overhead: The expected overhead is the expected number of packets transmitted over the network after a node initiates a random walk search. Let O_i denote the overhead incurred by the i^{th} random walker where $i \in [1, 2, \ldots, k]$. Clearly $1 \leq O_i \leq T$. For $1 \leq j < T$, $O_i = j$ corresponds to the event that the first $j-1$ nodes visited by the random walker do not have the resource while the node visited by the random walker at the j^{th} step has the resource. This event may occur with probability $p(1-p)^{j-1}$. $O_i = T$ corresponds to the event that the first $T-1$ nodes visited by the random walker do not have the resource, which has probability $(1-p)^{T-1}$. Thus

$$P[O_i = j] = \begin{cases} p(1-p)^{j-1} & j = 1, 2, .., T-1 \\ (1-p)^{T-1} & j = T \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

and hence,

$$\begin{aligned} E[O_i] = \sum_{j=1}^{T} j \cdot P[O_i = j] &= \sum_{j=1}^{T-1} j.p(1-p)^{j-1} + T.(1-p)^{T-1} \\ &= \frac{1-(1-p)^{T-1}}{p} + (1-p)^{T-1} \end{aligned} \tag{9}$$

where $E[\cdot]$ denotes the expectation operation. Let O denote the total overhead incurred by a search that deploys k random walkers. Then,

$$\begin{aligned} E[O] = & \quad E[\textstyle\sum_{i=1}^{k} O_i] = \sum_{i=1}^{k} E[O_i] \\ = & \quad k \cdot \left(\frac{1-(1-p)^{T-1}}{p} + (1-p)^{T-1} \right) \end{aligned} \tag{10}$$

Expected Delay: The expected delay equals the expected number of time steps elapsed between the initiation of a search and either the discovery of a node possessing the resource or unsuccessful termination of the random

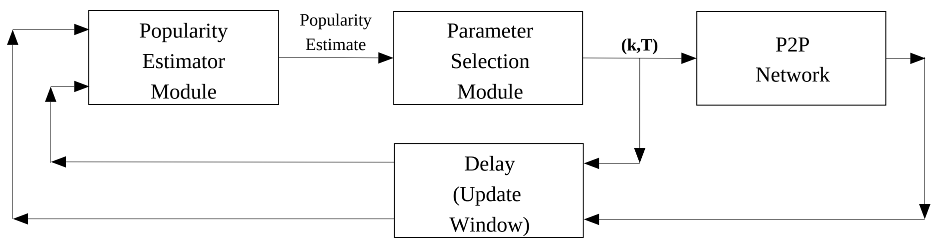


Figure 1. Main components of the feedback based adaptive search. The feedback obtained from previous searches is used to estimate the popularity of resources. The popularity estimate is used to choose the parameters for future searches.

walkers. Let the delay of a search be equal to D. Obviously $1 \leq D \leq T$. For $1 \leq j < T$, $D = j$ corresponds to the event that none of the nodes visited by the k random walkers during the first $j-1$ steps possess the resource and at least one of the random walkers visits a node with the resource at the j^{th} step. This occurs with probability $(1-p)^{k(j-1)}(1-(1-p)^k)$. $D = T$ requires that none of the k random walkers visits a node that has the resource in the first $T-1$ steps, which may happen with probability $(1-p)^{k(T-1)}$. Thus,

$$P[D = j] = \begin{cases} (1-(1-p)^k)(1-p)^{k\cdot(j-1)} & j = 1, 2, \ldots, T-1 \\ (1-p)^{k\cdot(T-1)} & j = T \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

So the expected delay is given by

$$\begin{aligned} E[D] &= \sum_{j=1}^{T} j \cdot P[D = j] = \sum_{j=1}^{T-1} j.(1-(1-p)^k)(1-p)^{k.(j-1)} + T.(1-p)^{k.(T-1)} \\ &= \frac{1-(1-p)^{k.(T-1)}}{(1-(1-p)^k)} + (1-p)^{k.(T-1)} \end{aligned} \tag{12}$$

5 Equation based adaptive search

EBAS consists of two components as shown in Figure 1; a *parameter selection module* and a *popularity estimator*. They are described in the next two subsections.

5.1 Parameter selection module

The objective of the parameter selection module is to select k and T for discovering a resource with popularity p such that a target performance is guaranteed. Our methodology can be summarized as follows. First, we formulate the desired performance as bounds on the success rate, delay, and overhead. Second, we find the range of feasible (k, T) pairs (if they exist) that satisfy the desired bounds. Third, within a feasible set of (k, T) pairs, we characterize the tradeoff between the delay and overhead. Finally, we discuss one possible approach to implement the parameter selection module and apply our methodology to a specific example.

The desired performance of the random search algorithm could be formulated as the following set of constraints:

$$p_s \geq 1 - \epsilon_p, \qquad 0 < \epsilon_p \ll 1 \tag{13}$$

$$E[O] \leq \alpha_p, \qquad \alpha_p \geq 1 \tag{14}$$

$$E[D] \leq \delta_p, \qquad \delta_p \geq 1 \tag{15}$$

where ϵ_p, α_p and δ_p are design parameters that characterize the minimum acceptable performance of the search algorithm for a resource with popularity p. The design parameters depend on the requirements of the application and the popularity of the resource.

