Collaborative Search in Large-scale Unstructured Peer-to-Peer Networks[*]

Yiming Zhang[1], Dongsheng Li[1], Lei Chen[2], Xicheng Lu[1]
[1]*National Laboratory for Parallel and Distributed Processing, NUDT*
[1]*{ymzhang, dsli, xclu}@nudt.edu.cn*
[2]*Department of Computer Science, Hong Kong University of Science and Technology*
leichen@cse.ust.hk

Abstract

Searching in large-scale unstructured peer-to-peer networks is challenging due to the lack of effective hint information to guide queries. In this paper, we propose POP, a Parallel, cOllaborative and Probabilistic search mechanism, in which query messages are viewed as search units to collaborate with each other and aggregate the distributed hints during the search process. A scheme called Distributed Bloom Filter (DBF) is presented to propagate the hints with a bandwidth-aware manner, in which a node divides the received Bloom filter vector into subvectors and disseminates the fragments to its neighbors according to their bandwidth capacity. The effectiveness of POP is demonstrated through theoretical analysis and extensive simulations.

1. Introduction

Searching in large-scale unstructured peer-to-peer (P2P) networks is a challenging problem. Flooding [1] and Random Walk [2] are two common search techniques in unstructured P2P networks. However, they produce large amounts of redundant messages and consume much bandwidth. This has motivated a lot of studies about bandwidth-efficient search [3-9]. *Biased Random Walks* [3] chooses the most powerful neighbor for query forwarding and caching pointers to content located one hop away; *Expanding Ring* [4] in turn starts multiple scoped flooding searches until all requests are satisfied; etc. All these works can be put into the category of the "blind" search, which is blind to the information about the network.

As opposed to the "blind" search, the "informed" search propagates hint information to guide queries based on Bloom filter [10], which is a space efficient technology for set membership lookups. However, there is a certain rate that the judgment is wrong, which is called false positive given by:

$$P_{err} = \left(1-\left(1-1/m\right)^{kn}\right)^{k} \approx \left(1-e^{-kn/m}\right)^{k} \qquad (1)$$

PlanetP [5] utilizes Bloom filters to represent the keywords of the shared files and propagates the Bloom filters using gossiping. *Attenuated search* [6] is a collection of d Bloom filters with the ith Bloom filter at a node being associated with content published by nodes i hops away. All these proposals can propagate resource information only 1 or 2 hops away for the challenge of maintenance cost and false positive rate.

Recently Kumar [7] proposed *SQR*, a search mechanism that uses *Exponentially Decaying Bloom Filter* (EDBF). In *SQR* a node decays the received BF vector by a uniformly predefined decay factor before propagating to others. The query is forwarded to the neighbor having the maximum amount of information.

However, the hint information far from the host (about 4 hops figured out in [7]) would be too weak to be distinguishable from random noise, and then *SQR* takes little effect in large scale networks. Furthermore, in realistic networks different nodes usually have different degree and bandwidth limitation. It's straightforward that high-degree and low-bandwidth nodes should ask their neighbors to take higher decay factors for them. The uniformly predefined decay factor of EDBF can't adapt to the heterogeneity of realistic P2P networks.

In this paper we propose *POP*, a Parallel, cOllaborative and Probabilistic search mechanism, which realizes efficient search in large-scale P2P networks. The contributions of this paper are listed as follows.

* The work was partially supported by the National Natural Science Foundation of China under Grant No. 60673167 and No. 90412011, and the National Basic Research Program of China (973) under Grant No. 2005CB321801.

(1) We propose the concept of "collaborative search", in which query messages are viewed as search units to collaborate with each other, aggregate the information that may be too weak to take effect separately, and dynamically adjust the search direction (forwarding to nodes with the maximum probability to reach the target) and search intensity (generating more query messages in hopeful direction). To the best of our knowledge, we are the first to propose a search mechanism which enables the collaboration of multiple query messages.

(2) To propagate the hint information with a scalable, bandwidth-aware manner, we present a scheme called Distributed Bloom Filter (DBF), in which a node divides the received Bloom filter vectors and disseminates the fragments to different neighbors according to their bandwidth capacity.

(3) Through theoretical analysis and extensive simulations, we prove that *POP* can effectively propagate the information farther and achieve an order of magnitude reduction in search latency over previous proposals.

The rest of this paper is organized as follows. Section 2 introduces the design of *POP*. The theoretical analysis is presented in section 3, followed by the simulations in section 4. Section 5 concludes this paper.

2. POP design

The key idea of *POP* is to distribute the hint information as far as possible with the bandwidth limitation of different nodes and to aggregate this distributed hints through the collaborative search of multiple parallel query messages, which we will discuss after a brief introduction of *search unit*.

