Hindawi Publishing Corporation
Advances in Multimedia
Volume 2007, Article ID 64938, 10 pages
doi:10.1155/2007/64938

Research Article

A Hybrid Query Scheme to Speed Up Queries in Unstructured Peer-to-Peer Networks

Zhan Zhang, Yong Tang, Shigang Chen, and Ying Jian

Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611-6120, USA

Received 31 January 2007; Accepted 6 June 2007

Recommended by Ben Y. Zhao

Unstructured peer-to-peer networks have gained a lot of popularity due to their resilience to network dynamics. The core operation in such networks is to efficiently locate resources. However, existing query schemes, for example, flooding, random walks, and interest-based shortcut suffer various problems in reducing communication overhead and in shortening response time. In this paper, we study the possible problems in the existing approaches and propose a new hybrid query scheme, which mixes intercluster queries and intracluster queries. Specifically, the proposed scheme works by efficiently locating the clusters, sharing similar interests with intercluster queries, and then exhaustively searching the nodes in the found clusters with intracluster queries. To facilitate the scheme, we propose a clustering algorithm to cluster nodes that share similar interests, and a labeling algorithm to explicitly capture the clusters in the underlying overlays. As demonstrated by extensive simulations, our new query scheme can improve the system performance significantly by achieving a better tradeoff among communication overhead, response time, and ability to locate more resources.

1. INTRODUCTION

Peer-to-peer networks surged in popularity in recent years. The core operations in most peer-to-peer networks is to efficiently locate data items, in which the fundamental challenges are to achieve faster response time, smaller network diameter, stronger ability of locating more resources, and better resilience to network dynamics.

Structured P2P networks have been proposed by many researchers [1–7], in which distributed hash tables (DHTs) are used to provide data location management in a strictly structured way. Whenever a node joins/leaves the overlay, a number of nodes need to update their routing tables to preserve desirable properties for fast lookup. While structured P2P networks can offer better performance in response time and communication overhead for query procedures, they suffer from the large overhead for overlay maintenance due to network dynamics.

Unstructured P2P networks such as Gnutella rely on a random process, in which nodes are interconnected in a random manner. The randomness offers high resilience to the network dynamics. However, basic unstructured networks rely on flooding [8] for users' queries, which is expensive in computation and communication overhead. Consequently, scalability has always been a major weakness for unstructured networks [9]. Even with the use of super nodes in Morpheus [10] and KaZaA [11], the traffic is still high, and even exceeds web traffic.

Searching through random walks is proposed in [12–14], in which incoming queries are forwarded to the neighbor that is chosen randomly. In random walks, there is typically no preference for a query to visit the most possible nodes maintaining the needed data, resulting in long response time.

Interest-based shortcut [15] exploits the locality of interests among different nodes. In this approach, a peer learns its shortcuts by flooding or passively observing its own traffic. A peer ranks its shortcuts in a list and locates content by sequentially asking all of the shortcuts on the list from the top until content is found. The basic principle behind this approach is that a node tends to revisit accessed nodes again since it was interested in the data items from these nodes before. The concept of interest similarity is vague and it is difficult to make a subtle, quantitative definition based on it. In addition, it may cause new problems as discussed later.

In this paper, we take the unstructured approach, and propose a new query scheme to address these problems. The main contributions of the paper include the following.

(i) *We define a metric, independent of any global information, to measure the interest similarity between nodes.*

Based on the metric, we propose a clustering algorithm to cluster nodes sharing similar interests with small overhead, and fast convergence.

(ii) *We propose a distributed labeling algorithm to explicitly capture the borders of clusters without any extra communication overhead.*

(iii) *We propose a new query scheme, which is able to deliver a better tradeoff among response time, communication overhead, and the ability to locate more resources by mixing intercluster queries and intracluster queries.*

The rest of the paper is organized as follows. Section 2 reviews the possible problems in prior approaches. Section 3 gives the overview of our scheme. Section 4 defines the interest similarity and proposes a light-weight algorithm to cluster nodes within the same interest group and a labeling algorithm to explicitly border the clusters. Section 5 introduces our query scheme in detail. Section 6 evaluates the scheme with extensive simulations. Section 7 draws the conclusion.

2. INEFFICIENCY IN PRIOR WORKS

Small communication overhead and short response time are the two main concerns in designing efficient query schemes in peer-to-peer networks. However, current approaches suffer various problems in achieving a better tradeoff between them due to the blindness in searching procedures.

Flooding

Flooding [8, 16] is a popular query scheme to search a data item in fully unstructured P2P networks such as Gnutella. While flooding is simple and robust, its communication overhead, that is, the number of messages, increases exponentially with the hop number. In addition, most of these messages visit the node that has been searched in the same query, and they can be regarded as duplicate messages. Consequently, communication overhead and scalability are always the main problems in this approach [9, 17].

Random walks

Random walks [12–14, 18] rely on query messages randomly selecting their next hops among neighbors with equal probabilities to reduce the communication overhead. A query may have to go through many hops before it successfully locates the queried data item. Consequently, this approach takes a long time to locate queried data items. If the networks are well clustered (nodes with similar interests are densely connected), it is expected that the query latency can be reduced significantly. However, it is not true, because the number of hops escaping out of the cluster decreases exponentially with the ratio r of the number of intercluster edge to the number intracluster edges, as shown in Figure 1.

In the case of a network with a small value of r, for example, $r < 0.01$, if queried data items are in different clusters from the source node, a query message has to walk a long distance to be able to traverse the cluster border and locate the queried data items. In the case of a network with a large value of r, for example, $r > 0.1$, query requests may escape out of the original cluster within a small number of hops, resulting in a long response time if the queried data is in the original cluster. These observations are also demonstrated by our simulations in Section 6. Consequently, random walks may suffer long response time regardless of the network having been well clustered or not.

