Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computer Networks 50 (2006) 2727–2746

Computer Networks

www.elsevier.com/locate/comnet

Small-world overlay P2P networks: Construction, management and handling of dynamic flash crowds

Ken Y.K. Hui [a], John C.S. Lui [a,*], David K.Y. Yau [b]

[a] *Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong*
[b] *Computer Science Department, Purdue University, United States*

Received 28 December 2004; received in revised form 28 July 2005; accepted 6 October 2005
Available online 29 November 2005

Responsible Editor: I. Matta

Abstract

We consider how to construct and maintain a structured overlay P2P network based on the small-world paradigm. Two main attractive properties of a small-world network are (1) a low average hop distance between any two randomly chosen nodes, and (2) a high clustering coefficient. A network with a low average hop distance implies small latency for object lookup. A network with a high clustering coefficient implies the ability to provide efficient object lookup even under heavy object traffic loadings, such as in a flash crowd scenario. We present the SWOP protocol for constructing a small-world overlay P2P network. We compare our system's performance with other structured P2P networks, such as Chord. Whereas the Chord protocol already provides a scalable object lookup latency of $O(\log(N))$, where N is the number of nodes in a P2P network, we show that the SWOP protocol can further improve the object lookup performance. We also take advantage of the high clustering coefficient of a small-world P2P network to design an object replication algorithm that can handle heavy object traffic loading situations. We show that the SWOP network can efficiently deliver popular, possibly dynamic, objects to all the requesting nodes. To the best of our knowledge, ours is the first work that addresses how to handle *dynamic* flash crowds in a structured P2P network.

Keywords: Structured P2P networks; Small world phenomenon; Dynamic flash crowd

1. Introduction

Peer-to-peer networks are distributed information sharing systems with no centralized control. Each node in a P2P network has similar functionalities and plays the role of a server and a client at the same time. This provides immense flexibility for users to perform application-level routing, data posting, and information sharing on the Internet. The first generation P2P systems such as Napster require a centralized directory service. The second generation P2P systems (e.g., Gnutella and Freenet) are unstructured networks, which use a fully distributed approach for file searching. A major problem

[*] Corresponding author. Tel.: +852 2609 8407; fax: +852 2603 5024.
E-mail addresses: ykhui@cse.cuhk.edu.hk (K.Y.K. Hui), cslui@cse.cuhk.edu.hk (J.C.S. Lui), yau@cs.purdue.edu (D.K.Y. Yau).

of the second generation systems is that in searching for an object, the amount of query traffic may be enormous, which leads to network congestion problems.

Over the past few years, researchers have focused on distributed, *structured* P2P networks. Noted work includes Chord, Tapestry, and Pastry [13,17,20]. By applying the approaches of consistent distributed hashing and structured routing, these networks improve the performance of object lookup, while also reducing the amount of query traffic in the network. For example, Chord has a worst case lookup complexity of $O(\log N)$ link traversals, where N is the number of nodes in a Chord network. The implication is that in finding a data object, one does not need to generate an enormous amount of query traffic which may overwhelm network resources.

In this paper, we address two fundamental issues for the design of a distributed, structured P2P network. They are:

- How can one further improve the performance of object lookup?
- How can the network handle heavy user demand for *popular* and *dynamic* objects, as in a flash crowd scenario? For example, during the 9/11 incident, the CNN news server received a huge number of user requests for updated information about the incident.

To address the first technical question, we propose to construct a P2P network under a *small-world* paradigm [7,18]: The average shortest hop distance between two randomly chosen nodes in the network is around six hops. To overcome the second problem, we take advantage of the high clustering coefficient of a small-world network to quickly *self-organize* and *replicate* dynamic, popular objects in the network.

The proposed small-world overlay protocol (SWOP) aims to *construct* a P2P network that exhibits the small-world properties, and *maintain* these properties in the network. We show that the constructed small-world network has better object lookup performance when compared with other structured P2P networks such as Chord. We also illustrate that the constructed network is robust under heavy traffic loading and, on average, provides fast lookup of popular and dynamic objects.

