Budget-Based Self-Optimized Incentive Search in Unstructured P2P Networks*

Yi Hu Min Feng Laxmi N. Bhuyan Vana Kalogeraki
Department of Computer Science and Engineering
University of California at Riverside, CA, U.S.A
{yihu, mfeng, bhuyan, vana}@cs.ucr.edu

***Abstract*—Distributed object search is the primary function of Peer-to-Peer (P2P) file sharing system to locate and transfer the file. The predominant search schemes in unstructured P2P systems have their problems: flooding creates excessive traffic overhead and random walk prolongs search delay. Moreover, both use uniform Time-to-Live (TTL) control for all users, which makes them vulnerable to selfish user attacks, and results in the "free-riding" and "tragedy of the commons" problems. In this paper, we propose a Budget-based Self-optimized Incentive Search (BuSIS) protocol for unstructured P2P file sharing systems, which is robust to and restricts selfish user behaviors. Furthermore, our protocol lowers the search overhead while keeping high hit rate. BuSIS provides differentiated search service for selfish users and ties a user's contribution to its service level. We present the analytical models on expected search performance, associated search cost and the user satisfaction level. Extensive emulations have been conducted at large scale network scenarios to compare performance of BuSIS with flooding and random walk searches with and without selfish user behaviors. The experimental results show that BuSIS always has the lowest search overhead without sacrificing the hit rate. When serving selfish users, flooding and random walk performance degrade dramatically, while BuSIS gracefully keeps the hit rate only with 20% overhead of flooding and 25% of random walk.**

I. Introduction

[1] The unstructured P2P file sharing systems have gained much popularity in recent years for their wide applicability and simplicity; however, flooding based search makes them inefficient in locating files and cannot scale well. In P2P systems like Gnutella [1], which is one of the top-three most popular file sharing systems, a query is broadcast and rebroadcast until the object is found, and a reply message is sent back to the requester following each inverse search path. Or the query is dropped when its TTL reaches zero. Flooding dominates the search for unstructured P2P networks due to its independence of the underlying infrastructure, tolerance to nodes failures, and adaptation to the dynamic network topology. However, flooding search generates a huge amount of traffic. Measurements show that in a Gnutella network with $50,000$ nodes and TTL set to 7, the flooding search generates 330 TB/month. Such huge amount of P2P traffic contributes to the largest portion of Internet traffic, and leads to serious scalability problem [2].

The need of P2P systems to support millions of peers simultaneously has resulted in various enhancements to the poor-scalability of flooding. One well-known example is random walk [2], where the requester sends out K independent search queries to randomly selected K neighbors. Each such query is termed as a Walker, which is associated with a TTL and forwarded to a random neighbor until either a query hits, or its TTL drops to zero. By limiting broadcast to K linear walkers, random walk reduces the exponentially increasing overhead to linear. But it still has redundant traffic because the object hit by one walker is unknown to other walkers and they continue to search until their TTL expire. If K walkers are sequentially issued, the reduced overhead may result in extended search delay to cover the same search region.

A common problem of both flooding and random walk is that their performance degrades sharply in the face of selfish users. The uniform TTL control for all users' searches does not provide differentiation in users' benefits, neither gives incentive to cooperate or contribute. Unfortunately, the search performance depends greatly on peers' cooperative forwarding and sharing; however, most P2P users are autonomous and selfish by nature. Forwarding or replying others' queries will consume their own resources, so they only download files without contributing any to the P2P community, which is named as "free-riding" [3] problem. Hence, most of the file download requests are directed to a small number of altruistic peers, who are willing to share, thus they are easily overloaded. This is another problem called "tragedy of the commons" [4]. Besides selfish users may aggressively repeat issuing the same query while waiting for the search result, such redundant searches cause network congestion. These behaviors greatly break down the search performance of both flooding and random walk. In this work, we assume that peers are selfish, but smart and non-malicious. Being selfish, peers only join the network for searching and downloading files. They leave once they complete the downloading. Being non-malicious means peers will not attack or cheat. While joining the P2P networks, they will obey the search and file sharing protocols to get and provide services. Peers are smart and they behave to get the most satisfaction from the service they can afford.

In this paper, we propose a Budget-based Self-optimized Incentive Search (BuSIS) protocol for unstructured P2P file sharing systems. BuSIS motivates the selfish users to contribute by differentiating search services and tying users' contribution to their search service. Our search is built upon random walk, but tailors the total number of TTL and the

[1]* This research was supported by NSF grant CNS-0509440.

number of walkers K to adjust the search performance and the associated overhead. Specifically, the TTL of a query is derived from the budget paid by its requester. The choice of budget maximizes the requester's satisfaction and balances the expected search performance with the associated overhead. The performance is represented by hit rate, hit delay and search reliability, and the search cost reflects the overhead. To improve the search performance they can enjoy, the selfish users are stimulated to earn more credits by providing services to others. As a result the average user online time is prolonged and the overlay network is more stable and reliable. Longer average online time and more tendencies to contribute also relieve the problem that the file host may go offline before the requester starts download or relay peers go offline before the search result is sent back. Thus, the search success rate perceived by end users is increased. Furthermore, the need of "PING-PONG " messages, which is used to probe the existence of neighbors and connect new neighbors, is greatly reduced, which originally occupies 55% of traffic in Gnutella network [5]. Therefore, the network maintenance overhead is lowered. In addition, BuSIS captures the subtle dynamics of query popularity and network conditions to adjust the query budget and moderate the search overhead. For example, searching rarer object needs more budget to increase the hit rate; when the object becomes popular, its search budget drops to reduce the search hops and avoid extra overhead. When the network overload is heavy, per hop search budget grows to constrain the query injection and help relieve network congestion. When the network overload gets lighter, the cost for each hop is cut down, so that the same amount of budget affords more query searches.