2.1. Search unit

In traditional search mechanisms a query message is passively forwarded all through, with the exception of periodically contacting the query originator to check whether the query has been satisfied, and terminating its search if the answer is "yes". *POP* extends the concept of query message to *search unit*, which has more intelligence with active behaviors and can be viewed as a high level abstraction of query messages. By aggregating the distributed information at different nodes, search units have some basic operations as follows.

Pause, *Resume* and *Terminate*: If a query message is temporarily stored at a node and not forwarded to any neighbors we consider it *pauses*, and when this message is forwarded again we consider it *resumes* its search. If a query is discarded we consider the search unit *terminates* its search.

Move, *Reside* and *Dispatch*: If a query at node A is forwarded to only one neighbor B, we refer to it as a *moving* search unit and consider it *moves* from A to B; if a query at node A is forwarded to more than one neighbor, we refer to it as a *resident* search unit and consider it *resides* at node A and *dispatches* multiple descendent search units, each of which is sent to one neighbor. The purpose of the *resident* search unit is to aggregate the information obtained by its descendants.

Report: A *moving* unit periodically *reports* the information discovered by itself to its ancestor; and a *resident* unit periodically *reports* the aggregate results to its ancestor, as well as to its descendants.

Order: Based on the aggregate result and/or the orders from its ancestor, a *resident* unit may *order* its descendants to *pause*, *resume* or *terminate*, etc.

Almost all operations can be done without any extra cost, with the exception of *Report* and *Order*. More *Report* operations result in more communication cost, obviously, and each *Report* operation requests the *moving* unit to carry its ancestor's address (e.g. the IP address) with it, which may result in some extra cost. Because the *resident* unit has no idea of the current address of its moving descendant, *Order* operation needs to be emulated as a side-effect of the acknowledgement of the descendant's reports, which introduces an extra one-hop delay.

2.2. Distributed Bloom Filter

In this subsection we describe the details of Distributed Bloom Filter (DBF). We begin with the design of 2-dimensional *Neighbor Information Table* (*NIT*), a data structure of DBF for efficient representation of hint information, and then describe the information propagation based on DBF.

2.2.1. 2-dimensional Neighbor Information Table. In DBF, node N with degree d maintains a 2-dimensional Neighbor Information Table (*NIT*) with $d \times c$ elements, each of which is a BF vector. The value of c can be determined by each node according to its link capacity (more capacity results in greater c). Element T_{ij} ($1 \le i \le d, 1 \le j < c$) represents the information received from the ith neighbor and originated by nodes j hops away from N; element T_{ic} ($1 \le i \le d$) represents the information received from the ith neighbor and originated by nodes at least c hops away from N. Besides the *NIT* table, each node maintains a traditional BF vector to represent local information of its own resources.

During the propagation of hint information, every node stores the received information in the corresponding element in its *NIT* table according to 1) the neighbor from which the information comes (the row number) and 2) the number of hops that the information has passed (the column number).

2.2.2. Hint information propagation. DBF allows a node to divide and distribute its information to each neighbor according to their bandwidth capacity, thus effectively propagating the information as far as possible with the bandwidth limitation. We use the following symbols for convenience of description.

BF(x): the Bloom filter vector of resource *x*; *BF(A)*: the Bloom filter vector of resources of *A*.

$T^{A}_{B,j}(1 \le j \le c)$: the BF vector in the *j*th column of *A*'s *NIT* table corresponding to neighbor *B*.

DBF(A,B,j): the update message that node *A* propagates to its neighbor *B*. This message has passed *j* hops before it arrives at node *B* and *j* = 0 means that *A* is the originator of this message.

BL_A: the bandwidth limitation of link capacity of node *A*.

Mask(*i,j*): a binary mask with *j* bits from bit *i* set to 1 and others set to 0, e.g., suppose that the BF vector has 8 bits, then *Mask*(0,3) = 11100000, *Mask*(2,5) = 00111110.

The algorithm is summarized as follows. Let the number of bits in BF vector is *m*. If some objects are added to or deleted from a node (e.g. node *B*), node *B* computes $\delta(BF(B))$ by taking bitwise-*XOR* between the new *BF*(*B*) and the old one. To propagate the update to its neighbors (e.g. node *A*), node *B* first computes the "fair share" of *A*, as defined by s_A. $s_A = m \times BL_A / \sum_{neigh(B)} BL_i$. Due to *A* is the first neighbor of *B*, *B* constructs the update message *DBF*(*B,A,*0) by taking the bitwise-*AND* between $Mask(0, \lceil s_A \rceil)$ and $\delta(BF(B))$. Node *B* propagates *DBF(B,A,*0*)* to *A*, and *A* updates element $T^{A}_{B,1}$ in its *NIT* table by taking bitwise-*XOR* of the old $T^{A}_{B,1}$ and *DBF*(*B,A,*0). Similar updates are propagated to each neighbor of *B* separately. Note that the mask for *j*th neighbor is $Mask(\left\lceil \sum_{i=0}^{i<j} s_i \right\rceil, \lceil s_j \rceil)$.