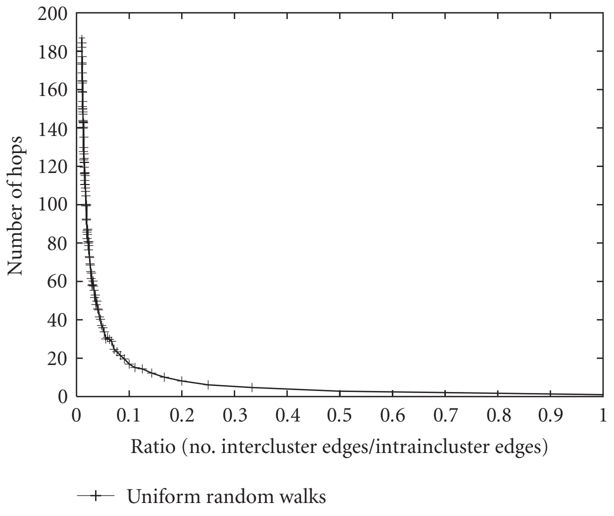

FIGURE 1: The number of hops of random walks escaping out of the cluster decreases exponentially with the ratio of number of intercluster edges to the number of intracluster edges.

Interest-based shortcut

Interest-based shortcut, for example, [15], tries to avoid the blindness in random walks by favoring nodes sharing similar interests with the source, which can be regarded as a variation of Markov random walks, biased towards some specific nodes. Markov random walks may accelerate the query process to some extent in some cases. However, it causes new problems. Suppose that nodes in an interest group have formed a cluster, and query messages can be artificially confined in this specific cluster. In the sense of nodes in the cluster share similar interests, any of them possibly maintains the queried data. Thus, the query procedure should shorten the covering time of the whole cluster instead of the hitting time of some specific nodes in it. However, due to the bias in selecting next hop in Markov random walks, it tends to keep visiting some specific nodes, resulting in less distinct nodes being covered comparing to uniform random walks, as illustrated by Figure 2. Consequently, Markov random walks work worse than uniform random walks if both of them can be confined in specific clusters.

3. SYSTEM OVERVIEW

Researchers [15, 19] have found many peer-to-peer networks exhibit small-world topology, and most of queried data items are offered by the nodes, which share similar interest with the source node.

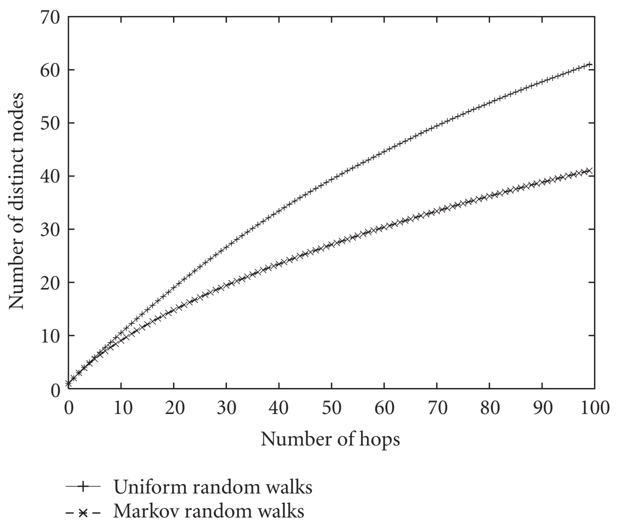

Figure 2: Markov random walks discover a smaller number of nodes than uniform random walks do.

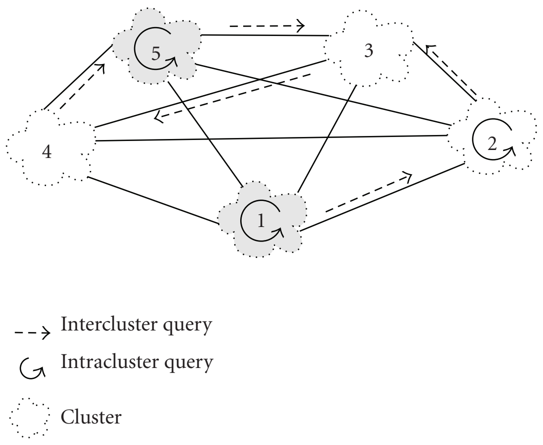

Figure 3: A query scheme mixing intercluster queries and intracluster queries (the nodes in the grey clusters fall into the same interest group).

Intuitively, the nodes sharing similar interests with the source node should have higher priority to be searched than others. Practically, there are two challenges in designing such a query scheme. The first one is how to cluster nodes with similar interests based on the small-world property of P2P networks. By saying "similar interests," we actually mean that two nodes are interested in a common set of data items. Thus, the number of common accessed data items can serve as a metric to measure the interest similarity between two nodes. A clustering algorithm based on the metric can be easily designed to densely connect the nodes in the same interest group. Moreover, each node u can explicitly pick up a set of *intercluster* neighbors that have different interests from u, and a set of *intracluster* neighbors that share similar interests with u. Take Figure 3 as an example. The network consists of 5 clusters, and nodes in the same cluster fall into the same interest group. Note that there exists an interest group consisting of two clusters, 1 and 5.

Suppose the network has been well clustered, and each node explicitly maintains a set of intercluster and intracluster neighbors. The second challenge is how to fast locate the clusters that share similar interests with the source node and how to exhaustively search nodes in the found clusters if the queried data items are in the source node's interest group. We introduce two types of queries, *intercluster* queries and *intracluster* queries. The intercluster queries are for the purpose of discovering the clusters that share similar interests with the source node, and they are issued by source node, carry the interest information, and only travel on intercluster neighbors. It can be expected that clusters sharing similar interests with the source node can be located quickly, because the number of cluster is much smaller comparing to the network size, and intercluster queries only travel among different clusters. The intracluster queries are spawned by intercluster queries when a cluster sharing similar interest with the source node is hit. An intracluster query thoroughly go through nodes in the found cluster, where it is spawned, by only traveling on intracluster neighbors. How to estimate to what extent a cluster has been searched will be discussed later. Note that intercluster queries and intracluster queries can be easily implemented if each node explicitly knows the types of its neighbors.

Occasionally, queried data may be out of the source node's interest group, and possibly maintained by a cluster(s) with different interests. This problem is addressed by blind search: intercluster messages randomly spawning intracluster messages when hitting clusters with different interests.