Substituting the above constraints (13), (14) and (15) in (7), (10) and (12), respectively, results in three inequalities that relate k and T with the design parameters for any $0 < p \leq 1$. For example, substituting (13) in (7) yields

$$k \cdot T \geq \frac{log(\epsilon_p)}{log(1-p)} \tag{16}$$

For the other two inequalities, obtained by substituting (10) and (12) in (14) and (15) respectively, it is not possible to explicitly separate the design parameters on one side and k, T, on the other, and thus must be solved numerically.

Let $S(p)$ denote the set of (k, T) feasible pairs for a resource with popularity p. $S(p) = \phi$ if there exists no (k, T) pair that satisfies each of these three inequalities. If $S(p) = \phi$ then the parameter selection module may inform the designer to relax the design parameters for the popularity value p by increasing ϵ_p, α_p and δ_p. This change might be accompanied with a corresponding change in the application design in order to make it tolerant to the loss in performance. If the designer's intervention is not possible then the parameter selection module may choose (k, T) such that p_s is maximized while satisfying (14) and (15).

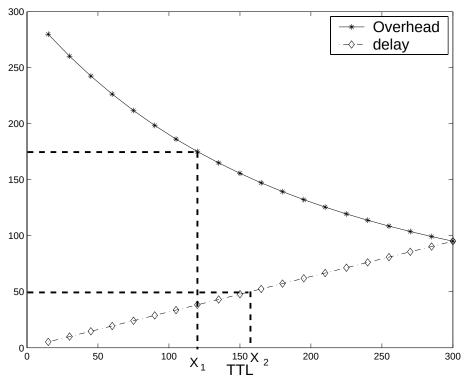


Figure 2. Values of $T \in [X_1, X_2]$ and $k = \frac{log(\epsilon_p)}{T \cdot log(1-p)}$ belong to $S(p)$ - the set of feasible (k, T) pairs for which constraints (13) - (15) are satisfied.

If $S(p) \neq \phi$, then there is a tradeoff between overhead and delay within the feasible set. The overhead increases with increase in k and/or T while delay decreases with increase in k. Thus, it is up to the designer to choose the operating point (i.e. the (k, T) pair) from the feasible set.

The following numerical example illustrates how a (k, T) pair is chosen to satisfy the design constraints. Consider a case where a node needs to search for a resource with $p = 0.01$. Suppose that the constraints for the application are given by $\epsilon_{0.01} = 0.05$, $\alpha_{0.01} = 175$ and $\delta_{0.01} = 50$. From (16), $k \cdot T \geq 298$ for satisfying the constraint on success rate. For $k \cdot T = 300$, Figure 2 shows the expected delay and overhead as a function of T. One feasible value of T is 150, and the corresponding k is 2. So the node sends 2 random walkers with TTL set to 150 to search for this resource.

One approach to implement this module is as follows. Each node has a table, called *parameter selection* table with values of k and T corresponding to intervals of popularity. When a node needs to search for a resource with popularity p, it looks up the entry in the parameter selection table corresponding to the interval in which p lies, and initiates random walk with parameters k and T specified in the entry. Table 1 is an example of a parameter selection table. So it is ensured that the desired average success rate is achieved while the average overhead and delay are within the specified bounds. This is in contrast to pure random walk whose parameters remain constant which often leads to low success rate or excessive overhead.

Since the parameter selection table has to be constructed only once, brute force methods such as exhaustive search may be used in order to construct the table. In order to find the entry of parameter selection table corresponding to popularity p, we begin by determining the lower bound on $k \cdot T$ using (16). Let the lower bound be

Popularity Interval	T	k
$[0,\ 1 \times 10^{-3})$	150	15
$[1 \times 10^{-3},\ 2 \times 10^{-3})$	150	13
$[2 \times 10^{-3},\ 3 \times 10^{-3})$	150	11
$[3 \times 10^{-3},\ 4 \times 10^{-3})$	150	9
$[4 \times 10^{-3},\ 5 \times 10^{-3})$	150	8
$[5 \times 10^{-3},\ 7 \times 10^{-3})$	150	4
$[7 \times 10^{-3},\ 9 \times 10^{-3})$	150	3
$[9 \times 10^{-3},\ 1)$	150	2

Table 1. An example of parameter selection table.

L_p. We then fix $k = k'$ and plug each value of $T \geq \frac{L_p}{k'}$ and k' into expressions for expected delay and overhead. If (14) and (15) are satisfied for T and k', the (k', T) pair is added to the feasible set $S(p)$. Since both $E[O]$ and $E[D]$ are increasing functions of T ($\partial E[O]/\partial T > 0$ and $\partial E[D]/\partial T > 0$), we stop this procedure when one of the constraints (14) or (15) is violated and move on to the next value of k. We repeat this for all values k in the interval 1 through L_p. At the end we have a feasible set with (k, T) pairs that satisfy the performance constraints. One of the pairs belonging to the feasible set could be entered into the parameter selection table for choosing parameters for searching resource with popularity p. This choice would depend upon various factors like the designer's desire to decrease overhead and the application's tolerance to delays. We may speed up this process by entering the first (k, T) pairs that is found to satisfy all the constraints. This method is repeated for each value of p at regular intervals to construct the complete parameter selection table. The length of the intervals determine the size of the parameter selection table and the time it takes to construct it. For searching a resource with popularity $\hat{p}$, the entry of the parameter selection table corresponding to the largest p less than or equal to $\hat{p}$ is used.