Most previous incentive mechanisms for P2P networks address file download [6] [7] [8] given the file providers. We should note that designing an optimal incentive mechanism is unrealistic, because it requires the complete information on every user's affordability and utility function, personalized solutions for heterogeneous users, and secure implementations of the central bank [6]. In large P2P systems, none of the above requirements can be satisfied. For file searching in unstructured P2P networks, the unknown file host makes the incentive search design more challenging. When initiating a query search, the requester hardly can predict the TTL large enough to find the file, or decide which path to follow and how much costs to search at each hop. Thus, choosing a proper budget for the query search with an unknown service provider makes our BuSIS design distinct from previous pricing solutions that require the service providers to be specified. BuSIS is fully autonomous and distributed, no additional infrastructure or software is required. Moreover, it personalizes the query initialization and forward schemes for heterogeneous peers based on their local estimations and preferences.

Our paper makes the following major contributions:

1) Design a Budget-based Self-optimized Incentive Search (BuSIS) protocol for unstructured P2P file sharing systems, which is resilient to selfish user behaviors and greatly reduces the overhead of uniform TTL-controlled search.
2) Present probabilistic models to estimate the expected search performance by learning the relevant network information.
3) Develop a BuSIS-emulator middleware and conduct extensive emulations to verify the adaptability and self-optimization capability of BuSIS with comparisons of random walk and flooding search techniques.

The rest of the paper is organized as follows. Sec.II discusses related work. Sec.III presents the BuSIS protocol. Sec.IV gives the detail derivations and the calculations for the parameters used in the BuSIS. Sec.V describes our evaluation methodology and presents the experimental results. We conclude the work in Sec.VI.

II. Related Work

The excessive overhead of flooding-based search algorithms has been improved from three aspects: 1) by changing the search behavior and limiting the flooding degree, like random walker [2] [9] reduces traffic overhead from exponential increase to linear increase; 2) by exploring the cache information like DiCAS [10] or the congestion information like Congestion-aware Search [11] to restrict the search scope and cut down the overhead; 3) by matching the physical network topology with the overlay topology like LATM [12] to trim the redundant traffic in the underlying physical links. Random walk gains popularity by its simplicity and low overhead, while this comes at the cost of easy convergence (rare objects in low connectivity nodes are harder to be found), low reliability (single node crash leads to search failure) and high delay (proportional to the search length) due to its linear search. Hence, hybrid methods of flooding and random walk (e.g. Percolation Search [13] and BubbleStorm [14]) are proposed to take advantages of strength of each method. Also an adaptive random walk search based on object popularity is presented in [15].

BuSIS adopts the simple and lightweight random walk search as basis, the hybrid of one-step flooding is an option here, because of the in-negligible overhead of exchanging and updating the sharing folders between neighbors, the detail explanation is given in Sec.V. On top of this, BuSIS provides differentiated search as an incentive to selfish users, which are not concerned by the above schemes and makes them vulnerable to selfish user behaviors. Three major techniques are applied to incentive network protocol design: economic based "pay to service" method (*e.g.* [6] [16] [17] [18] [19]), reputation based method (*e.g.* [20] [21] [22] [23]), and encryption based method (*e.g.* [24]). BuSIS follows "pay to service" method to provide incentive query search in unstructured P2P networks. All economic-based methods require a pricing scheme to address how and how much service consumers pay service producers. There are three major pricing schemes: auction based game theory approach (*e.g.* [25] [17] [7]), price-demand function based approach (*e.g.* [26]), and control based pricing (*e.g.* [8]). Although some are theoretically desirable, like [7] is proved

to be strategy-proof, reach Nash equilibrium, and Pareto-optimal social welfare, their assumptions are unrealistic and infeasible to be applied in query search for unstructured P2P networks. For example, auction methods require to globally collect bidding and utility information from each node, which is hard to achieve in unstructured P2P network consisting of millions autonomous users. The idealized linear demand-price function and the total freedom for the source to determine the traffic demand cannot match our query search, where the requester does not know the service provider (file host). And to use control-based pricing, each consumer needs to probe every candidate service provider periodically, which cannot be met by our search with unknown file host.

A recent work [27] address the incentive query propagation in unstructured P2P systems, but they assume every node is identical and behave the same, which is a fatal problem for real P2P systems, where nodes are highly heterogeneous and autonomous, each behaves to maximize its own interests. While our BuSIS tailors search initialization to each user locally.

III. Budget-Based Self-Optimized Incentive Search Protocol

BuSIS consists of four components: Search Performance Estimation (SPE), Budget Assignment (BA), Query Forwarding (QF) and Parameter Maintenance (PM). BuSIS locates between application layer and transport layer and makes all its optimizations transparent to end uses. It can integrate with any transport layer protocol, for example, our emulator builds BuSIS on top of TCP to run in the Internet. To request a search, the requester peer first runs the SPE module to estimate search performance for a certain set of operation parameters (total TTL, and the number of simultaneous walkers). Then the requester runs the BA module to choose the most beneficial pair of operation parameters, and assign budget on the query walkers. The QF module is run by a peer to send out a query after assigning budget and to relay others' query when the query does not hit locally. A peer runs the PM module to update the input parameters of the PE, BA and QF modules. The update is performed whenever the peer sends out a query, receives a query and receives a query feedback. The feedback message is sent along the reverse search path after a query hits.

BuSIS models the unstructured P2P file sharing system as a *competitive market* [28], where each peer acts as a consumer or a producer. The major services provided in the P2P file sharing market include object search and file transfer, and BuSIS focuses on object search service. A peer takes the role of a consumer when initiating a search query, and it pays the search service by associating an amount of budget on the query. When a peer receives other's query, it plays as a search service producer; it processes the query and deducts its charges from the query budget. If it holds the query object, it replies to the requester by a query feedback message, otherwise, it relays the query to another neighbor for further search. In BuSIS, each peer charges the same for both query hit and query forwarding in terms of providing search service. After receiving the query feedback, the file host charges a larger amount of credits for the real file transfer. The credit is the currency in our P2P market and in this paper, we assume no fraud in currency implementation. The techniques for secure currency implementation are discussed in [16] [19] [29].