In order to reduce the bandwidth consumption, the actual propagation is a little different from the above description and uses a periodic cumulative delta updates mechanism. E.g., suppose node *B* has 4 neighbors, namely, *A*, *C*, *D*, and *E*. To update element $T^{E}_{B,2}$ of node *E*, node *B* first takes the union (bitwise-*OR*) of *DBF*(*A,B,*0), *DBF* (*D,B,*0) and *DBF* (*C,B,*0) which are received during the period, then generates *DBF*(*B,E,*1) and sends *DBF*(*B,E,*1) to node *E*.

When a new node (for example, node *F*) joins in the network and becomes a neighbor of node *B*, it can be regarded as a special case of the above update process, in which $\delta(BF(F))$ equals *BF*(*F*), the BF vector of node *F'*s local resources set. The update to its neighbor *B* is *DBF* (*F,D,*0) = *BF*(*F*).

2.3. Query routing

POP realizes efficient routing by aggregating the hint information distributed at different nodes. We will describe the details of *POP* routing mechanism after a brief introduction of the aggregation of the hint information.

2.3.1. *Max* and *Average* aggregates. In DBF, given a query at node *A* for resource *x*, element $T^{A}_{N,j}$ returns the *matching degree* which roughly indicates the probability of finding *x* through neighbor *N* after *j* hops. Let *U* be the BF vector corresponding to resource *x* and let *V* be the element $T^{A}_{N,j}$ in node *A*'s *NIT* table. The *matching degree* is given by:

$$Sim(x, T^{A}_{N,j}) = \sum_{i=1}^{m} (U[i] \times V[i]) / \sum_{i=1}^{m} U[i] \qquad (2)$$

According to formula 2, the matching degree is proportional to the strength of target resource information.

There are two kinds of aggregates in current version of *POP*, namely, *Max* and *Average*. More aggregates [13,14] (such as *MIN* and *MEDIAN*) will be added to *POP* in our future work. *Max* aggregates are the maximum matching degree discovered by some search units; and *Average* aggregates are the average matching degree discovered. *POP* utilizes the global *Max* aggregate, local *Max* aggregate, and *Average* aggregate, which are summarized as follows and introduced in details in the next subsection.

(1) The global *Max* aggregate value roughly denotes the minimum distance to the target resource from all search units and is used to select different search policies accordingly.

(2) The local *Max* aggregate value of a search unit denotes its probability of finding target resource and is used to decide whether it is worthy to be continued.

(3) The *Average* aggregate value is computed by dividing the *Sum* aggregates (the sum of matching degrees of all BF vectors in *NIT* tables examined) by the *Count* aggregates (the number of nodes examined). It roughly denotes the ratio of the gains to the cost of a resident search unit with its descendants, and is used to decide which unit is most efficient when the *Max*

aggregate is dominated by noise at the beginning of the search.

2.3.2. Routing mechanism. There are two kinds of information stored in the *NIT* table, namely, the information strength (number of bits set to 1) and the column number of each element. Consequently the routing mechanism of *POP* should utilize this information. Suppose that a query message arrives at node *A* and its ancestor resides at node *P*. We refer to this query and its ancestor as SU_A and SU_P respectively. Let *MaxSim*(*A*) be the local maximum matching degree that SU_A discovers at node *A*.

The algorithm for routing in *POP* is summarized as follows:

(1) SU_A computes the matching degree of resource *x* and each BF vector in node *A*'s *NIT* table by formula 1, then decides the local maximum matching degree *MaxSim*(*A*) and *reports MaxSim*(*A*) to SU_P.

(2) SU_A chooses a subset of the neighbors satisfying $Sim(x, T^A_{N,j}) \geq \alpha \times MaxSim(P), (1 \leq j \leq c)$, in which *N* is a neighbor of node *A* and α is a fault-tolerant factor and $\alpha \leq 1$. SU_A *dispatches* a descendant to each node in the subset. If the subset is null SU_A pauses at node *A* and dispatches a descendant to the neighbor with nonzero *MaxSim*(*A*). If *MaxSim*(*A*) = 0 SU_A *moves* to a randomly chosen neighbor.

(3) SU_P aggregates the reports from all descendants as well as its direct ancestor to update *MaxSim*(*P*) and *reports* new *MaxSim*(*P*) to them including SU_A. SU_A reexamines the above condition and chooses the subset again. If SU_A has *paused* and the new subset is not null SU_A will *resume* its search by *dispatching* one descendant to each node in the new subset. Otherwise if some nodes to which SU_A *dispatched* descendants are excluded outside the new subset SU_A will *order* these descendants to *pause*. Note that SU_A may carry indirect ancestors' address information for a faster aggregation.