For example, in Figure 3, first inter-queries are initiated by a node in cluster 1, which travel among different clusters. By the interest information carried in the inter-queries, cluster 5 is found to share similar interests when it is hit, and an intracluster query is spawned, which then will exhaustively search the nodes in it. In addition, an intracluster query is spawned in cluster 2 by intercluster queries to support blind search.

4. CLUSTERING ALGORITHM

4.1. Measuring the interest similarity between two nodes

Cluster is generally formed by connecting nodes with similar interests in a network. Thus, we start our discussion with the definition of interest similarity between two nodes in P2P networks.

If node u and node v share similar interests, then it is very likely that they have accessed same data items more or less previously. Therefore, the size of the common subset of accessed data items can serve as a metric to measure to what extent the interests of two nodes are similar.

However, each node may offer hundreds of data items, and hence, there may exist a large number of data items even in a small network. As a result, only if u and v have visited a large number of data items, respectively, they are able to show some degree of similarity. An alternative to evaluate

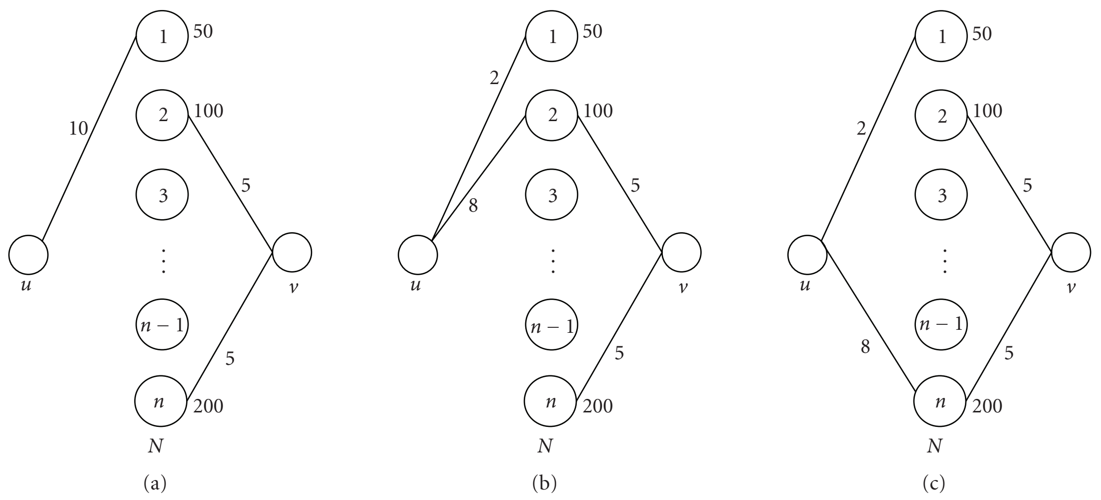

FIGURE 4: Interest similarity between nodes. The number of data items in 1, 2, and n is 50, 100, and 200, respectively. (a) No common visited nodes (different interests), (b) u and v have visited node 2 (100 data items) with 8 and 5 times, respectively (a certain level of similar interests), (c) u and v have visited node n (200 data items) with 8 and 5 times, respectively (falls in between).

the interest similarity is by the number of common accessed nodes, which may enable a clustering algorithm to converge faster than the former approach. The problem in this approach is that two nodes visiting a common node does not indicate they have similar interests, because a node may offer data items belonging to multiple interest groups. For instance, a user u may offer resources for two groups: a number of mp3 music files for one group, and a number of research literatures in P2P networks for the other group. It is possible that two nodes that have visited u may be interested in data items in different interest group. Thus, we have to address the discrepancy between the common set of accessed nodes and the common set of visited data items.

Suppose that there are n nodes $N = \{1, 2, \ldots, n\}$ in the whole P2P network. Suppose a node i offers a number of data items to others. It categorizes (maps) all of these data items into α_i different categories, denoted as $C^i = \{c_1^i, c_2^i, \ldots, c_{\alpha_i}^i\}$. Suppose a data item x in i is mapped to a category $c^i(x)$, where $c^i(x) \in C^i$. How to categorize the data items is determined by the node i independently. For instance, node i may classify music files as category 1, while another node may classify music files as category 2. On the other hand, node i may fall into multiple interest groups, denoted as $G^i = \{g_1^i, g_2^i, \ldots, g_{\beta_i}^i\}$.

For a node u, the access history with respect to each of its interest groups, for example, g_1^u, can be specified by a set of data items x, denoted as $(i, c^i(x))$, where i represents the node offering the data item, and $c^i(x)$ is the category in C^i defined by i. If two nodes u and v share "similar interests," for example, $g_1^u \approx g_2^v$, their histories for g_1^u and g_2^v tend to consist of a common set of $(i, c^i(x))$.

Note that in the above definitions, each node determines its interest groups and categories independently, indicating a node need not maintain any global information.

For easy explanation, we study a basic approach by assuming each user only falls into one interest group, and offers one category of data items. In this scenario, the access history can be represented by the accessed nodes alone. This approach can be easily extended to the multicategories and multigroups based on the definitions above.

One node u may access another node multiple times for different data items, and hence, the access history of node u can be represented by a vector $V^u = (v_1^u, \ldots, v_n^u)$,[1] where v_x^u represents the number of times u has visited node x. To cancel out the number of queries a node has issued, the access vector V^u is normalized to the *frequency vector* F^u, in which the ith element in F^u is denoted as f_i^u, computed by V^u as $f_i^u = v_i^u / \sum_{j \in N} v_j^u$, representing the frequency of the corresponding node i having been accessed.

Note that the value in F^u falls into the range $[0, 1]$. If u has never accessed node i, the corresponding element f_i^u is equal to 0. The summation of all elements is equal to 1.

Furthermore, if the number of data items in node i, denoted as d_i, is large, the chance that two nodes u and v have visited common data items in i may be small even if both of them have visited i multiple times. As an example, in the middle and right parts in Figure 4, u and v have visited one common node. But u and v in the middle part have more chance of having visited common data items because the number of data items in node 2 is half of that in node n. To account for this issue, we introduce a weighted diagonal matrix W with (i, i)th value $w_{i,i}$ equal to $1/d_i$. It represents the probability of both u and v visiting a common data items, if both of them visit i once.