The balance of the paper is as follows. In Section 2, we present a protocol for constructing and maintaining a small-world P2P network. We also describe the object lookup protocol that individual client nodes use to locate any data object. We show that using the SWOP protocol, one can have a lower average object lookup latency, measured in the number of link traversals, compared with Chord. In Section 3, we discuss how the proposed small-world properties can be realized in an existing structured P2P network. In Section 4, we present an algorithm for handling flash crowd heavy traffic loadings. We will consider both static and dynamic flash crowd scenarios. Related work is presented in Section 5. Section 6 concludes.

2. Small-world P2P protocols

In this section, we first provide the necessary background and state some important properties of a small-world network. We then present protocols for constructing and maintaining a small-world P2P network, and a complementary protocol for data lookup. We derive analytically an upper bound for the average object lookup latency. Last, we present experimental results that compare the proposed system with Chord in data lookup efficiency.

The notion of *small-world phenomenon* originated from social science research [10]. It is currently an active research are in physics, computer science, and mathematics [11]. Researchers observed that the small-world phenomenon is pervasive and can be found in diverse settings, such as social communities, biological systems, and communication networks. For example, recent studies show that peer-to-peer networks like *Freenet* may exhibit the small-world properties [19]. Informally, a small-world network can be viewed as a connected graph in which two randomly chosen nodes are connected by just *six degrees of separation*. In other words, the *average shortest distance* between two randomly chosen nodes is approximately six hops. This property implies that information stored at any random node of a small-world network can be located using only a small number of link traversals.

One way to construct a network that exhibits the small-world properties is the following: (1) each node in the network is connected to some neighboring nodes, and (2) each node keeps a small number of links to some randomly chosen *distant* nodes. Links to neighboring nodes are called *cluster links*, whereas links to distant nodes are called *long links*.

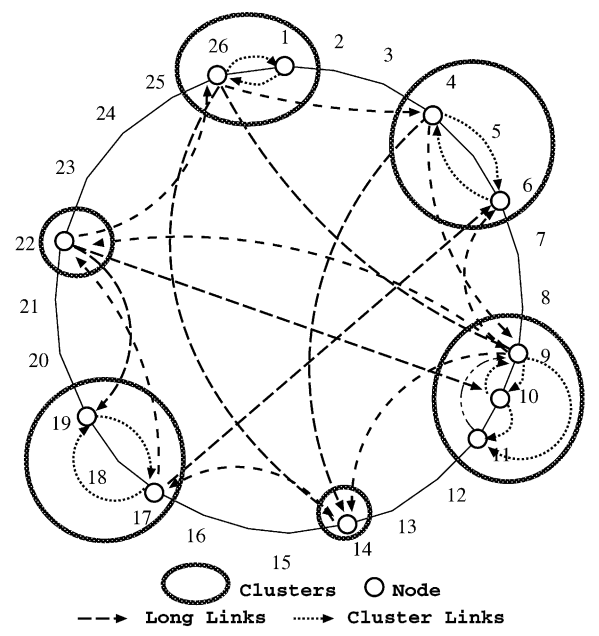

Fig. 1. An example of a small-world network with 11 nodes and six clusters.

Fig. 1 illustrates an example of a small-world network with 11 nodes and six clusters. For example, nodes 9, 10, and 11 form one cluster. They have neighboring links to each other, and node 9 has long links to node 6, 14, and 22.

The two important properties of a small world network are (1) a low average hop count between two random chosen nodes, and (2) a high clustering coefficient. To mathematically define these two properties, let $\mathscr{G} = (V, E)$ denote a connected graph representing the small-world network. There are N vertices in $\mathscr{G}$, where $|V| = N$ and $D(i,j)$ represent the length (in hops) of the *shortest path* between two vertices i, $j \in V$. We have the following definitions:

Definition 1. The average shortest hop count of a graph $\mathscr{G}$, denoted as $\mathscr{H}(\mathscr{G})$, is equal to

$$\mathscr{H}(\mathscr{G}) = \frac{1}{\binom{N}{2}} \sum_{i,j \in V} D(i,j). \tag{1}$$

In other words, $\mathscr{H}(\mathscr{G})$ is the ratio of the sum of all shortest paths between any two nodes in $\mathscr{G}$ and all possible pairwise connections of the connected graph. To define the clustering coefficient, let κ_v be the number of attached links for a node $v \in V$. The *neighborhood* of a vertex v is a set of vertices $\Gamma_v = \{u : D(u,v) = 1\}$.