5.2 Maintaining estimate of popularity

The popularity of a resource is required for the parameter selection module to set the values of the parameters of random walk. However, the popularity of resources in the network change over time because of arrival and departure of nodes or because of exhaustion/deletion/replication of resources. Centralized methods for informing nodes about the popularity of the resources would incur large overhead. In this section we describe methods for maintaining the popularity estimate with the help of feedback from previous searches, and adapting the parameters of the random walk as a function of this variable estimate.

Suppose there are n different resources of interest in the network. Each node maintains a popularity estimate of each resource in a *popularity table*. The i^{th} entry of the table corresponds to the popularity estimate of i^{th} resource. When a node joins the network, it needs to initialize its popularity table. The initialization of the popularity table is done by selecting a neighbor and querying it for its popularity table. The neighbor then transfers its current popularity table to the node that has recently joined. Thereafter, the popularity estimate of a resource is updated after feedback from l most recent searches for the resource. The interval between two successive updates is referred to as *update window*.

The algorithm works as follows. Within each update window, say the j^{th} update window, an instantaneous popularity estimate is computed from the parameters k_j, T_j and the success rate during that window. An exponentially weighted moving average is used to update the popularity estimate. New parameters k_{j+1} and T_{j+1} for the random walk for the next (i.e. $j+1^{st}$) window are then computed by the parameter selection module using the updated popularity estimate.

Specifically, let $\hat{p}_i(j)$ denote the popularity estimate of the i^{th} resource during the j^{th} update window. Let $r_i(j)$ denote the fraction of successful searches in the j^{th} update window. From (7), the instantaneous popularity estimate of the i^{th} resource computed from the searches within the j^{th} window, denoted by $\hat{q}_i(j)$, is given by

$$\hat{q}_i(j) = 1 - e^{\frac{log(1-r_i(j))}{k_j \cdot T_j}} \tag{17}$$

The popularity estimate is updated after each update window using

$$\hat{p}_i(j+1) = \beta \cdot \hat{p}_i(j) + (1-\beta) \cdot \hat{q}_i(j+1), \quad 0 < \beta < 1 \tag{18}$$

$\hat{p}_i(j+1)$ is used by the parameter selection module to select k_{j+1} and T_{j+1} that will be used in the next update window.

The performance of the popularity estimator module largely depends on the size of the update window, l, and the *smoothing factor*, β, used to maintain the popularity estimate. The larger the size of the update window, the closer would be the fraction of successful searches in the update window to the real success rate. As a result a large update window would imply that the instantaneous popularity estimate is closer to the instantaneous popularity of the resource. The effect of the size of the update window on the instantaneous popularity estimate may be analyzed by treating each search within an update window j as a Bernoulli trial. The outcome of a trial is 1 with probability $p_s = 1 - (1-p)^{k_j T_j}$ (i.e. the search is successful) and 0 with probability $1 - p_s$. Assuming that the popularity of the resource remains constant during an update window, the fraction of successful searches in the update window is an estimate of the mean of the Bernoulli random variable (p_s). We know that the standard deviation of the Bernoulli random variable, denoted by σ, equals $\sqrt{p_s(1-p_s)}$. So from basic statistics [14] we may conclude that for an update window of size l searches, p_s lies in the range $[q_i(j) - \frac{z_{0.99}\sigma}{\sqrt{l}}, q_i(j) + \frac{z_{0.99}\sigma}{\sqrt{l}}]$ with 99% confidence, where $\mathrm{erf}(z_{0.99})^4$ = 0.99. Therefore, with 99% confidence, the real popularity p of the resource would lie in the range $\left[1 - e^{(\log(1-r_i(j)+\frac{z_{0.99}\sigma}{\sqrt{l}})/k_jT_j)}, 1 - e^{(\log(1-r_i(j)-\frac{z_{0.99}\sigma}{\sqrt{l}})/k_jT_j)}\right]$, or

$$P[p \in [1 - e^{(\log(1-r_i(j)+\frac{z_{0.99}\sigma}{\sqrt{l}})/k_jT_j)}, 1 - e^{(\log(1-r_i(j)-\frac{z_{0.99}\sigma}{\sqrt{l}})/k_jT_j)}] = 0.99 \tag{19}$$

The uncertainty in the estimate of p may be represented by the size of the 99% C.I. of p, which is equal to

$$\mathcal{L}_{0.99} = e^{(\log(1-r_i(j)+\frac{z_{0.99}\sigma}{\sqrt{l}})/k_jT_j)} - e^{(\log(1-r_i(j)-\frac{z_{0.99}\sigma}{\sqrt{l}})/k_jT_j)} \tag{20}$$

From (20) it is clear that the uncertainty in estimating p decreases with the increase in the size of update window l. We further illustrate this with the help of an example. Consider that the popularity of the resource of interest is 0.01 and $k = 3$, $T = 100$ is used for random walk search within an update window. Suppose that 95.2% of the searches were successful within the update window. Figure 3 shows how the certainty in terms of the length of $\mathcal{L}_{0.99}$ varies with the size of the update window (l). It is observed that initially the uncertainty decreases rapidly with l while the rate of decrease diminishes as l becomes large. This because the uncertainty depends on $\sqrt{l}$, which grows slowly as l increases.