Now we define the following metric to evaluate the interest similarity between two nodes:

$$A^{u,v} = F^{u^T} W F^v = \sum_i f_i^u f_i^v \frac{1}{d_i}. \quad (1)$$

[1] The real size of the data structure maintaining V^u is much smaller than the network size n, and can be fixed to only record the nodes accessed most frequently by u.

Take Figure 4 (middle) as an example:

$$
\begin{aligned}
A^{u,v} &= F^{u^T} W F^{v} \\
&= \begin{pmatrix} .2 \\ .8 \\ 0 \\ \cdot \\ \cdot \\ 0 \end{pmatrix}^T \begin{pmatrix} .02 & 0 & 0 & 0 & 0 & 0 \\ 0 & .01 & 0 & 0 & 0 & 0 \\ 0 & 0 & d_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdot & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdot & 0 \\ 0 & 0 & 0 & 0 & 0 & .005 \end{pmatrix} \begin{pmatrix} 0 \\ .5 \\ 0 \\ \cdot \\ \cdot \\ .5 \end{pmatrix} \\
&= 0.004.
\end{aligned} \tag{2}
$$

Similarly, the interest similarity in Figure 4(c) (right) is 0.002, which means nodes in the middle example show more interest similarity.

If we view f_i^u and f_i^v as the probabilities of nodes u and v visiting node i, and $1/d_i$ as the probability of u and v visiting a common data items if both of them visit i, then the summation $A^{u,v}$ can be used to predict the probability that both u and v will visit a common data item in their future queries.

Note that if a node i has not been visited by both u and v, then $f_i^u = 0$ and/or $f_i^v = 0$, indicating that a node does not need to maintain any information of the nodes it has never visited. As we have discussed, the size of the vector V^u is fixed. Hence, both the storage for access history and the computation overhead for $A^{u,v}$ are constant.

Our definition is advantageous in manyfold. First, nodes u and v need not maintain any global information to compute $A^{u,v}$. Second, the frequency vector cancels out the effect of the number of queries that a node has issued, enabling a clustering algorithm based on this definition to converge fast with the average number of queries. Third, the definition prefers the nodes with good properties. For instance, if two nodes i, j maintaining the same data items can be accessed by 1 Mp/s and 56 Kbp/s, respectively, i obviously will be accessed more often, resulting in a larger value of f_i^u and f_i^v. Fourth, it reduces the impact of the possible discrepancy in the category definitions by nodes. For instance, if a category in node i is poorly defined, and consists of data items belonging to various interest groups, then the category will be seldom accessed comparing to its size, resulting in a small value of $f_i^u f_i^v (1/d_i)$.

4.2. Clustering nodes with similar interests

Given the metric to evaluate the interest similarity between two nodes, we propose a light-weight clustering algorithm to connect nodes sharing similar interests.

In our strategy, each node i maintains a list L with limited size, for example, 30, to record the nodes that possibly share same interests with itself. Each time a query message is processed, the similarity between the querying node itself and the node that owns the data items is computed. The overhead of similarity computation is fixed given that the size of the list L is predefined. The newly obtained interest similarity, and the corresponding node's address are inserted into the list L. If the list is full, the stored neighbor with the lowest interest similarity is dropped.

By assuming that interests of nodes will not shift in a limited time frame, the nodes collected in L possibly fall into the same interest group as i, and will serve as candidates of its intracluster neighbors.

4.3. Bounding clusters

Although a small-world topology can be formed along with queries by the above clustering algorithm, existing query schemes, for example, random walks, can only benefit marginally from it as we discussed in Section 3.

To exploit the characteristics of the small-world topology, our approach is to explicitly capture the clusters in the underlying topology by each node i maintaining a set of intercluster neighbors in other interest groups, and intracluster neighbors in its own interest group.

For intercluster neighbors, a node i can learn them easily. For example, i can issue a certain number of random-walk messages only traveling on other nodes' intercluster neighbors, and choose the nodes hit by the messages as its intercluster neighbors. Note that the list L collects candidates of its intracluster neighbors and should not overlap with the set of intercluster neighbors.

Thus, it is of the most importance to learn the intracluster neighbors, which can be selected from the nodes collected in the list L. The purpose of intracluster neighbors is to confine intracluster queries within a specific interest group. Two nodes falsely regarded as intracluster neighbors may create a dramatic impact because an intracluster query may traverse to another cluster with different interests. On the contrast, two nodes i and k that are falsely regarded as intercluster neighbors will only have limited impact, because i and k may be connected by other intermediate intracluster neighbors j. In addition, the chance of i and k falling into the same cluster tends to become larger along with query procedures if they are in the same interest group.

Based on this observation, we propose a labeling algorithm, which ensures that if a link (i, j) is labeled as an intracluster edge, then i and j are in the same interest group with high probability.

For a node i, we normalize the interest similarity of its neighbors j in L as follows, where k is the neighbor of i:

$$
p_{i,j} = \frac{A^{i,j}}{\sum_k A^{i,k}}. \tag{3}
$$

If a matrix P is organized such that its i, jth element is $p_{i,j}$, then the rows in P sum to 1 as the matrix P is row stochastic. Intuitively, $p_{i,j}$ can be viewed as the transition probabilities for the Markov random walk.

The transition probability $p_{i,j}$ can serve as a good metric to determine whether i and j are in the same interest group or not by introducing a threshold, denoted as T, as a lower bound. T can be set as a relatively larger value, because the false negative has limited impact as discussed. Suppose there are α neighbors in L that are possibly in the same interest group with i. T can be set as $1/\alpha$. Note that $p_{i,j}$ and T are computed by node i locally. Thus, the labeling algorithm does not involve any extra communication.

5. A HYBRID QUERY SCHEME

5.1. Mixing intercluster queries and intracluster queries

By explicitly capturing the cluster structures in the underlying network, we can formally define the following three types of query messages.