Initially a peer enters the market without any credit, it gets the *participation payment* from the system for each online time-slot. This is regardless of whether this peer shares objects or forwards queries. After establishing connections with neighbor peers, it receives others' queries, processes and charges them. If the query matches the local sharing object, it replies a feedback message to the query requester. On receiving a feedback message, each relay peer along the reverse search path runs the PM module to update the related input parameters. If the peer does not have the object, it runs the QF module to forward the query to another neighbor. In both cases this peer charges the query the same price. In case the query lacks budget, if the peer holds the object it still charges and sends back the feedback message, indicating the amount the requester needs to pay later. If the peer does not have the object, it stops forwarding and drops the query with insufficient budget.

To search an object, the peer first runs the SPE module to estimate the expected search performances for all sets of feasible operation parameters, which are then used by the BA module to choose the most beneficial one and set the query budget to be the associated search cost. To issue a search query, the requester peer should wait until it earns enough credits to prepay the budget determined by the BA module. After accumulating enough credits, the peer runs probabilistic forwarding in the QF module to send out each walker of the query search and deducts the budget from its credits account. If the query hits and the requester successfully receives the feedback message, it directly contacts the file host to perform file transfer.

Fig.1 shows the basic operations and data flowchart of the BuSIS protocol. The equations cited in the flowchart are used to compute various cost functions based on which the decisions are made. For better readability, we describe the protocol operations in this section and derive the equations in Sec.IV. The detailed derivations may be skipped without affecting the flow of the paper. In the following, we elaborate the four modules in detail.

A. Search Performance Estimation (SPE)

Given certain operation parameters, the SPE module estimates the expected search performance. BuSIS is built on top of the random walk search, where the requester sends out several walkers. Each walker is assigned a TTL and randomly searches along a path until a hit occurs or its TTL reaches zero. SPE module uses the total TTL T_q and the number of simultaneous walkers x_q as the operation parameters to estimate the expected hit rate $prob_q$, hit delay del_q and search reliability R_q, assuming each walker equally shares the total

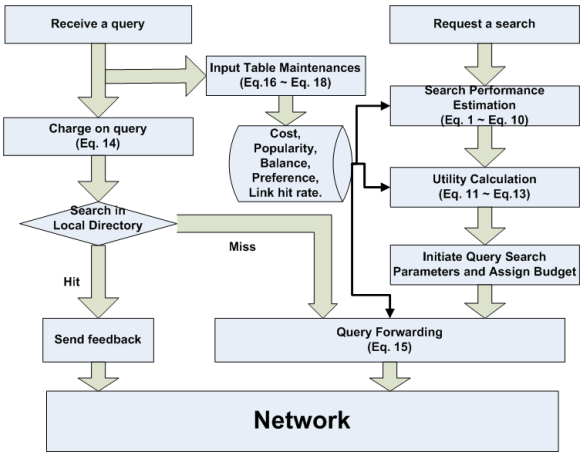

Fig. 1. BuSIS Flowchart

TTL. For better readability, we put the detail derivation steps in Sec.IV-A.

B. Budget Assignment (BA)

Instead of setting uniform TTL, BuSIS uses the budget to control the number of search hops for each walker. The requester peers prepays an amount of budget for a query search, which is the associated search cost for an expected search performance. Clearly, low budget limits the search scope and can result in a low hit probability, while surplus budget wastes resources and causes unnecessary traffic overhead. The total budget per query can also be split among the multiple simultaneous walkers to shorten the search delay, while the number of simultaneous walkers cannot exceed the node connection degree. In case of query hit, the remaining budget is saved and returned to the requester. But for multiple walkers, even if one of them hits, the other ones will still use up their budget. The BA module determines the budget amount and the simultaneous walkers number with the aim to maximize the search requester's interests, represented by the utility function. The cost and utility functions to determine the budget assignment are derived in Sec.IV-B for better readability.

A selfish peer is modeled as a utility maximizer, making all choices toward maximizing its search utility. Since the specific form of utility function is essentially unknown, we aim to formulate it in a simple form and properly reflects the impacts of relevant factors.

The requester i scans all feasible pairs of total TTL T and the number of walkers x, calculates the resulting utility from each pair, and chooses the pair (T, x) with the maximum utility as Eq.12 in Sec.IV-B. The utility goes up with better performance and gets lowered by larger search cost, and the maximum utility balances search performance with the cost, which automatically constrains the total search overhead. This makes bound the total TTL explicitly unnecessary. But without any constraints, the peer will initialize queries once they earn a little credits. The query budget is subject to the amount of credits this peer currently has, even under the utility function control, issuing queries with insufficient credits leads to poor search performance and large overhead. Therefore, the utility function requires that the estimated hit probability should be no worse than the acceptable hit probability threshold P_{thre}. Furthermore, to adapt to time sensitive applications, the estimated hit delay should be no worse than the delay threshold D_{thre}.

After selecting the pair (T, x) with maximal utility, the requester i waits to earn enough credits to prepay the associated cost $C_i(T, x)$ as the budget, then sends out x simultaneous walkers with equal share of budget.

C. Query Forwarding (QF)

The QF is run by a peer for two cases, one is for forwarding other's query because no local objects match, and the other is for sending out the query initiated by itself. The forwarder adopts the probabilistic forwarding assisted by the knowledge of the object popularity, outgoing link quality and network condition. The goal of QF is to increase the query hit probability for future search under its budget constraint.

First, if the query is other's, the local peer i checks the remaining budget of the query and computes its search price X^i_{srch}. The query is dropped if the remaining budget cannot afford X^i_{srch}. The search price is determined by the link quality, the more neighbor connections the more choices it has to forward the query; the shorter queuing delay the faster the query will move forward and the lower risk being dropped due to buffer overflow. Better link quality deserves higher charge. The search price calculation is given in Sec.IV-C. We allow each peer to locally compute its price, assuming the peers are honest. To prevent fraud in price calculation, some security techniques can be applied, like having the system administrator conduct random anonymous sampling to examine the price.