Definition 2. For a given vertex $v \in V$, let C_v be the local cluster coefficient of v and it is equal to $C_v = |E(\Gamma_v)| \Big/ \binom{\kappa_v}{2}$ where $|E(\Gamma_v)|$ is the operator that counts the total number of links for all vertices in the set Γ_v. The cluster coefficient of a graph $\mathscr{G}$, denoted as $\mathscr{C}(\mathscr{G})$, is equal to

$$\mathscr{C}(\mathscr{G}) = \frac{1}{N} \sum_{v \in V} C_v. \tag{2}$$

In other words, $\mathscr{C}(\mathscr{G})$ measures the degree of compactness of the graph $\mathscr{G}$.

In the following, we first describe protocols for constructing a small-world P2P network and for performing data lookup in the network. We then provide a mathematical analysis on the worst case average number of link traversals for locating an object in the network. Last, we show that when compared with other structured P2P networks, a small-world P2P network achieves a lower number of link traversals in object lookup.

2.1. Overview

The goal of SWOP is to efficiently locate any object in the network, while achieving the ability to access *popular* and *dynamic* objects under heavy traffic loading. The protocol can be implemented as an add-on layer above a structured P2P network layer, without affecting the functionalities the P2P network layer itself.

Let us provide a brief background on structured P2P object lookup protocols. Generally, a structured P2P lookup protocol consists of a consistent hashing function h (e.g., the SHA-1 function) that gives a unique key assignment for each node or object in the system. With the key value, each node can determine its logical position in the system. For example, in Chord, the logical position of a node is a point in a circular key space. For a CAN network, it is a point in a grid. Another property of a structured P2P protocol is its use of a routing table (e.g., the *finger table* in Chord), which speeds up the object lookup process. It is shown that the worst case number of link traversals to locate an object is $O(\log(N))$, where N is the number of network nodes [17].

Each node can insert objects into the structured network using the same consistent hashing function h. Each object o has a unique key value $h(o)$. Let $\mathscr{N}(o)$ be the set of nodes whose key values are greater than or equal to $h(o)$. The node in $\mathcal{N}(0)$ which has the minimum key value is responsible for caching and maintaining the object o.

We use a circular ring to logically represent a small-world P2P network, since the ring helps to illustrate the small-world effects introduced by the SWOP protocol. Let us first define the following parameters for SWOP:

- Cluster size G—the maximum number of nodes within a cluster.
- Cluster distance D—the maximum hash space distance between two adjacent nodes within a cluster.
- Long links k—the number of long links in a cluster.

Our small-world network has two types of node, namely, *head node* and *inner node*, and two types of link, namely *long link* and *cluster link*. Long links connect two different nodes from two different clusters, while cluster links connect two different nodes in the same cluster. Each cluster has *one* head node, which has at most k long links. In addition, the head node has cluster links to all the nodes in its cluster. An inner node has a link to the head node within its cluster, and cluster links to some of the nodes within its cluster. Let m be the average number of clusters in the network. The expected number of connections (or links) for each node is: $(m * k + G * n)/n$, where n is the number of nodes in the SWOP network. (Note that the average number of clusters can be estimated by our CNEP protocols, which will be presented in the next section.)

Given the above setup, an inner node i can communicate with a target node j within its cluster either by a cluster link (provided that nodes i and j are connected), or by sending a message to its head node, who will then forward the message to node j using a cluster link. For communicating with a target node in a different cluster, node i has to first send the message to its corresponding head node, who will then forward the message to node j using the long link closest to j. The message may arrive at some node which is not in the same cluster of j. The procedure will repeat until the message is forwarded to some node in the same cluster as j. Fig. 2 shows an example of small-world overlay P2P network with 11 nodes, six clusters, and SWOP parameters $G = 3$, $D = 2$, and $k = 3$. In the figure, node 1 sends a message to node 17, and the resulting flow of object lookup messages is shown.