The first one is called *l-query message*, which is a special type of intercluster message only traveling on intercluster neighbors. The purpose of it is to quickly locate the clusters that may share similar interests with the source node and disperse intra-queries among different clusters. Messages in this type are issued by the source node, and walk among different clusters randomly. Moreover, if the queried data is in the source node's interest group, l-query messages should piggyback the source node's frequency vector, such that nodes hit by the messages can determine whether their clusters share similar interests with the source node.

The second one is called *s-query message*, which is a special type of intracluster message confining itself within a specific cluster by doing uniform random walks only on intracluster neighbors. s-query messages are only spawned/issued in the clusters that share similar interests with the source node. The purpose of it is to exhaustively search nodes that fall into the same interest group as the original node. Messages in this type are spawned by l-query messages when clusters sharing similar interests are hit.

Now the problem is how a node estimates to what extent the cluster has been covered. If the cluster has been well covered, the s-query message should be discarded in order to reduce the query overhead. Accurately estimating the covering time of a cluster is difficult and resource-consuming in a distributed system. Heuristically, if the message has been consecutively hitting a certain number, denoted as h, of nodes that have been visited before, it indicates that most of nodes have been covered, and hence, the message should be discarded. This information can be maintained by a counter in each s-query message.

The last one is called *b-query message*, which is also a special type of intracluster message similar to s-query message. But b-query messages may be spawned in clusters that have different interests. The purpose of it is to support blind search, because occasionally, the queried data may be out of the source node's interest group. The chance that the queried data item in a cluster has different interests is very small. Thus, once a b-query message hits a node that has been visited by intracluster messages before, the message is discarded in order to reduce the query overhead.

To control the communication overhead, the total number of concurrent query messages has to be limited. The source node needs to count the number of l-query, s-query, and b-query messages, denoted as m_l, m_s, and m_b, respectively. The overhead to maintain the counter is negligible given that the counter needs to be updated only if s-query message or b-query message is spawned or discarded. Only if the summation of m_l, m_s, and m_b is smaller than a certain number, denoted as m, a new b-query message can be spawned to support blind search. s-query messages can be spawned without restriction, and thus, the total number of concurrent messages may be larger than m temporarily. In addition, all messages need to periodically check the status of the source node so that they can stop if the query has been successfully returned.

With these three types of messages, a query scheme is designed as follows.

Initialization

To initiate a query request, a node u issues a number m_l of l-query messages. If the queried data item falls in u's interest group, l-query messages carry the source's frequency vector, and a certain number m_s of s-query messages are issued to exhaustively search its own cluster. Otherwise, a b-query message is issued in the meantime.

Receiving an l-query message

In the case of a node u receiving an l-query message, it calculates the interest similarity with the source node. If u shares similar interests, for example, the similarity is larger than a small value, it spawns a new s-query message and update m_s maintained by the source node. Otherwise, a new b-query message is spawned if the node has not been hit by s-query messages and b-query messages, and $m_l + m_s + m_b < m$. Finally, node u forwards the received message to a randomly selected intercluster neighbor.

Receiving an s-query message

In the case of a node u receiving an s-query message, if u has been hit by s-query messages or b-query messages, it increases the counter in the message by 1. Otherwise, it resets the counter to 0. Next, if the counter is larger than the threshold h, the node discards the message and notifies the source node to update the counter m_s. Otherwise, it forwards the message to a randomly selected intracluster neighbor.

Receiving a b-query message

In the case of a node u receiving a b-query message, if it has been hit by messages in s-query messages and b-query messages, the node discards the message and notifies the source node to update the counter m_b. Otherwise, it forwards the message to a randomly selected intracluster neighbor.

Our scheme can be considered to be *stateful*, in which if the same queries are reissued multiple times, less intracluster queries will be spawned in the clusters that have been well searched, and in contrast, more intracluster queries will be spawned in the less-searched clusters, resulting in stronger ability to discover more resources/replicas.

5.2. Reducing the communication overhead

By mixing intercluster and intracluster queries, it can be expected that the system performance can be improved significantly. Because the access vector V^u of a node u can be fixed

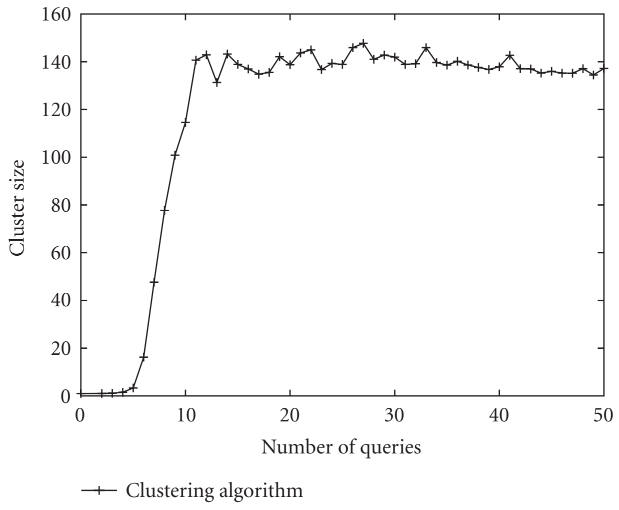

Figure 5: The effect of the average number of queries on the cluster size.

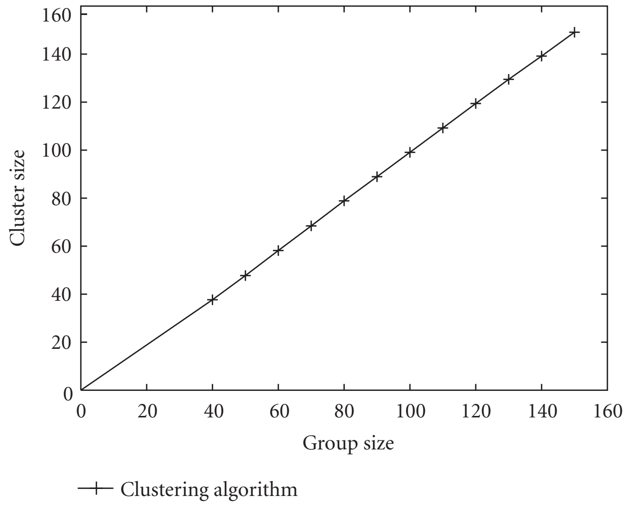

Figure 6: The interest association is a good metric to estimate interest similarity.

to a small size, the extra overhead will not increase largely (only l-query messages need to carry the frequency vector).