The peer i takes inputs as the object popularity pop^x_i of the query object x, and the outgoing link hit rate $hit_{i,k}$ to calculate Pr^x_k, the forward probability to neighbor k for searching object x. The calculation of Pr^x_k is given in Sec.IV-C.

When the object holder is unknown, probabilistic forwarding gives each outgoing link some chances to find the object. For a popular object, an existing good path is preferred, so the forward probability is biased on high hit rate link. For a rarer object, uniform random forwarding is chosen to explore potential good path, so the forward probability equally distributes to each link. Since biased on high hit rate links converges the search path to those "hot peers", which have high connectivity and can easily be visited, and restricts the search scope. This will make some low connectivity peers rarely be visited.

Finally, peer i charges this hop cost X^i_{srch} from the query budget, updates the query remaining budget, and forwards the query.

D. Parameter Maintenance (PM)

The PM module maintains five input parameters used by the SPE, BA, QF modules, which are per hop search cost, object popularity, link hit rate, credit account balance, and

performance threshold. The first three are periodically updated. We consider a discrete time domain, where time is slotted and one time slot equals a update duration. The estimated value of these three parameters are weighted averages of the previous estimation and the most recent estimation. We use α to denote the weight of the previous estimation, where $0 < \alpha < 1$. Whenever a peer receives a query, or a feedback message, it examines the packet header and updates the most recent estimation of the related parameters. No extra communication is required to perform the parameter maintenance. The exact update equations are given in Sec.IV-D.

IV. BuSIS Analytical Model

A. SPE Derivations

Following some analytical expressions in [15], this section gives detail derivations of the search performance estimation made in the SPE module.

Hit rate: Given a feasible operation parameter pair: total number T_q of hops and number of simultaneous walkers x_q, we use a uniform random distribution model. Assuming p is the probability that a peer holds the target object (p is the expected popularity of the object). Eq.1 is the expression of the expected hit probability $prob_q$.

$$prob_q = 1 - (1-p)^{T_q} \tag{1}$$

Hit Delay: We measure the expected hit delay del_q by the number of hops before the walker terminates, which we denote as D, and assume that ρ is the average per hop delay. $E[D]$ is the expected value of D, and for a single random walker, the termination probability at each hop is defined in Eq.2, since the walker terminates either by a query hit or exhausting T_q hops. Thus the expected hit delay for a single random walker is defined in Eq.4, and $E[D]$ is defined in Eq.3.

$$P(D=j) = \begin{cases} p*(1-p)^{j-1} & 1 \le j < T_q \\ (1-p)^{j-1} & j = T_q \end{cases} \tag{2}$$

$$E[D] = \Sigma_{j=1}^{T_q} \quad j * P(D=j) \tag{3}$$

$$del_q = E[D] * \rho \tag{4}$$

In the case of x_q simultaneous random walkers, and each with $\frac{T_q}{x_q}$ hops for searching, the expected delay of each walker $E[D_x]$ is defined in Eq.6. And the expected hit delay for simultaneous x walkers is defined in Eq.7.

$$P(D_x=j) = \begin{cases} (1-(1-p)^x)*(1-p)^{x(j-1)} & 1 \le j < \frac{T_q}{x_q} \\ (1-p)^{x(j-1)} & j = \frac{T_q}{x_q} \end{cases} \tag{5}$$

$$E[D_x] = \Sigma_{j=1}^{\frac{T_q}{x_q}} \quad j * P(D_x=j) \tag{6}$$

$$del_q = E[D_x] * \rho \tag{7}$$

Search Reliability: Since the query hit reply propagates back to the requester through the reverse search path, any relay peer on the path may disconnect after forwarding the query but before the reply is back. So we model the reliability for a query hit path, which reflects the possibility that the query hit result may never reach the requester and should be considered in the requester's utility function. Let t_{on} be the average peer online time slots, and $E[D_x]$ be the number of hops the query hit away from the requester. t_{on} can be set locally by observing neighboring peers average online time. For easy notation, set y equal to $E[D_x]$ and relay peers are ordered as $p_1, p_2 \dots p_y$ based on their hops from the requester. We define the holding period for each relay peer p_j $(1 \le j \le y)$ in Eq.8 as the number of time slots it should wait for the hit reply after it forwards the query.

$$P_{hold}^j = 2 * (y-j) \tag{8}$$

Then the probability for p_j $(1 \le j \le y)$ keeping online during its holding period is given in Eq.9.

$$p_{on}^j = (1 - \frac{1}{t_{on}})^{P_{hold}^j} \tag{9}$$

Thus, the expected search reliability R_q associated with (T_q, x_q) is the probability that all relay peers remain online during the whole query hit reply period, which is given in Eq.10.

$$R_q = \prod_{j=1}^{y} \quad p_{on}^j \tag{10}$$

B. BA Calculations

Search cost: Let $C_q^i(T_q, x_q)$ denote expected cost for peer i to conduct a search with T_q total hops and x_q simultaneous walkers. Let C_{hop}^i be the estimated per hop cost at peer i. The expected total cost $C_q^i(T_q, x_q)$ pays the estimated hops before each independent walker terminates and is defined in Eq.11.

$$C_q^i(T_q, x_q) = x * C_{hop}^i * \{\Sigma_{j=1}^{\frac{T_q}{x_q}} \quad j * P(D=j)\} \tag{11}$$

Search requester's utility: The utility function, defined in Eq.13, takes the total hop counts T_q and the number of simultaneous walkers x_q as input and follows the derivations in Sec.IV-A to get the expected hit probability $prob_q$, hit delay del_q, search reliability R_q and search cost $C_q^i(T_q, x_q)$.

$$(T, x) = arg \ \max \ U_i(T_q, x_q) \ s.t. \ prob_q \ge P_{thre}, \ del_q \le D_{thre} \tag{12}$$

$$U_i(T_q, x_q) = \frac{prob_q * R_q}{C_q^i(T_q, x_q)} \tag{13}$$

In Eq.13, a weight can be added to the estimated hit probability as an coefficient to scale the impacts of hit probability and search reliability. In this work, we treat them equally.