From the above discussion it seems that we should use as large an update window as possible. However the above analysis is based on the assumption that the popularity of the resource remains constant throughout the update window. In reality the popularity of a resource may change during an update window. This might lead to serious performance degradation if the size of the update window is too large. This is because the values of the search parameters (k, T) are updated only at the end of an update window. So if an update window is very large and the popularity of the resource decreases during an update window then a large number of searches within the window might be unsuccessful since the system would still be using the search parameters set according to the larger value of popularity. So although it desirable to have as large an update window as possible, its size must be chosen according to the frequency with which the popularity of the resources in the network changes.

[4] $\mathrm{erf}(z) = \frac{2}{\pi}\int_0^z e^{\tau^2}\mathrm{d}\tau$

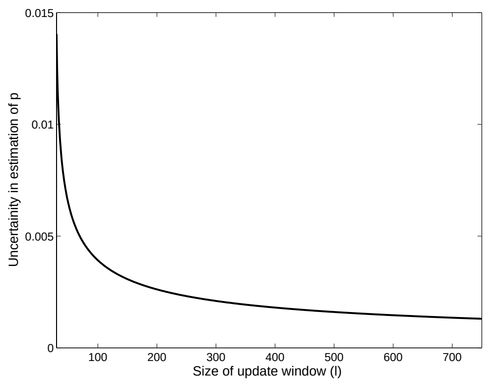


Figure 3. The effect of the update window size on the uncertainty of popularity estimation. The uncertainty decreases as the size of update window increases, provided p remains constant during the window. Also the rate of decrease in uncertainty decreases as the update window size increases.

The smoothing factor β decides what weight must be given to the instantaneous popularity with respect to the old popularity estimate while evaluating the new popularity estimate. If we have large confidence that the instantaneous popularity estimate accurately reflects the current popularity of the resource, then β should be small. However if the instantaneous popularity estimate is not very accurate due to a small update window or due to frequent fluctuations in the popularity of the resource, then β should be large.

6 Simulation results

This section presents extensive simulations that verify that: (i) The analytical results (Section 4.2) are in agreement with the simulation results, (ii) The popularity estimator module (Section 5.2) effectively maintains an estimate of the variable popularity, (iii) The parameter selection module (Section 5.1) achieves the target design objectives of equations (13), (14) and (15), (iv) EBAS provides tremendous improvements over adaptive flooding and (v) EBAS performs better than the non-adaptive random walk and other simple adaptive techniques such as *additive-subtractive parameter adjustment*. The simulation scenario and results are presented below.

6.1 Simulation scenario

The network graph used for simulations consists of 10^4 nodes. The network is grown using the methodology specified in [13], which is an extension of Barabasi's model [4]. This ensures that the node degree follows a power law distribution (defined in [16]) and the network has high clustering coefficient. The average node degree and the exponent of power law of the network are 3.5 and 3, respectively, which are close to the actual values of Gnutella network as observed in [28] .

In order to populate the network with a resource with popularity p, $p \times 10^4$ nodes are randomly selected and marked to own the resource. In the simulations of EBAS, the update window size, l, is set to 100 and the smoothing factor, β, is set to 0.1. In order to make the interpretation of results convenient, time is normalized with respect to the update window size. So time $t = j$ refers to the time at which the j^{th} update of popularity estimate takes place. During an update window the querying node keeps track of the number of searches that have been successful and uses it to calculate the fraction of successful searches. This fraction is used to evaluate the popularity estimate using (17) and (18). The popularity estimate is used to select the search parameters (k, T) from the parameter selection table which are used for searches in the next update window. While constructing the parameter selection table for the simulations, if more than one (k, T) pairs satisfy the performance constraints, then we randomly choose one of the (k, T) pairs to be included in the table.

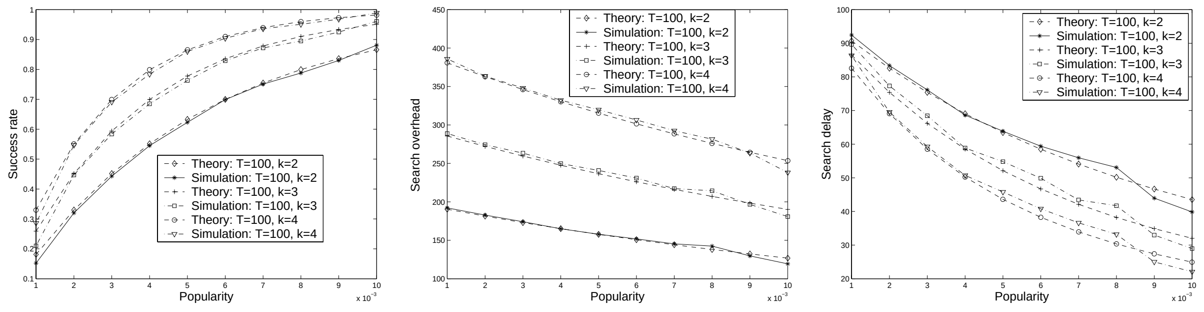
6.2 Verification of analytical results

For verifying the analytical results derived in subsection 4.2 we simulated pure random walk and compared the values of average success rate, overhead and delay thus obtained against the numerical values obtained from equations (7), (10) and (12) respectively. The average values of performance metrics are obtained by averaging results of 10^4 runs of the same search in order to get good estimate of average values. The resource distribution remains the same for all the 10^4 runs of the search.