Moreover, l-query messages only travel among different clusters, and the number of clusters, especially in a well-clustered network, is much smaller comparing to the real network size. It can be expected that most of clusters can be covered by l-query messages within a small number of hops. Thus, l-query messages can remove the frequency vector from the payloads after a certain number of hops. In the meantime, a source node can specify one l-query message to keep the frequency vector for the case that some clusters sharing similar interests with the source node have not been discovered after the specified number of hops.

6. SIMULATION

In this section, the performance of the proposed clustering algorithm and query scheme is studied by simulations. If not explicitly defined, the default number of nodes is 10 000, and each node maintains 1 000 data items, which are randomly generated. The number of nodes in each interest group is 150, the average number of queries issued by each node is 30, and the threshold h is equal to 10. We set m to be 32, and m_l to be 16. Moreover, the probability of a node incorrectly classifying its queries or data items is 0.1. We also simulate other scenarios, in which nodes classify its queries or data items with various probabilities. The simulation results are similar, except that it takes a few more queries for the converge of the clustering algorithm.

We compare our scheme to random walks, in which a source node issues 32 random walk messages in each query, correspondingly. We also have compared our scheme to flooding schemes. As expected, we observe the flooding schemes suffer from very large communication overhead.

In Figures 7–10, the legend "Uniform random walks (0)"/"Uniform random walks (1)" denotes that the queried data items are out of/in the source node's interest group in the uniform random walks query scheme, and similarly "Inter/intra (0)"/"Inter/intra (1)" denotes that the queries are out of/in the source node's interest group in the proposed scheme.

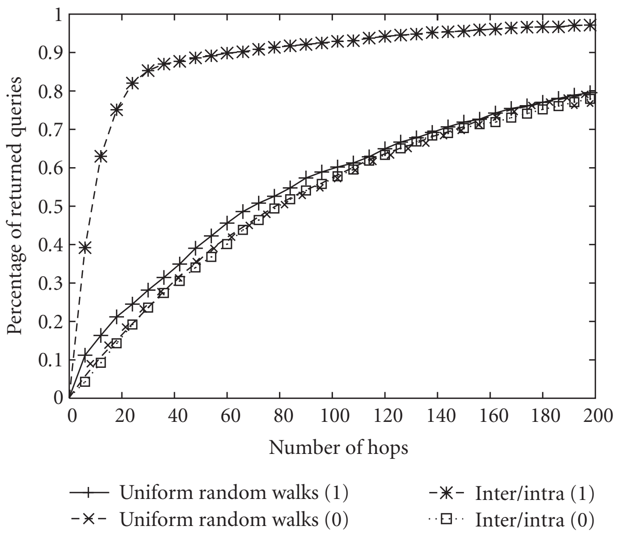

Figure 7: The percentage of returned query within a specific hop number.

First, we study the effectiveness of the metric measuring interest similarity and the clustering algorithm. In Figure 5, it is observed that when the average query number is larger than 10, the algorithm reaches a stable state and almost all nodes in the same interest group form a single cluster. It indicates that our algorithm converges fast with the average number of queries, which is especially useful in P2P networks, where nodes tend to join/leave the system more frequently.

By Figure 6, it can be observed that the average number of nodes in a cluster is almost the same as group size, demonstrating that $A^{u,v}$ can effectively measure the nodes' interest similarity.

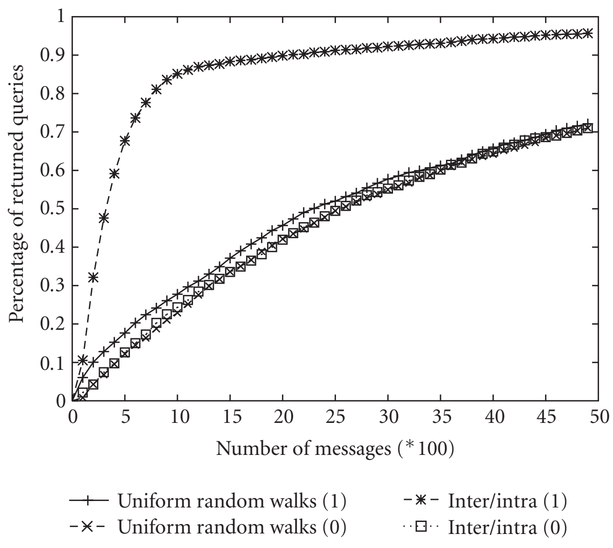

Figure 8: The percentage of returned query within a specific message number.

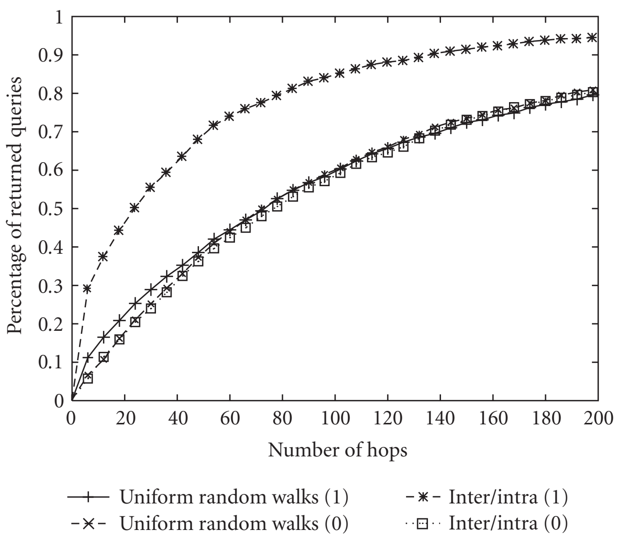

Figure 9: The percentage of returned query within a specific hop number in a less-clustered network.

Second, we study the performance of our scheme with respect to query latency and communication overhead.