(4) As discussed in section 2.1, besides the matching degree, SU_A also utilizes the column number information of elements in *NIT* tables to assist query routing. Let the column number of the element having *MaxSim*(*A*) be j_{max}, SU_A *dispatches* a descendant to each neighbor *N* satisfying $Sim(x, T^A_{N,j}) \geq Threshold$, ($1 \leq j < j_{\max}$), which chooses the maximum value of *j* satisfying the above condition as its *TTL*. The function of SU_N is to determine whether resource *x* can be accessed after *j* hops through *N*. If *N* has at least one neighbor *V* satisfying $Sim(x, T^N_{V,j'}) \geq Sim(x, T^A_{N,j})$ and $j' < j$, SU_N will *move* (or *dispatch* descendants) to node *V*, which is similar to the case at *A*. Otherwise it indicates that the above assumption of SU_A is wrong and SU_N *terminates* immediately.

The algorithm is formalized in figure 1, in which *type*-1 descendants use the matching degree for query routing as in step 2, and *type*-2 descendants use the column number as in step 4.

```
Procedure ForwardQuery (Node B, Unit SU, Resource x)
if (HasFound(B, x))  { return B; }
if (HasSeenSameSU(B, SU))  {
  SelectAnotherNeighbortoForward(SU); return 0; }
if (HasSeenAnotherSUforSameResource(B, SU)) {
  Terminate(SU); return 0;}
$SU_{parent}$ ← SU->parent;  A ← $SU_{parent}$ ->reside_node;
SU->MaxSim ← 0  // Initiate it to 0.
for each $U \in neighbors(B) \wedge U \neq A$
  for (j=1; j++; j<=c)                 // MN: MaxSim_Node
    if ($Sim(x, T^B_{U,j}) > SU->MaxSim$)  { // MT: MaxSim_TTL
      $SU->MaxSim \leftarrow Sim(x, T^B_{U,j})$ ; MN ← U; MT ← j; }
Report (SU->MaxSim, $SU_{parent}$);
if (SU->MaxSim < $SU_{parent}$ ->MaxSim && SU->type=1) {
  if (SU->MaxSim > 0) {
    GenerateSubSU (B, MN, SU, MT, 1);
    ForwardQuery (MN, MN->SU, x); Pause(SU);
    return 0; } else { // to a randomly chosen neighbor.
    ForwardQuery(Random_N, SU, x); return 0; } }
$CandidateSet \leftarrow \{null\}$ ; // for dispatching descendants.
for each $U \in neighbors(B) \wedge U \neq A$  {
  Temp_TTL ← 0; // to generate type-2 descendants.
  for (j=1;j++;j<=c) {// one to each satisfying them.
    if ($Sim(x, T^B_{U,j}) \geq \alpha \times (ST_{parent} -> MaxSim)$)  {
      GenerateSubSU(B, U, SU, j, 1);
      $CandidateSet \leftarrow CandidateSet \cup \{U\}$ ; }
    else if ( $Sim(x, T^A_{N,j}) \geq Threshold, (1 \leq j < j_{\max})$ ) {
      Temp_TTL ← j; // Choose maximum j as its TTL.}}
  if ((SU->type=1) && (Temp_TTL>0)) {
    GenerateSubSU(B, U, SU, Temp_TTL, 2);
    $CandidateSet \leftarrow CandidateSet \cup \{U\}$ ; } }
if (SU->type==2) {// to neighbors with smaller col. No.
  if ( $\exists j' < TTL, \exists V \in neigh(B)$ , $Sim(x, T^B_{V,j'}) \geq Sim(x, T^A_{B,TTL})$ ){
    GenerateSubSU (B, V, SU, j', 2);
    $CandidateSet \leftarrow CandidateSet \cup \{V\}$ ; } }
  else  {// The assumption is wrong.
    Terminate(SU); return 0; } }
if ( $CandidateSet == \phi$ ) { ReturntoAncestor(A);}
for each $V \in CandidateSet$ { // to all nodes in the subset.
  ForwardQuery (V, V->SU, x); }  return 0;
```

Figure 1. **Algorithm for query routing.**

In *POP* local aggregation eases the burden of the query originator, and periodic and cumulative communication between search units reduces the overall traffic in networks.

As discussed in 2.3.1, the global *Max* aggregate value (which is referred to as *gMAX* in the rest of this paper) roughly denotes the minimum distance to the target resource from all units. Intuitively, there should be some certain threshold Th_1 close to 1, and if *gMAX* > Th_1 we consider that that there are some units very close to the target resource and all units except the ones having *gMAX* can *pause* to reduce search cost. Similarly, there should be another threshold Th_2 close to 0, and if *gMAX* < Th_2 we consider that all units are very far from the target resource, which we will discuss in detail later. If Th_2 < *gMAX* < Th_1 all queries would be routed normally as shown in figure 1. We will give approximate values for Th_1 and Th_2 in analysis in section 3.