C. QF Calculations

Search price calculation: The search price X_{srch}^i of peer i is defined in Eq.14, where $\mathcal{N}_i$ is the connection degree of peer i, L_i and $\mathcal{Q}_i$ are the average link and queuing delay of peer i, which consist of one hop delay. ω_{hpd} is the weights on one hop delay, which tunes the connection degree and per hop delay to be comparable. In our experiment, we set ω_{hpd} based

on the relationship between node average connection degree and estimated per hop delay.

$$X^i_{srch} = \mathcal{N}_i + \frac{\omega_{hpd}}{(L_i + \mathcal{Q}_i)} \quad (14)$$

Forward probability of an outgoing link: The forward probability to neighbor k for searching object x: Pr^x_k is defined in Eq.15, where S_i denotes the neighbor set of peer i, $hit_{i,k}$ denotes the hit rate of outgoing link (i,k), which is the ratio between the number of query hit from this link and the total number of queries forwarded through this link. pop^x_i is the query object popularity observed by peer i.

$$Pr^x_k = \frac{hit^{pop^x_i}_{i,k}}{\sum_{q \in S_i} hit_{i,q}{}^{pop^x_i}} \quad (15)$$

D. PE Calculations

Parameter Update:

1) Per Hop Cost
Let $C^{hop}_i(t)$ denote the estimation in the t^{th} time slot, α denote the weight of previous estimation, $C^i_{total}(t)$ denote the total search cost the peer i counts during the t^{th} time slot, and $N^i_{hop}(t)$ denote the total search hops paid by $C^i_{total}(t)$. $C^i_{total}(t)$ and $N^i_{hop}(t)$ are computed by examining the query packets passing through peer i, and the query results replying to peer i during the t^{th} time slot. The updating is performed as in Eq.16.

$$C^i_{hop}(t) = \alpha * C^i_{hop}(t-1) + (1-\alpha) * \frac{C^i_{total}(t)}{N^i_{hop}(t)} \quad (16)$$

2) Object Popularity
The estimated popularity of the object x is derived from the average hit rate. Let $q^x_i(t)$ denote the estimated popularity in the t^{th} time slot, and $pop^x_i(t)$ denote the weighted average popularity until the t^{th} time slot. The estimated popularity is updated as Eq.17.

$$pop^x_i(t) = \alpha * pop^x_i(t-1) + (1-\alpha) * q^x_i(t) \quad (17)$$

3) Link Hit Rate
The estimated hit rate of an outgoing link is the ratio of total number of queries forwarded through that link to total number of feedback messages received through that link, the final result is averaged over time. For each neighbor j, let $hit_{i,j}(t)$ denote the weighted average hit rate examined by peer i up to the t^{th} time slot for outgoing link (i,j), $h_{i,j}(t)$ denote the estimated hit rate in the t^{th} time slot. The link hit rate is updated as Eq.18.

$$hit_{i,j}(t) = \alpha * hit_{i,j}(t-1) + (1-\alpha) * h_{i,j}(t) \quad (18)$$

4) Credit Account Balance
C_i denotes the total available credits at peer i, which is updated after each time peer i issues a search, forwards and replies others' queries. For security reason, some authentication and forge-proof mechanism must be used to manage the peer credit account. In our protocol, we adopt the simple receipt method and assume that each peer is honest in maintaining the credit account.

5) Performance Threshold
Performance threshold includes the thresholds of hit probability P_{thre} and hit delayD_{thre}. These constants are set once at initialization to prune out infeasible operation parameters.

V. Performance Evaluations

In this section, we develop a custom emulator to evaluate the performance of BuSIS, and compare with random walk and flooding search techniques. We introduce our experimental methodology in Sec.V-A and explain the network figuration in Sec.V-B. Then, in Sec.V-C, we analyze the experimental results, elaborate the implications of the figures and discuss some important performance issues.

A. Experimental Methodology

It is infeasible to configure a large number of peers in Gnutella network with BuSIS to evaluate the performance. Thus, we develop a BuSIS-emulator middleware, which can be deployed in a large distributed system to form a BuSIS-based unstructured P2P network. Our BuSIS-emulator creates multiple independent processes; each process representing a single peer and implementing BuSIS middleware on top of TCP/IP protocol to run in the Internet. A peer is emulated for joining and leaving the network, establishing and maintaining connections with neighbor peers, generating queries, processing and replying others' queries, and updating sharing directories to simulate file transfers. Current emulations are based on deployment in the LAN in our computer science department to simulate the performance with 1000 nodes. Our emulations aim to verify:

1) the adaptivity and self-optimization capability of BuSIS in the face of changing object popularity;
2) the effectiveness of BuSIS under selfish and selfless user behavior;
3) the positive impact of BuSIS on smoothing the query injection in case of traffic burst;
4) the effectiveness of BuSIS for heterogeneous peers topologies.

We design different emulation scenarios to fulfill the above four objectives, and the results are presented in the following sections.

BuSIS strives for both better user-perceived search quality and system efficiency, so we focus on two metrics: the *hit rate* and the *overhead-per-query*. Hit rate is defined as the number of query hits over the total number of issued queries. And the overhead-per-query is defined as the total search overhead (total number of hops queries traversed) divided by the total number of issued queries.

B. Network Configuration

There are total five parameters need to be set up, which are summarized at Table.I. Their setup values is based on their functionalities we explained when introduced them.