In Figure 7, it is observed that if the queried data items fall into the original node's interest group, the number of hops needed for the majority of the queries is significantly reduced to about 20, while in the uniform random walks, it takes a much longer time. Correspondingly, the number of messages is also much smaller in our scheme than that in random walks, as shown in Figure 8. The figures also show that if the queried data are out of the source node's interest group, the performance of our scheme is similar to uniform random walks. Note that a longer response time is acceptable since only a few queries will be out of source node's interest group in many P2P networks. In addition, these two figures also demonstrate that random walks for queries in the source node's interest group can only benefit marginally from the underlying clustered topology, for example, only a little larger percentage of them can be returned than those out of source node's interest group within the same number of hops (messages).

We also have studied the performance of a network, in which each group consists of multiple different clusters, as shown in Figures 9 and 10. The results show the similar trends, which keeps true with respect to all other metrics that will be studied later. Moreover, comparing Figure 7 with Figure 9, and Figure 8 with Figure 10, it can be observed that the performance of random walks in two different (well-clustered and poor-clustered) networks is similar, which further verifies our argument in Section 3.

As have been observed, when the queried data items are in the source nodes' interest group, our scheme works much better than random walks. The reason behind it is that our scheme can discover more distinct nodes in the source nodes' interest groups within the same number of messages or hops, as shown in Figures 11 and 12. In these figures, it can be observed that within the first 1 000 messages or 30 hops, more

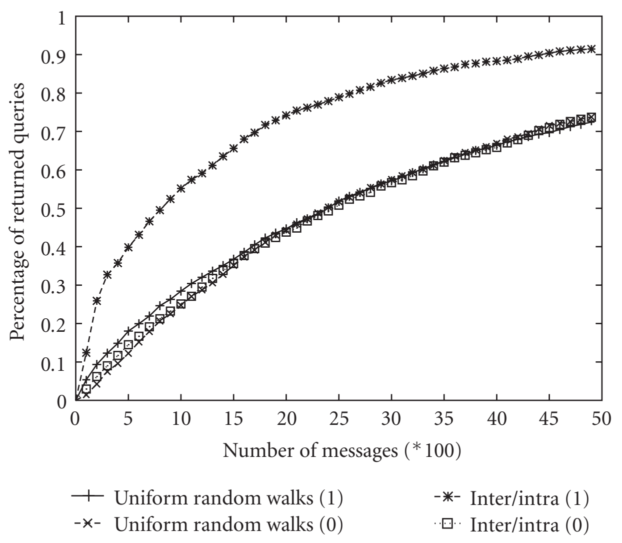

Figure 10: The percentage of returned query within a specific message number in a less-clustered network.

than 120 nodes in the source node's interest group have been searched by query messages. Consequently, the majority of queried data falling into the node's interest group can be found with smaller overhead and shorter latency. It also indicates that our scheme has a strong ability to locate more replicas since it can discover a much larger number of nodes sharing similar interests.

Occasionally, the queried data item may be maintained by nodes in other interest groups, or classified into wrong interest group by source node. In the former case, *l*-queries will not carry any interest information, but in the latter case, *l*-queries will carry wrong interest information. In both cases, the efficiency of our query scheme can be evaluated by the

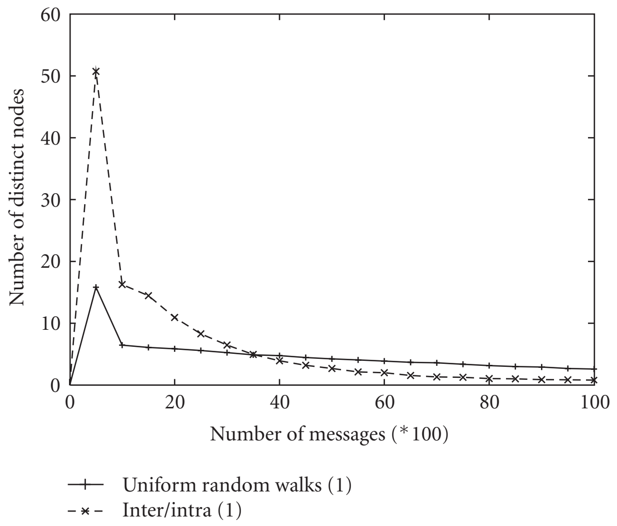

Figure 11: The number of distinct nodes discovered in the same group within a certain message range.

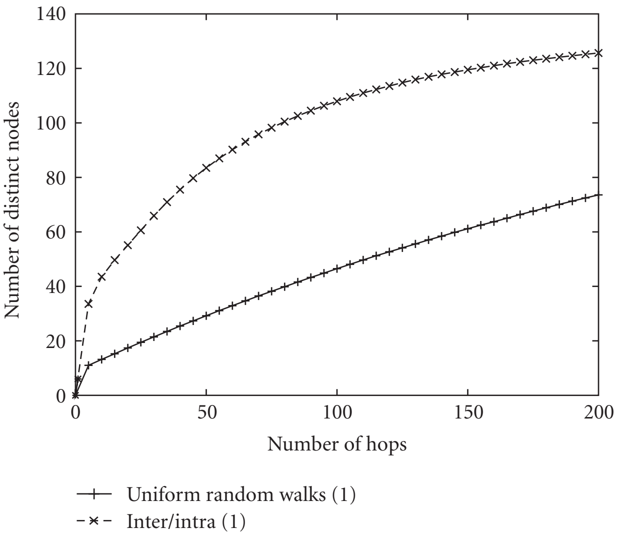

Figure 12: The number of distinct nodes discovered in the same group within a specific hop number.

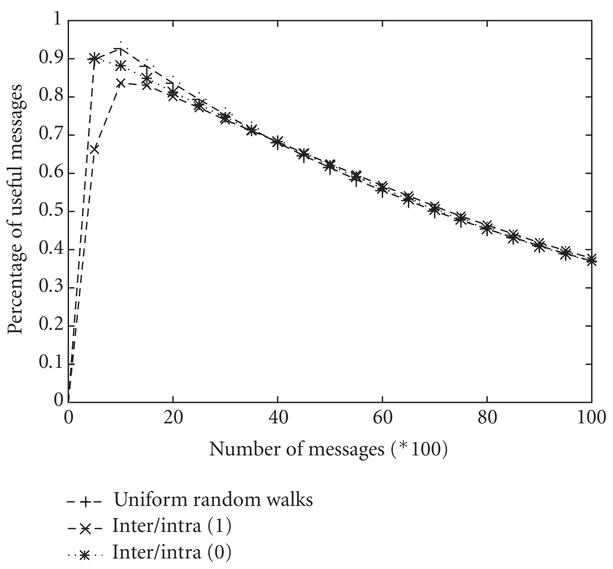

Figure 13: The percentage of messages discovering distinct nodes within a certain message range.