To accelerate the search process, at the beginning *POP* enhances search as follows. We refer to a search unit having *m* ancestors as a level-*m* unit, e.g., the query originator is a level-0 unit and its direct descendants are level-1 units. We also refer to the query originator as a root unit and refer to the moving unit as a leaf unit. Let *AvrSim*(*m*, *SU*) denote *Average* aggregate value, which is given in section 2.3.1 and roughly denotes the ratio of the gains to the cost of a level-*m* search unit *SU.* To accelerate the search in the "blind" phase, if *gMax* keeps less than threshold Th_2, *POP* periodically (once per successive *NH* hops) enhances the search of level-*m* units having top *k* maximum values of *AvrSim*(*m*, *SU*). When *gMax* is greater than or equal to Th_2 *POP* will stop this enhancement and order all units with zero local *Max* aggregate values to pause.

3. Analysis

In this section we give the theoretical analysis of *POP*'s search latency based on a partial model of *POP*. We model the underlying network topology as a random graph of *n* vertices, with the degree uniformly distributed in the interval [*a, b*], and the average node degree is $d_a = (a+b)/2$. Let *m* be the size of each Bloom filter and *k* be the number of hash functions, and suppose each node has *l* resources to share, $m >> k \times l$. According to [7], for small *i* the random graph could be viewed as a tree about *i* layers rooted at an arbitrarily chosen node, and then the number of nodes i hops away from an arbitrary node is:

$$n_i \approx d_a \times (d_a - 1)^{i-1} \qquad (3)$$

During *POP*'s search process, the search latency is determined by the error rate of query routing at each node, which will be introduced in section 3.1. The deductions of the formulas in this section are omitted due to lack of space. An expanded version of this paper, including a detailed deduction of all formulas, is presented in our technical report [11].

3.1. Error rate of POP routing

POP may make wrong judgments when choosing the neighbor set *S* for forwarding queries at node *A*. Suppose that neighbor *N* is the correct neighbor. If *S* doesn't include neighbor *N* we think an error occurs.

Suppose a query for resource *x* is to be routed at node *A*. We use *error rate* $P_{err,i}$ to represent the routing efficiency, which is defined as the possibility that at node *A* (*i* hops away from the resource) the chosen neighbor set for forwarding queries doesn't include the correct neighbor (the one on the shortest path to target resource). If no error occurs in a routing we refer to this routing as a *correct routing*.

Let *p* denote the link capacity ratio $p = BL_U / \sum_{neigh(A)} BL_i$ that node *U* for its neighbor *A*. For convenience of representation we simply assume *p* is a random variable and $\log p$ is uniformly distributed with the mean value $\log p_0$. Then the product of *n* random variables can be approximated as $p_0{}^n$.

Let $q = (d_a - 1) * p_0$ and choose proper p_0 to ensure $q < 1$ for scalability. Let $p_r(i)$ be the probability that a bit of the element in the *i*th column is 1. $p_r(i)$ is given by:

$$p_r(i) \approx \frac{klq^{i-1}}{m}, \ (1 \le i < c) \qquad (4)$$

$$p_r(c) \approx \frac{klq^{c-1}(1-q^{h_0-c})}{m(1-q)} \qquad (5)$$

Let random variable Y_i denote the number of bits which are indexed by *k* hash functions and set to 1 in BF vector $T^A_{N,i} (1 \le i \le c)$, in which *N* is an arbitrary neighbor other than the correct one. Let $P_r(Y_i = y)$ denote the probability that the value of Y_i is *y*, in which *y* is a non-negative integer and $y \le k$. $P_r(Y_i = y)$ is given by:

$$P_r(Y_i = y) = C^y_k p_r(i)^y (1 - p_r(i))^{k-y}, (1 \le i \le c) \qquad (6)$$

Suppose that the information of target resource *x* arrives at node *A* through its neighbor *B* after *h* hops propagation. If $h < c$ the information is stored in element $T^A_{B,h}$; otherwise it is stored in element $T^A_{B,c}$.