Figures 4(a), 4(b) and 4(c) show plots of success rate, expected overhead and expected delay, as obtained from (7), (10) and (12), along with average values obtained from simulations. It is observed that simulation results agree closely with the analytical results. The slight deviation in the average values obtained from simulations is because the random walk is statistically similar to independent uniform sampling only after a sufficient number of steps (see [11] for detailed discussion).

6.3 Performance of popularity estimator

In order to test the performance of the popularity estimator we changed the actual popularity of the resource during the simulation and observed how the popularity estimate, calculated using feedback from previous searches, changes with changing popularity. Figure 5 shows the actual popularity and estimated popularity of the resource



(a) Success rate (b) Search overhead (c) Search delay

Figure 4. The simulation results agree well with the analytical results for success rate, average overhead and delay.

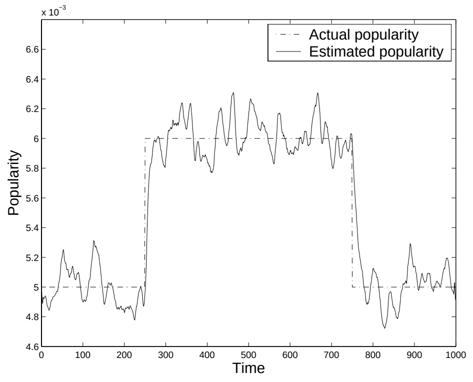


Figure 5. When the popularity of a resource is changed during simulations, it is observed that the estimated popularity closely follows the actual popularity.

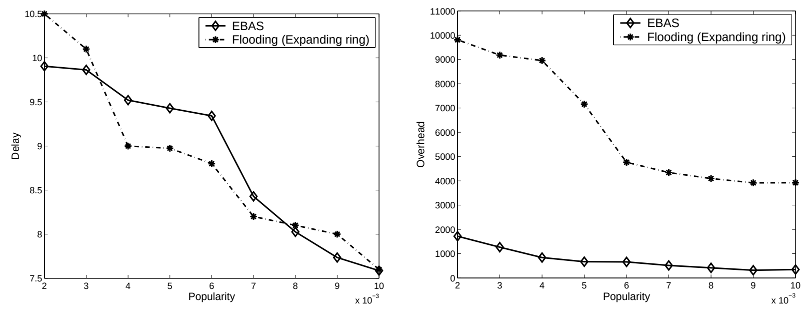
as obtained from simulations. Initially the actual popularity of the resource is 5×10^{-3} and is increased to 6×10^{-3} at $t = 250$ and later decreased to 5×10^{-3} at $t = 750$. Figure 5 shows that the popularity estimate very closely follows the actual value of popularity. So the proposed popularity estimator is efficient in tracking popularity changes.

6.4 Satisfaction of design objectives

In Section 5.1 we described the parameter selection module, which chooses the parameters of random walk such that the constraints of (13), (14) and (15) are satisfied. In order to verify this we ran simulations by populating the network with resources of various popularity. The parameters of random walk for discovering each resource

p	ϵ_p	α_p	δ_p	k	T	$\overline{p_s}$	$\overline{O}$	$\overline{D}$
0.01	0.05	175	50	2	150	0.95	158	30
0.007	0.05	325	50	3	150	0.96	290	47
0.005	0.05	500	50	4	150	0.97	431	50

Table 2. Search parameters and corresponding performance of EBAS for various values of resource popularity and performance criterion. It is observed that EBAS is able to satisfy the performance constraints.



(a) Search delay (b) Search overhead

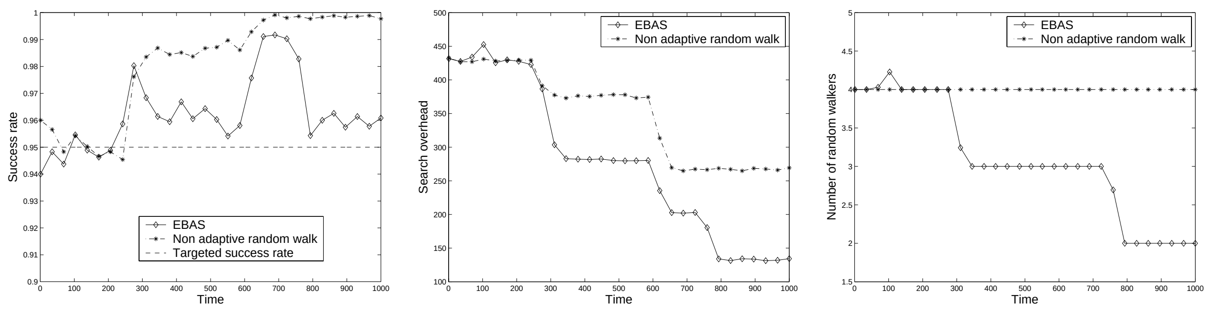
Figure 6. Comparison of performance of EBAS with that of ER. It is observed that although the delay incurred by EBAS and ER is similar, the overhead incurred by EBAS is considerably smaller than ER.

are adjusted according to the design constraints. The performance metrics ($\overline{p_s}$, $\overline{O}$ and $\overline{D}$) for each resource were measured. Table 2 shows the average values of the performance metrics obtained from simulations, along with constraints imposed in choosing the parameters (k, T) for resources with different popularity. The average values obtained from simulations satisfy all the constraints. Thus EBAS fulfills its objective.