When a new peer joins, it first tries its cache, which contains addresses of previous neighbor peers and peers who replied

TABLE I
SUMMARY OF PARAMETERS SETUP

Symbol	Definition	Value
P_{thre}	hit probability threshold	0.5
D_{thre}	hit delay threshold	10s
ω_{hpd}	weight of per hop delay in search price calculation	0.1
α	weight of previous estimation for periodic updates	0.8

its queries before, to establish neighbor connections. If those cached peers are not active, the new peer asks a well known node (landmark) for a list of candidate neighbors. We bound the maximum connection degree of each node to x_{max}. Then the landmark node sends back a list of x_{max} recent active peers. The new node connects to neighbors from this list. When a peer's connection degree becomes smaller than half of x_{max}, it re-sends request message to the landmark node for more candidate neighbors. In emulation we set the maximum node degree to 5, based on the average Gnutella node degree of 3 to 5.

P2P networks are highly dynamic with peers that go on-line and off-line frequently. It has been shown that 20% of the P2P connections last no more than 1 minute, and around 60% of peers keep on-line no more than 10 minutes each time after they enter the system [30]. We emulate the dynamics of P2P networks by assigning each peer a lifetime, which is the time period the peer will be online in the system. The lifetime follows the distribution observed in [31], with the mean value to be 10 minutes [30]. During each second, a number of peers leave the network with there lifetime reduced to zero. We let the same number of new peers enter the network to keep the network size as 5000 peers. The query generation by a peer follows a poisson process with inter generation time of 5 seconds. The number of replicas of any object ranges from 10 to 200, all of which are randomly spread in the network. Each peer queries objects which are locally unavailable with the query object popularity simulated as Zipf distribution [32].

For TTL setting, BuSIS initiates a query search by determining the number of simultaneous walkers and allocating a budget to each walker. The actual TTL of each walker is determined by the assigned budget and the charges of relay peers. To fairly compare performance with flooding and random walk, we set their TTL such that the total search hop counts remain the same, and use this as an upper bound for BuSIS. In our emulation we set random walk TTL to be 25 given the average node connection degree is 5, then TTL for flooding is set to be 2. For multiple walkers in random walk, the total TTL can be split equally among the walkers. We also bound the maximum TTL for our incentive protocol to 25 for fairness.

Another important parameter is *participation payment*, whose setting is subtle and impacts the system functionality. Small participation payment leads to long warm up period of low system throughput due to shortage of credits for activating query searches. A new-join peer has no credit to issue query until earning enough participation payment. On the other hand, large participation payment will make peers ignore the earning from processing others' queries. It is because staying online earns enough credit to afford good search service. This undesirably stimulates the peers only to be online but not contribute and serve others. Besides, large participation payment causes currency inflation as the total credits in the system grows unbound. We test the system throughput by varying the participation payment, the results is shown in Fig.2 based on which we set participation payment to be 5 for our emulation. Because this value is large enough to achieve good system throughput, but small enough to not generate query injection burst due to surplus credits.

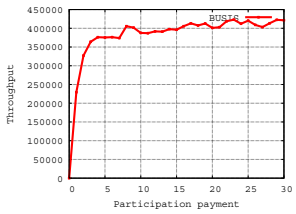

Fig. 2. System throughput changes over participation payment

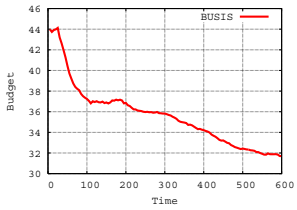

Fig. 3. Budget adaptive trend over system operation time

C. Experimental Results

1) Performance Impact of Budget Assignment: In this emulation we examine the adaptivity and self-optimization capability of BuSIS in the face of changing object popularity. We emulate the object popularity changes by letting each peer perform searches and provide the search result as a replica to share with others. Initially each node holds 10% of total object, that is, an average object popularity is 0.1, and the object popularity increases as the system runs thus making the object search easier. The experimental results show that the average query budget decreases from 44 to 32 by 25% as shown in Fig.3, while the hit rate increases from 40% to 70% in Fig.4. The overhead per query is moderate from 6.5 hops to 5.5 hops as in Fig.5. The reason for such changes is that initially each peer does not have any knowledge on the input parameters, it takes the warm-up period to learn and update its estimation. Then assisted by the updated estimation, the utility function helps the peers to conduct the most beneficial search with high hit rate and low overhead. The self-optimizing and adaptability of BuSIS makes the search sensitive to the object popularity changes and always keep good performance with low overhead.

2) Performance Impact of User Behaviors: In this emulation we model selfless and selfish users by assigning different lifetimes, and compare the performance of BuSIS with that of Random walk and flooding. The selfless users keep on-line after they join the network, while the selfish users only join for searching file objects and leave once they get the objects. We first apply the selfless user behavior model, and compare the performance for the three search methods under fair TTL settings while adjusting the number of objects available per node. As shown in Fig.**??**, the hit rate of the three methods are similar, but the per query overhead sharply differs in Fig.7. Overhead is defined as the average number of hops used