If $h < c$, let *r*(*h*) be the probability that a bit indexed by *k* hash functions in element $T^A_{B,h}$ is set to 1. *r*(*h*) is given by formula 7. Similarly if $h \ge c$, *r*(*h*) is given by formula 8.

$$r(h) \approx p_0^{h-1} + p_r(h) - p_0^{h-1} \times p_r(h), (1 \le h < c) \qquad (7)$$

$$r(h) \approx p_0^{h-1} + p_r(c) - p_0^{h-1} \times p_r(c), (h \ge c) \qquad (8)$$

Let random variable *X* denote the number of bits which are indexed by *k* hash functions and set to 1 in

BF vector $T^A_{B,h}(h < c)$ or $T^A_{B,c}(h \geq c)$, in which B is the correct neighbor. Let $P_r(X = x)$ denote the probability that the value of X is x, in which x is a non-negative integer less than or equal to k. $P_r(X = x)$ is given by:

$$P_r(X = x) = C_k^x r(h)^x (1 - r(h))^{k-x} \tag{9}$$

Suppose that the target resource information arrives at node A through its neighbor B after h hops propagation. A query routing at node A is a *correct routing* if and only if at least one element $T^A_{B,j}$ satisfies $Sim(x, T^A_{B,j}) \geq \alpha \times MaxSim(A), j \in [1,c], \alpha \leq 1$.

We first study element $T^A_{B,h'}$, if $h<c$, $h'=h$; otherwise $h'=c$. $Sim(x, T^A_{B,h'}) \geq \alpha \times MaxSim$ is equivalent to $\forall N \neq B, \forall i \in [1,c], Sim(x, T^A_{B,h'}) \geq \alpha \times Sim(x, T^A_{N,i})$. Let $P_{eff,h'}$ be the probability of realizing a *correct routing* due to element $T^A_{B,h'}$. $P_{eff,h'}$ is given by:

$$P_{eff,h'} = \prod_{i=1}^{c}{}^{d-1}\left(\sum_{s=1}^{k}(P_r(X = s) \cdot \sum_{j=0}^{\min(\lfloor s/\alpha \rfloor, k)} P_r(Y_i = j))\right) \tag{10}$$

We then study element $T^A_{B,i}, (i \in [1,c], i \neq h')$. Let $P_{eff,noise}$ be the probability of realizing a *correct routing* due to element $T^A_{B,i}, (i \neq h')$. $P_{eff,noise}$ is given by:

$$P_{eff,noise} \approx 1/d \tag{11}$$

The probability a node has degree d is $1/(b-a+1)$. The error rate $P_{err,h}$ and accuracy rate $P_{eff,h}$ at an arbitrary node h hops away from the target resource are given by:

$$P_{err,h} = \frac{1}{b-a+1} \cdot \sum_{d=a}^{b}(1 - P_{eff,h'})(1 - P_{eff,noise}) \tag{12}$$

$$P_{eff,h} = 1 - P_{err,h} \tag{13}$$

Let h_0 be the network diameter. Let n = 10000 (with the corresponding h_0 = 8), d_a = 4, k = 64, l = 3, m = 32k bit, p_0 = 0.25, c = 5, Th_1 = 1/4, Th_2 = 1/16 (Th_1 and Th_2 are the two thresholds which we discussed in section 2.3). The error rate $P_{err,h}$ computed by formula 12 is summarized in table 1.

Table 1. **Error rate with different values of *h*.**

h	2	3	4
$P_{\text{err},h}$	3.52×10^{-8}	2.60×10^{-2}	1.08×10^{-1}
h	5	6	7
$P_{\text{err},h}$	4.47×10^{-1}	7.37×10^{-1}	7.37×10^{-1}

3.2. Search performance

From table 1 we can conclude that 1) when h is greater than 5 hops the error rate is almost the same with different values of h, indicating that the information propagated more than 5 hops is indistinguishable from random noise and has little influence on error rate; and 2) when h is less than 3 hops the error rate is almost 0, indicating that the information propagated less than 3 hops is much stronger than the noise. Let h_1 be 5 and let h_2 be 2. Let h be the distance between the target resource and the query originator. We analyze the latency according to different values of h.

(1) If the query originator is very close to the target resource ($h \leq h_2$), we approximately think each query routing is correct. Let $hops_1(h)$ be the search latency in this case, $hops_1(h)$ is given by:

$$hops_1(h) \approx h \tag{14}$$

(2) If the query originator is not close to, but also not far away from the target resource ($h_2 < h \leq h_1$), let $hops_2(h)$ be the search latency in this case, $hops_2(h)$ is given by:

$$hops_2(h) \approx h + (d_a - 1) \times \sum_{i=h_2+1}^{h} P_{err,i} \tag{15}$$

(3) If the query originator is very far away from the target resource ($h > h_1$), let $hops_3(h)$ be the search latency in this case, $hops_3(h)$ is given by:

$$hops_3(h) \approx (NH + 1) \times h - NH \times h_1 + (d_a - 1) \times \sum_{i=h_2+1}^{h_1} P_{err,i} \tag{16}$$