6.5 Comparison of performance EBAS with expanding ring search

In this section we demonstrate that EBAS offers huge improvements over adaptive flooding techniques like *expanding ring (ER)*. In ER search, the querying node floods the network with a query with TTL T_{initial}. If the search is unsuccessful then the querying node floods the network with a query with TTL $T_{\text{initial}}+1$. In this manner, after each unsuccessful attempt, the querying node re-floods the network with a query whose TTL is one more than that during the previous attempt until the required resource is discovered or the TTL becomes equal to $T_{\max}$.

In order to evaluate the performance of EBAS and ER search for discovering a resource with popularity p,



(a) Success rate (b) Search overhead (c) Number of random walkers

Figure 7. Comparison of performance of EBAS with that of non-adaptive random walk for scenario 1. In this scenario the resource popularity increases with time. It is observed that EBAS incurs lower overhead while maintaining the desired success rate.

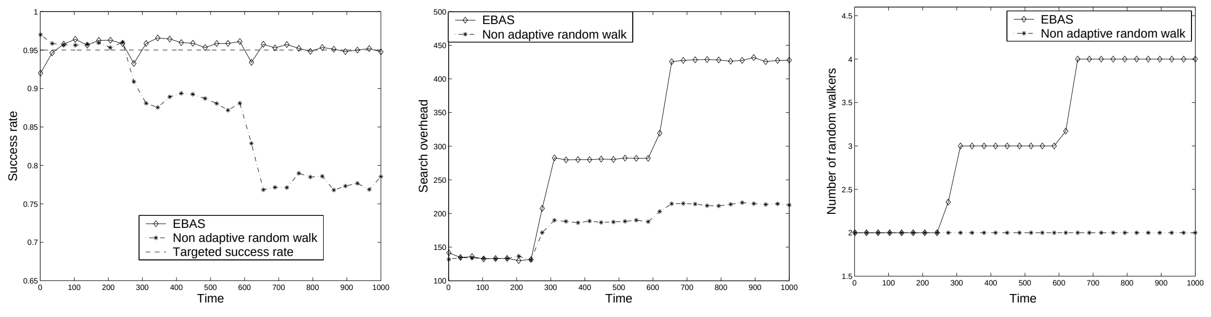
we choose a random querying node and repeat EBAS and ER searches for various resource distributions while keeping popularity equal to p. The overhead and delays of these searches is averaged to obtain the average delay and overhead for the given popularity. For the ER simulations we choose $T_{\text{initial}} = 1$, $T_{\max} = 10$. To make the comparison of ER and EBAS meaningful we choose $\delta_p = 10\ (= T_{\max})$ for EBAS simulations, i.e., we make sure that the upper bound on the average delay incurred by EBAS is comparable to that incurred by ER. Other parameters of EBAS simulations are: $\alpha_p = 2 \times 10^3$ and $\epsilon_p = 0.05$.

Figures 6(a) and 6(b) show the average overhead and delay over a large range of resource popularity. The figures indicate that although the delay incurred by EBAS is almost the same as that incurred by ER, EBAS incurs much less overhead than ER. Thus EBAS is a much more efficient search mechanism than ER.

6.6 Comparison of performance of EBAS with "pure" random walk

We compare EBAS with non-adaptive pure random walk in two scenarios. In one scenario the popularity of the resource increases with time and in the other scenario the popularity decreases with time. In both scenarios, the initial values of the random walk parameters for EBAS and non-adaptive random walk are equal.

Scenario 1: At $t = 0$ the resource of interest has popularity $p = 0.005$. The popularity is increased to 0.007 at $t = 250$ and further increased to 0.01 at $t = 600$. Figure 7(a) shows that both EBAS and non-adaptive search strategies maintain success rate above the targeted rate. A sharp increase in the success rate of EBAS is noticed at $t = 250$ and $t = 600$ corresponding to the increase in popularity of the resource. This is because the popularity



(a) Success rate (b) Search overhead (c) Number of random walkers

Figure 8. Comparison of performance of EBAS with that of non-adaptive random walk for scenario 2. In this scenario the resource popularity decreases with time. It is observed that EBAS maintains the desired success rate while the success rate of random walk is considerably less than the desired value.

estimator module uses weighted average sum to update the popularity and thus the popularity estimate does not immediately become equal to the real popularity. As a result EBAS continues to search for the resource with aggressive k and T, although the popularity of the resource has increased leading to a higher success rate. When the popularity estimate becomes equal to the real popularity and hence appropriate values for k and T are chosen, the success rate decreases slightly (but still remains above the desired level of performance). Figure 7(b) shows that the overhead of EBAS is less than the overhead of the non-adaptive search strategy after the popularity of the resource increases (i.e. after $t = 250$). The initial values of the parameters k and T are too aggressive for the higher value of popularity. The adaptive search strategy adjusts the value of parameters causing less overhead while non-adaptive random walk continues searching with fixed k and T thus incurring higher overhead. Figure 7(c) shows that the number of random walkers decrease adaptively with increasing popularity in case of EBAS while number of random walkers remain constant for the case of pure random walk causing higher overhead.