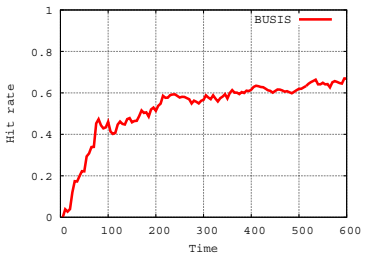

Fig. 4. Hit rate changing trend over system operation time

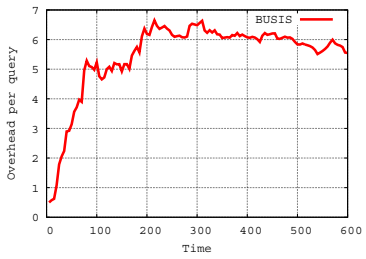

Fig. 5. Overhead per query changing trend over system operation time

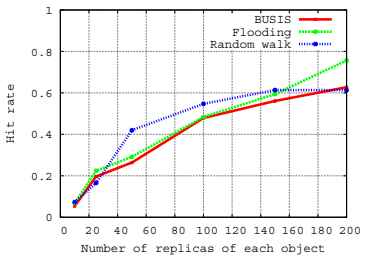

Fig. 6. Hit rate under selfless user behavior

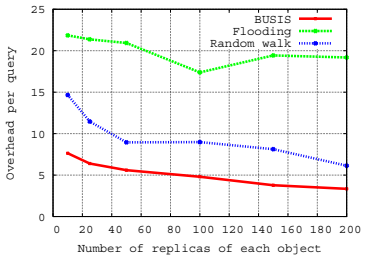

Fig. 7. Overhead per query under selfless user behavior

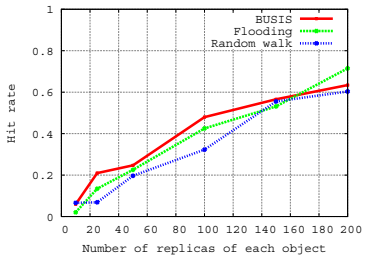

Fig. 8. Hit rate under selfish user behavior

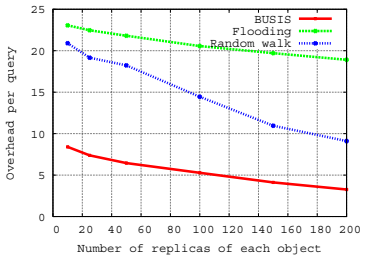

Fig. 9. Overhead per query under selfish user behavior

for each query. Flooding has the heaviest overhead, random walker has much less overhead than flooding and BuSIS has the lowest overhead, averaging around 25% of flooding and 50% of random walk. Especially when each user only holds a small percentage of the total objects (like 1%), BuSIS has remarkably reduced overhead while achieving the same hit rate. This scenario is common to the P2P file sharing systems, where each user only has a small number of files and needs to collaborate with others to get mutual benefits. The overhead reduction in BuSIS results from the self-optimization in the utility calculation, which optimizes the query budget to gracefully control the search scope according to the estimated object popularity.

When we apply the selfish user behavior model, BuSIS outperforms flooding and random walk by a larger amount than that in selfless user behavior model. As in Fig.8, the hit rate of BuSIS in most cases is higher than that of flooding and random walk. More importantly, comparing their results in Fig.9 with Fig.7 selfish behavior leads to dramatic overhead increase for both flooding and random walk. In contrast, BuSIS keeps low overhead; especially when each node only has a small percentage of objects, our overhead is only 20% of flooding and 25% of random walk. This demonstrates that BuSIS always reduces the search overhead without hurting the hit rate, and is especially resilient to selfish user behavior. Flooding always has the largest overhead no matter whether users are selfless or selfish. Random walk has moderate overhead when users are selfless, but sharply increased overhead in case of selfish users, which is the common situation in real P2P file sharing systems.

3) Performance Impact of Dynamic Network Overload: In this emulation we examine the effectiveness of BuSIS in regulating user's behavior of query injection and moderating the query traffic load. We compare the query injection load by BuSIS and random walk under the same user demands, and the results are shown in Fig.10. To simulate user demands, after the warm up period at time $t = 300$ we put 100 queries at each peer's outgoing queue. The random walk produces the traffic burst around $t = 330$ due to the queuing delay and network conditions. In contrast, BuSIS does not generate traffic burst, the most intense traffic is around $1/3$rd of random walk's. This smoothed traffic resulted from the query budget constraint in BuSIS. The peers are restricted from aggressively issuing queries, because they should wait until they earn enough credits to pay for the budget as decided by the utility function. So the traffic load is moderated over time by BuSIS.

4) Performance Impact of Heterogeneous Peers: All previous simulations only consider homogeneous peers, that is each peer on an average holds the same number of objects and has the same number of neighbor connections. Here we examine the heterogeneous peers (which is practical for the P2P systems), where each end user is of different bandwidth and processing capacity, and has different number of file objects. We create two kinds of nodes: normal nodes and super nodes. Each normal node holds 10 objects and has around 4 to 5 neighbors, while a super node has 100 objects and 50 neighbors. We vary the ratio of normal nodes over super nodes and show the results of hit rate and overhead per query in Fig.11 and Fig.12 respectively. When the ratio of regular nodes over super nodes is small, random walk has higher hit rate than BuSIS but with much higher overhead, BuSIS reaches $2/3$ hit rate of random walk with only $1/2$ the overhead. This is because random walk search path easily converges at the super nodes, who aggregate the majority of the objects and

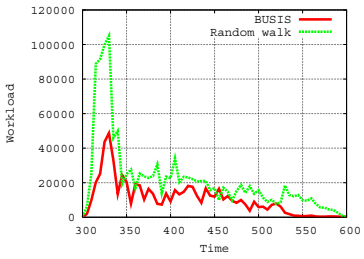

Fig. 10. Workload changing over system operation time

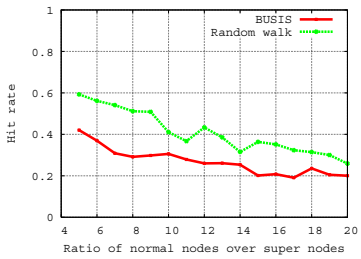

Fig. 11. Hit rate of heterogeneous peers topology

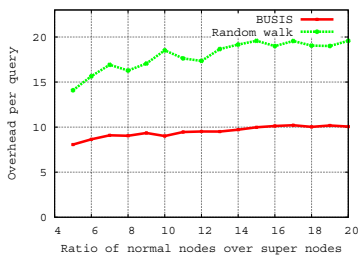

Fig. 12. Overhead per query of heterogeneous peers topology

make the common object search easier. However, this gain in performance requires a large portion of super nodes, around 20%, which is not the case in practice. When the portion of super nodes decreases, the hit rate of BuSIS catches up with that of random walk, while the overhead always stays at only half of the random walk. As shown in Fig.11 and Fig.12, when the percentage of super-nodes is around 5%, the hit rate of BuSIS is almost the same as random walk, but only with 50% overhead.