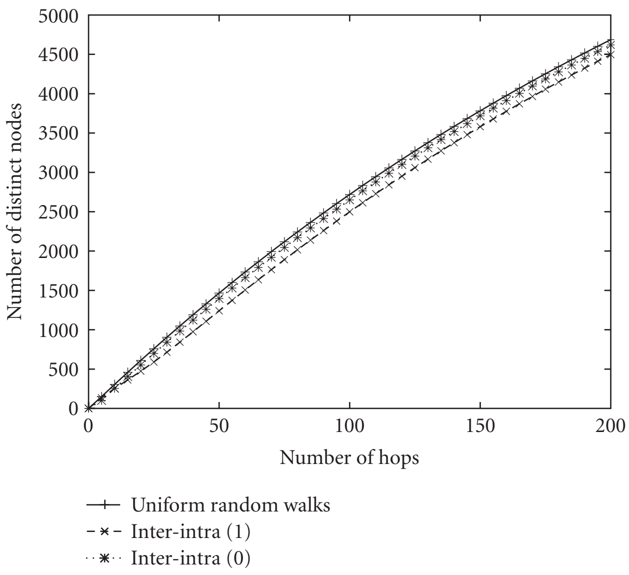

Figure 14: The total number of distinct nodes discovered within a specific hop number.

number of distinct nodes discovered by queries, including those out of the source node's interest group, within a certain number of messages and hops. Note that whether the queries are in or out of the original node's interest group makes no difference to random works. Figures 13 and 14 show that in the first 1 000 messages, if the queries carry interest information, fewer distinct nodes can be searched in our scheme. The reason is that *s*-query messages mistakenly exhaustively search the nodes in the clusters that share "similar" interests in the beginning, which has been demonstrated by our previous simulations. Consequently, the number of *b*-query messages is limited. Along with the increment of the number of messages/hops, our scheme works similar to the uniform random walks. It is because after most of nodes sharing similar interests are covered, more *b*-query messages will be spawned to search clusters with different interests, which are able to discover more distinct nodes. In addition, by the figures, if the queries carry no interest information, our scheme works similar to uniform random walks.

7. CONCLUSION

In this paper, we strictly define the metric to measure the interest similarity between nodes. A distributed clustering algorithm has been also presented, which gives P2P networks better resilience to network dynamics. We propose an algorithm to explicitly capture clusters in the underlying networks without any extra communications. A query scheme mixing intercluster and intracluster queries has been designed for unstructured P2P networks. It can achieve a better tradeoff among communication overhead, response time, and the ability to locate more resources (replicas). The performance of the algorithms has been demonstrated by simulations.

REFERENCES

[1] I. Stoica, R. Morris, D. Liben-Nowell, et al., "Chord: a scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01)*, pp. 161–172, ACM Press, San Diego, Calif, USA, August 2001.

[3] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: a resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, 2004.

[4] A. I. T. Rowstron and P. Druschel, "Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware '01)*, pp. 329–350, Heidelberg, Germany, November 2001.

[5] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," *Theory of Computing Systems*, vol. 32, no. 3, pp. 241–280, 1999.

[6] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: a scalable and dynamic emulation of the butterfly," in *Proceedings of the 21st Annual Symposium on Principles of Distributed Computing (PODC '02)*, pp. 183–192, ACM Press, Monterey, Calif, USA, July 2002.

[7] A. Kumar, S. Merugu, J. Xu, and X. Yu, "Ulysses: a robust, low-diameter, low-latency peer-to-peer network," in *Proceedings of the 11th IEEE International Conference on Network Protocols (ICNP '03)*, pp. 258–267, Atlanta, Ga, USA, November 2003.

[8] N. Chang and M. Liu, "Revisiting the TTL-based controlled flooding search: optimality and randomization," in *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MOBICOM '04)*, pp. 85–99, Philadelphia, Pa, USA, September-October 2004.

[9] J. Ritter, 2001, Why Gnutella can't scale. no, really. http://www.darkridge.com/~jpr5/doc/gnutella.html.

[10] Morpheus, "Morpheus file sharing system," 2002, http://www.musiccity.com/.

[11] KaZaA, "Kazaa file sharing network," 2002, http://www.kazaa.com/.

[12] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like P2P systems scalable," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '03)*, pp. 407–418, ACM Press, Karlsruhe, Germany, August 2003.

[13] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," in *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '04)*, vol. 1, pp. 120–130, Hong Kong, March 2004.

[14] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proceedings of the 16th International Conference on Supercomputing (ICS '02)*, pp. 84–95, ACM Press, New York, NY, USA, June 2002.

[15] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient content location using interest-based locality in peer-to-peer systems," in *Proceedings of the 22nd Annual Joint Conference on the IEEE Computer and Communications Societies (INFOCOM '03)*, vol. 3, pp. 2166–2176, San Francisco, Calif, USA, March-April 2003.

[16] N. B. Chang and M. Liu, "Optimal controlled flooding search in a large wireless network," in *Proceedings of the 3rd International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt '05)*, pp. 229–237, Trentino, Italy, April 2005.

[17] K. Sripanidkulchai, "The popularity of Gnutella queries and its implications on scalability," February 2001, http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html.

[18] C. Gkantsidis, M. Mihail, and A. Saberi, "Hybrid search schemes for unstructured peer-to-peer networks," in *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '05)*, vol. 3, pp. 1526–1537, Miami, Fla, USA, March 2005.

[19] A. Iamnitchi, M. Ripeanu, and I. Foster, "Small-world file-sharing communities," in *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '04)*, vol. 2, pp. 952–963, Hong Kong, March 2004.