Scenario 2: At $t = 0$ the resource of interest has popularity 0.01 which is decreased to 0.007 at time $t = 250$ and to 0.005 at $t = 750$. Figure 8(a) shows that EBAS is able to maintain the success rate above the target value while the success rate of non-adaptive random walk falls much below the acceptable value as the popularity of the resource decreases. Figure 8(b) shows that in order to maintain success rate above the target value, EBAS pays a larger overhead since it adaptively increases the value of the search parameters. However, this additional overhead is *necessary* in order to make sure that the success rate remains above the required value. Figure 8(c) shows how the number of random walkers increases adaptively as the popularity decreases in case of EBAS while the number of random walkers remains constant in case of pure random walk causing low success rate.

In summary, the simulation results show that if the parameters of random walk remain fixed then it may lead to large overheads or poor success rate because of changing popularity (or poor initial estimate) of the resource of interest. In contrast, EBAS adjusts the parameters according to the current popularity estimate of the resource thus maintaining the desired performance.

6.7 Comparison of performance of EBAS with additive-subtractive parameter adjustment

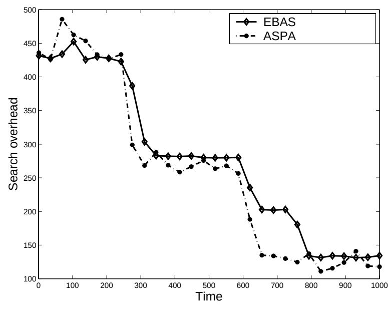
EBAS sets the value of the parameters of random walk search according to the mathematical model of random walk developed in section 4. To do that, EBAS needs to maintain the estimate of popularity of each resource. In this section we explore a simpler heuristic method for adjusting the values of the parameters and compare its performance with that of EBAS. We call the heuristic method *additive-subtractive parameter adjustment (ASPA)*.

In ASPA each node employs k random walkers, each with TTL T in order to search for a resource. If, over an update window, the success rate of a resource at a node is smaller than the required success rate p_s, then the node increments the number of random walkers by 1. On the other hand, if the success rate over an update interval is greater than p_s, then the number of random walkers is decreased by 1. ASPA is therefore much simpler than EBAS; it does not require maintaining a popularity estimate and is hence intuitively very appealing.

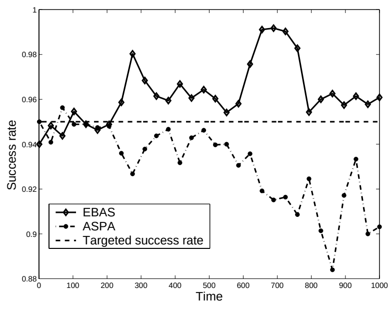
We compare the performance of EBAS with that of ASPA for Scenarios 1 and 2 described in Section 6.6. The delay and overhead for both scenarios are plotted in Figures 9 and 10. It is observed that although the overhead incurred by ASPA is almost the same as EBAS, ASPA fails to ensure a steady success rate. The success rate of ASPA falls rather dramatically when the popularity is high. This is because ASPA may react rather aggressively to small perturbations. On the other hand, EBAS is able to make more judicious choice on the basis of the popularity estimate and analytical model.

7 Discussion and conclusion

In this paper we developed analytical expressions for the success rate, delay and overhead of random search as a function of the random walk parameters and resource popularity. These results were used as a guideline for the design of a search method that uses popularity estimate in order to adaptively adjust the parameters of random walk while maintaining a target performance. Simulation results showed that (i) the analytical expressions match the simulation results, (ii) the proposed popularity estimation method closely tracks the popularity of the resource as

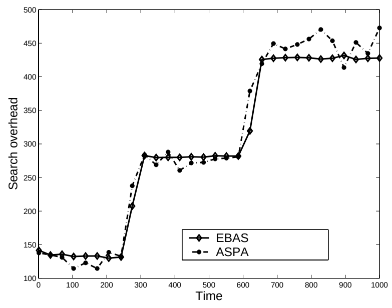


(a) Search overhead

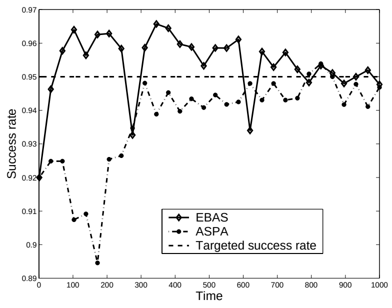


(b) Success rate

Figure 9. Comparison of performance of EBAS with that of ASPA for scenario 1. EBAS maintains success rate above the desired value at the cost of slightly higher overhead than ASPA.



(a) Search overhead



(b) Success rate

Figure 10. Comparison of performance of EBAS with that of ASPA for scenario 2. EBAS maintains the desired success rate while incurring almost same overhead as ASPA.

it changes with time, and (iii) the proposed EBAS algorithm achieves the target level of performance at negligible overhead.

EBAS stores the popularity estimate of all resources in the network. This would require $O(n)$ memory space, where n is the number of resources (or categories of resources) in the network. State-full search mechanisms, such as APS [29], that maintain the rank of neighbors, store tables of size $O(n \cdot d)$ where d is the average degree of a node. Thus the memory requirement of EBAS is less than other search methods.