5) Discussions: BuSIS was designed to augment the random walk search with one-step implicit flooding by replicating neighbors sharing directories. This hybrid technique increases the hit rate of searching and also provides a greater chance for the peer's objects to be hit by others' queries to earn more credits. But this imposes more communication overhead for exchanging directories with neighbors. Especially, the overhead is heavier for updating the directories while the objects are replicated in more peers due to downloads. So we put this as an option for end users.

VI. Conclusion

In this work, we presented a Budget-based Self-optimized Incentive Search (BuSIS) protocol for unstructured P2P networks, where local optimization is adopted by each requester for query initialization and by each forwarder for query forwarding. Our extensive experimental results demonstrate that BuSIS adapts well to the changing object popularity and query trafic burst. Moreover, BuSIS handles selfish user behavior elegantly without degrading search performance.

References

[1] "Gnutella." [Online]. Available: http://www.rfc-gnutella.sourceforge.net/
[2] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shanker, "Search and replication in unstructured peer-to-peer networks," in *ACM ICS'02*.
[3] E. Adar and B. Huberman, "Free riding on gnutella," *FirstMonday*, vol. 5, no. 10, 2000.
[4] G. Hardin, "The tragedy of the commons," *Science*, vol. 162, 1968.
[5] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the gnutella network: properties of large scale peer-to-peer systems and implications for system design," in *IEEE Internet Computing'02*.
[6] P. Antoniadis, C. Courcoubetis, and R. Mason, "Comparing economic incentives in peer-to-peer networks," *COMPUTER NETWORKS*, vol. 46, no. 1, pp. 1640–50, 2004.
[7] R. T. Ma, S. C. Lee, J. C. Lui, and D. K. Yau, "Incentive and service differentiation in p2p networks: a game theoretic approach," *IEEE/ACM Trans. on Networking*, vol. 14, no. 5, pp. 978–91, 2006.
[8] W. Wang and B. Li, "Market-based self-optimization for autonomic service overlay networks," *IEEE J-SAC*, vol. 23, no. 12, pp. 2320–32, 2005.
[9] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," in *IEEE Infocom'04*.
[10] C. Wang, L. Xiao, Y. Liu, and P. Zheng, "Distributed caching and adaptive search in multilayer p2p networks," in *IEEE ICDCS'04*.
[11] K. Kwong and H. Tsang, "A congestion-aware search protocol for heterogeneous peer-to-peer networks," *The Journal of Supercomputing*, vol. 36, no. 3, pp. 265–82, 2006.
[12] Y. Liu, X. Liu, L. Xiao, L. Ni, and X. Zhang, "Location-aware topology matching in p2p systems," in *IEEE Infocom'04*.
[13] N. Sarshar, P. O. Boykin, and V. P. Roychowdhury, "Percolation search in power law networks: making unstructred peer-to-peer networks scalable," in *IEEE Computer Socienty P2P'04*.
[14] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann, "Bubblestrom: resilient, probabilistic, and exhuastive peer-to-peer search," in *ACM SIGCOMM'07*.
[15] N. Bisnik and A. Abouzeid, "Modeling and analysis of random walk search algorithms in p2p networks," in *2nd IWHTP2PS*, 2005.
[16] D. Figueiredo, J. Shapiro, and D. Towsley, "Incentives to promote availability in peer-to-peer anonymity systems," in *IEEE ICNP'05*.
[17] S. Zhong, L. Li, Y. G. Liu, and Y. R. Yang, "On designing incentive-compatible routing and forwarding protocols in wireless ad hoc networks – an integrated approach using game theoretical and cryptographic techniques," in *ACM MobiCom'05*.
[18] W. Wang, S. Eidenbenz, Y. Wang, and X. Li, "Ours: optimal unicast routing systems in non-cooperative wireless networks," in *ACM MobiCom'06*.
[19] S. Zhong and F. Wu, "On designing collusion-resistent routing schemes for non-cooperative wireless ad hoc networks," in *ACM MobiCom'07*.
[20] S. Marti, T. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *ICMCN'00*.
[21] S. Buchegger and J. LeBoudec, "Confidant protocol: cooperation of nodes - fairness in dynamic ad hoc networks," in *IEEE/ACM Symposium on Mobile Ad Hoc Networking and Computing*, 2002.
[22] "Kazaa peer-to-peer file sharing client." [Online]. Available: http://www.kazaa.com/
[23] "emule peer-to-peer file sharing client." [Online]. Available: http://www.emule-project.net/
[24] S. Sundaramurthy and E. M. Belding-Royer, "The ad-mix protocol for encouraging participation in mobile ad hoc networks," in *IEEE ICNP'03*.
[25] K. Chen and K. Nahrstedt, "ipass: an incentive compatible auction scheme to enable packet forwarding service in manet," in *IEEE ICDCS'04*.
[26] M. Li, E. Kamioka, and S. Yamada, "Pricing to stimulate node cooperation in wireless ad hoc networks," *IEICE TRANS. COMMUN.*, vol. E90-B, no. 7, pp. 1640–50, 2007.
[27] C. Li, B. Yu, and K. Sycara, "An incentive mechanism for message relaying in unstructured peer-to-peer systems," in *IFAAMAS'07*.
[28] H. R. Varian, *Microeconomic Analysis*. New York, U.S.: W. W. Norton Company, 1992.
[29] B. Yang and H. Garcia-Molina, "Ppay: micropayments for peer-to-peer systems," in *10th ACM CCCS*, 2003.
[30] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," in *ACM SIGCOMM Internet Measurement Workshop'02*.
[31] S. Saroiu, P. Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Multimedia Computing and Networking'02*.
[32] L. Breslau, P. Cao, G. P. L. Fan, and S. Shenker, "Web caching and zipf-like distributions: evidence and implications," in *IEEE Infocom'99*.