Let h_0 be the maximum distance between the target resource and an arbitrary node. Let n be the number of nodes in P2P networks. Let n_h be the number of nodes h hops away from the target resource, according to the assumption at the beginning of this section, n_h is given by $n_h \approx d_a \times (d_a - 1)^{h-1}, (h < h_0)$. And n_{h0} is given by:

$$n_{h_0} \approx n - \frac{d_a}{d_a - 2}((d_a - 1)^{h_0 - 1} - 1) \tag{17}$$

Let $P_r(h)$ be the probability that the query originator is h hops away from the target resource, which is given by: $P_r(h) = n_h / n$. Let $E(hops)$ be the average latency.

$$E(hops) = \sum_{h=1}^{h_2}(P_r(h) \times hops_1(h)) + \sum_{h=h_2+1}^{h_1}(P_r(h) \times hops_2(h)) + \sum_{h=h_1+1}^{h_0}(P_r(h) \times hops_3(h)) \tag{18}$$

Let n = 10,000 and 2,500, NH = 2. According to the above analysis let h_1 = 5 and h_2 = 2. The average search latency by formula 18 is summarized in table 2.

Table 2. **Average search latency.**

n	h_0	E(hops)
10,000	8	18.326
2,500	7	13.169

4. Evaluations

We implement and evaluate *POP* with different configurations by modifying Neurogrid simulator [12] and running on a workstation with Intel Pentium 3.0GHz processor and 1GB memory. We compare the latency and cost of *POP* with those of other search mechanisms for unstructured P2P networks, including Gnutella (flooding), 2-way Random Walk (forwarding to two randomly chosen neighbor at each step), and *SQR* [7].

4.1. Simulation configurations

The common simulation configurations are summarized in table 3.

Table 3. **Common configurations.**

Parameter	Value
No. of Nodes (n)	2000,2500
Average degree of all nodes (d_a)	4
Distribution of degrees of nodes	Same as in Section 3
Size of pool for available resources	2500
No. of resources per node (l)	3
Size of the pool for keywords	3000
No. of keywords per resource	1
Distribution of resources	Uniform distribution
Distribution of searches	Uniform distribution
Simulation times	10,000

The private simulation configurations of each mechanism are shown in table 4.

Table 4. **Private configurations.**

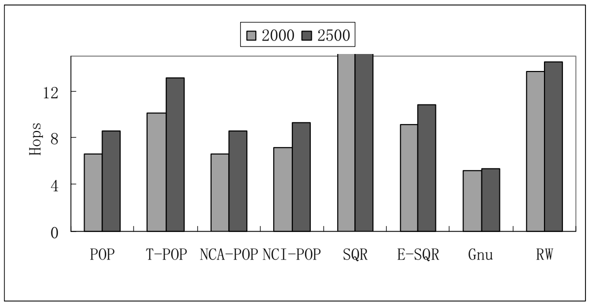
Parameter	Value
POP (With DBF)	
Width of Bloom Filter (m)	32k bit
No. of hash functions (k)	64
Mean value of log (p)	log 0.25
Fault-tolerant factor (α)	0.6
No. of initial search units	3
No. of hops between 2 enhances (NH)	2
Threshold to stop normal POP routing	$Th_1 = 1/4$
Threshold to stop large-scale enhance	$Th_2 = 1/16$
Mean value of Col. nNo. in *NIT* (c)	5
SQR (With EDBF)	
Width of Bloom Filter (m)	32k bit * 5
No. of hash functions (k)	64
Decay factor	0.3
Gnutella	
Forwarding manner	Flooding
Random Walk	
Forwarding manner	2-way

As discussed in section 2.3, in order to control the overall search cost, when *gMax* reaches Th_2 *POP* will stop the enhancement and order all units with zero local *Max* aggregate values to pause; and when *gMax* reaches Th_1 *POP* will order all units except the ones having *gMAX* to pause. By contrast we modify *POP* to *NCA-POP*, which has *No* such *Controls* and do not take into account the *Average* aggregates. Furthermore, we also simulate *POP* with NO use of the *Column* number *Information*, which is referred to as *NCI-POP*. Finally we refer to the theoretical model of *POP* as *T-POP*.

To stand out the difference between DBF and EDBF we replace *SQR* with *E-SQR* to compare with *POP*. *E-SQR* is an extension of *SQR* which is almost the same as *SQR* except using *POP*'s enhancement policy at the beginning of search until *gMax* is greater than Th_2.

4.2. Search latency and cost

In our simulations search latency is represented by the number of hops traversed from the query originator to the target resource; and search cost is represented by the total number of query messages (one search unit generates one message at each hop) during the search. The simulation results are summarized in figure 2 and the following analysis is based on the results with 2500 nodes.