The overhead incurred in order to maintain the popularity estimates is confined to transfer of popularity estimates to a new node that joins the network. Thus the overhead for arrival of each node is of the order of n. The total overhead cased will depend on the node arrival rate of the network. This overhead will not effect the scalability of the network if the node arrival rate of the network is not very large. More importantly, the overhead affects only the one-hop neighbor.

This method of adaptively choosing the values of parameters of random walk using popularity estimates can be used along with other state-full search mechanisms like APS. This would further decrease the delay and increase the number of hits of adaptive random walk.

The problem of choosing optimal values of k and T may also be modeled in a control theoretic framework. But the response of the P2P network to the parameters of random walk is non-linear, and thus the analysis of the performance using control theory is not straightforward. However the performance could be analyzed by linearizing the response of P2P network around a specific operating point. The application of control theory to analyze the performance of adaptive random walk is the subject of our future work.

References

[1] The Gnutella. `http://www.gnutella.com`.

[2] L. A. Adamic, R. M.Lukose, B. Huberman, and A. R. Puniyani. Search in power-law networks. *Physical Review E*, 64:046135, 2001.

[3] D. Aldous and J. Fill. *Reversible Markov Chains and Random Walk on Graphs*. Manuscript available at: `http://stat-www.berkeley.edu/users/aldous/RWG/book.html`.

[4] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, October 1999.

[5] N. Bisnik and A. Abouzeid. Modeling and analysis of random walk search in P2P networks. In *Proc. of Second International Workshop on Hot Topics in Peer-to-Peer Computing (HOT-P2P'05)*, July 2005.

[6] A. Broder and A. Karlin. Bounds on the cover time. *Journal of Theoretical Probability*, 2:101–120, 1989.

[7] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–509, 1952.

[8] Clip2 DSS. Gnutella: To the bandwidth barrier and beyond. Technical report, Clip2 DSS, November 2000. Available at `http://cs-www.cs.yale.edu/homes/arvind/cs425/doc/gnutella.html`.

[9] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 23. IEEE Computer Society, 2002.

[10] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'00)*, pages 43–56, Stockholm, Sweden, August 2000.

[11] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *Proc. of IEEE INFOCOM*, volume 1, pages 130–140, March 2004.

[12] W. Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58(301):13–30, March 1963.

[13] P. Holme and B. J. Kim. Growing scale-free networks with tunable clustering. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 65:026107, 2002.

[14] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, New York, NY, USA, 1991.

[15] I. Jawahar and J. Wu. A two level random search protocol for peer-to peer networks. In *Proc. of Systemics, Cybernetics and Informatics (SCI'04)*, 2004.

[16] M. A. Jovanovic, F. S. Annexstein, and K. A. Berman. Modeling Peer-To-Peer Network Topologies Through Small World Models and Power Laws. In *Proc. of IX Telecommunication Forum Telfor, Belgrade*. IEEE, Nov 2001.

[17] M. A. Jovanovico, F. S. Annexstein, and K. A. Berman. Scalability issues in large peer-to-peer networks - a case study of gnutella. Technical Report, ECECS Department, University of Cincinnati. Available at `http://www.ececs.uc.edu/~annexste/Papers/scalabilityissues.ps`, 2001.

[18] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *Proc. of Conference on Information and Knowledge Management (CIKM'02)*, pages 300–307, 2002.

[19] K. Kant, R. Iyer, and V. Tewari. A performance model for peer to peer file-sharing services. `http://kkant.ccwebhost.com/download.html`.

[20] K. Klemm and V. M. Eguiluz. Growing scale-free networks with small-world behavior. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 65, 2002.

[21] C. Law and K.-Y. Siu. Distributed construction of random expander networks. In *INFOCOM*, 2003.

[22] T. Lin and H. Wang. Search performance analysis in peer-to-peer networks. In *Proc of Third International Conference on Peer-to-Peer Computing (P2P'03)*, pages 82–82. IEEE, Sept. 2003.

[23] L. Lovasz. Random walk on graphs: A survey. *Combinatorics, Paul Erdos is Eighy*, 2:353–398, 1996.

[24] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proc. of the ACM SIGMETRICS'02*, pages 258–259. ACM Press, 2002.

[25] M. Mihail, C. Papadimitriou, and A. Saberi. On certain connectivity properties of the internet topology. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 28, Washington, DC, USA, 2003. IEEE Computer Society.

[26] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[27] G. On, J. Schmitt, and R. Steinmetz. The effectiveness of realistic replication strategies on quality of availability for peer-to-peer systems. In *Proc. of Third International Conference on Peer-to-Peer Computing (P2P'03)*, pages 57–64. IEEE, Sep 2003.

[28] M. Ripeanu and I. Foster. Mapping the gnutella network. *IEEE Internet Computing Journal*, 6:50–57, 2002.

[29] D. Tsoumakos and N. Roussopoulos. Adaptive probabilistic search for peer-to-peer networks. In *Proc. of Third International Conference on Peer-to-Peer Computing (P2P'03)*, pages 102–109. IEEE, sep 2003.

[30] D. Tsoumakos and N. Roussopoulos. A comparison of peer-to-peer search method. In *International Workshop on the Web and Databases(WebDB)*, pages 61–66, june 2003.

[31] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proc of ICDCS'02*, page 5, 2002.