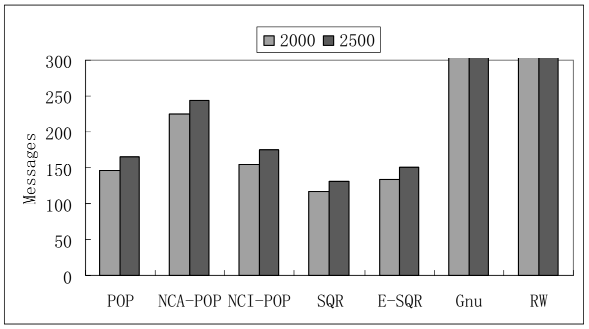


Figure 2. **Search latency and cost.**

The average search latency of *NCA-POP* is almost the same as that of *POP*, while *NCA-POP* has a 46.5% increase in average cost over *POP*. This proves that the policy for control of cost (including the *Average* aggregates) in *POP* successfully reduces the cost with almost no extra latency.

The average search latency of *POP* is similar to that of flooding and is only about 1/2 that of 2-way Random Walk. Furthermore, *POP* has more than an order of magnitude reduction in average cost over both flooding and 2-way Random Walk.

The latency of *SQR* (130.7) is too high to appear in figure 2, and so we only compare *POP* with *E-SQR*.

POP has a 9.3% increase in cost over *E-SQR*. But *POP* has a 27% reduction in latency over *E-SQR*.

4.3. Impact of different parameters

We further evaluate *POP* by simulations with different combinations of average *p* (link capacity ratio) and *NH* (hops between 2 enhancements), which is represented by (*p*, *NH*). The result is summarized in table 5.

Table 5. **Simulation results with different combinations of *p* and *NH*.**

	Latency	Cost
(0.5, 2)	7.915	108.8
(0.5, 4)	8.348	54.6
(0.33, 2)	8.237	153.4
(0.33, 4)	8.720	79.2
(0.25, 2)	8.538	166.1
(0.25, 4)	9.988	120.5

According to table 5, with the increase of *NH*, the search latency increases modestly while the search cost decreases greatly; and with the increase of *p*, the search latency and search cost decreases simultaneously.

5. Conclusions

In this paper, we first present Distributed Bloom Filter (DBF) to propagate the hint information with a bandwidth-aware manner, in which a node divides the received information and disseminates the fragments to its neighbors according to their bandwidth capacity. We then propose *POP*, a DBF-based search mechanism, in which query messages are viewed as search units to collaborate with each other. The effectiveness of *POP* is demonstrated through theoretical analysis and extensive simulations.

References

[1] S. Jiang, et.al, "Lightflood: an efficient flooding scheme for file search in unstructured peer-to-peer systems," in Proceedings of the 32nd *International Conference on Parallel Processing* (*ICPP*'03). Kaohsiung, Taiwan: IEEE Computer Society, October 2003.

[2] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," in Proceedings of *IEEE Infocom*, 2004.

[3] Chawathe Y, et.al. "Making Gnutella-like P2P systems scalable", in Proceedings of the *ACM SIGCOMM* 2003.

[4] Q. Lv, P. Cao, E. Cohen, K. Li and S. Shenker. "Search and Replication in Unstructured Peer-to-Peer Networks". Proc. of 16th *ACM International Conference on Supercomputing* (*ICS*'02), New York, USA, June 2002.

[5] F. Matias. "PlanetP: Using Gossiping to Build Content Addressable P2P Information Sharing Communities". *Technical Report*.

[6] S. C. Rhea and J. Kubiatowicz, "Probabilistic location and routing," in Proc. of *IEEE Infocom*, 2002.

[7] Abhishek Kumar, Jun (Jim) Xu and Ellen W. Zegura, "Efficient and Scalable Query Routing for Unstructured Peer-to-Peer Networks", in Proceedings of *IEEE Infocom*, 2005.

[8] Yunhao Liu, Li Xiao, and Lionel M Ni, "Building a Scalable Bipartite P2P Overlay Network", *IEEE IPDPS* 2004, USA, April 2004.

[9] Yunhao Liu, Xiaomei Liu, Li Xiao, Lionel M Ni, and Xiaodong Zhang, "Location-Aware Topology Matching in P2P Systems", *IEEE INFOCOM*, 2004.

[10] B. Bloom. "Space/time trade-offs in hash coding with allowable errors". *Communications of the ACM*, Vol. 13, No. 7, July 1970.

[11] Yiming Zhang. "Collaborative Search in Large-scale Unstructured Peer-to-Peer Networks". *Tech. Report*, available at http://www.kylinx.com/Papers/POP.pdf.

[12] S. R. H Joseph. "NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks". Proc. of *IWPC*, Pisa, Italy, 2002.

[13] S. Madden, et al. "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks". In Proceedings of *OSDI*, 2002.

[14] R. VanRenesse, K. P. Birman, and W. Vogels. "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining". *TOCS*, 2003.