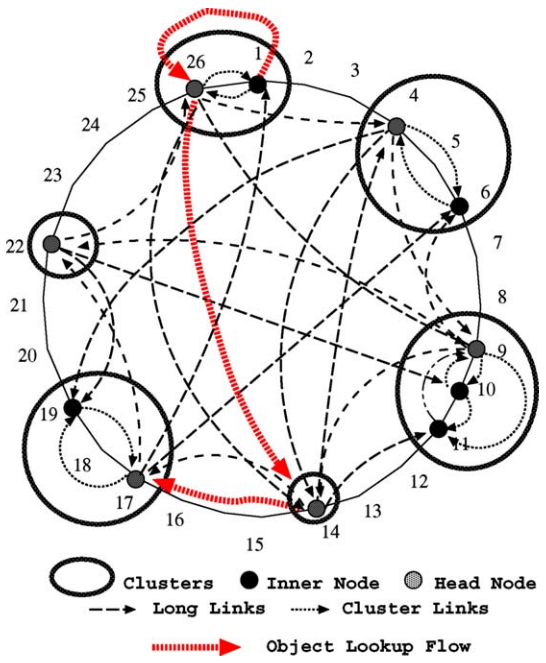

Fig. 2. A SWOP network with six clusters, $G = 3$, $D = 2$, $k = 3$ and the flow of object lookup messages.

We now describe protocols for forming and maintaining a SWOP network. Protocol actions for events like node join, node leave, and node failure will be given.

Join cluster protocol (JCP): If node i wishes to join a SWOP network, it uses the consistent hashing function h to obtain its key $h(i)$. It then creates a link to its predecessor node a and a link to its successor node b in the underlying P2P network. The predecessor node a is an existing node in the network, and its key value $h(a)$ is the largest key value such that $h(a) < h(i)$. The successor node b is an existing node in the network, and its key value $h(b)$ is the smallest key value such that $h(b) > h(i)$. After finding its predecessor and successor nodes, node i executes the *join cluster protocol (JCP)*. The joining node i first determines the *distance* (which is defined based on the hashed key value) between the predecessor and successor nodes. Let d_1 and d_2 be the distances between the joining node i and its predecessor and successor nodes, respectively; i.e., $d_1 = h(i) - h(a)$ and $d_2 = h(b) - h(i)$. The joining node i inquires its predecessor and successor nodes about their respective cluster sizes. If both nodes a and b have a cluster size greater than G (the maximum cluster size), then the joining node i will form a *new* cluster with itself being the only node in the new cluster. Otherwise, i determines which cluster to join depending on the values of d_1 and d_2. If both d_1 and d_2 are greater than D, the joining node i will form a new cluster with itself being the only node in the new cluster. Otherwise, the joining node i joins the predecessor's cluster (respectively, the successor's cluster) if d_1 (respectively, d_2) is less than D and less than d_2 (respectively, d_1).

Next, the joining node i determines whether it should be a head node or an inner node. If the node i forms a new cluster, or if it joins its successor node b which was a head node of that cluster, then the joining node i becomes the new head node of that cluster. Otherwise, the joining node i is an inner node. When the joining node i is an inner node, it needs to create a cluster link to its head node. On the other hand, if the joining node i is the head node, it needs to create cluster links to all the inner nodes within the cluster, and additionally generates k long links to other nodes in different clusters.

In order to achieve a low average shortest hop count $\mathscr{H}(\mathscr{G})$ and high cluster coefficient $\mathscr{C}(\mathscr{G})$, one needs to generate long links based on the following distance dependent probability density function $p(x)$. Let m be the number of clusters in a small-world P2P network. The head node will generate a random variable $\widetilde{X}$, which has the following probability mass function:

$$\text{Prob}[\widetilde{X} = x] = p(x) = \frac{1}{x\ln(m)} \quad \text{where } x \in [1, m]. \tag{3}$$

The above probability mass function is that is biased towards nodes that are far away from the head node. Given the value of x, the head node creates a long link between itself and a random node that is in cluster x. It is important to point out that a long link serves as an *express link* for nodes in two different clusters. The *cluster distance* between two different clusters is defined as the number of long link traversals between these two clusters.

```
Join Cluster Protocol
$n$.join_cluster() {/* For each node $n \in \mathscr{V}$ */
    int predId; // predecessor Id
    int succId; // successor Id
    double $d_1$; // distance for predecessor
    double $d_2$; // distance for successor
    int $g_1$; // group size of predecessor cluster
    int $g_2$; // group size of successor cluster
    /*
      Retrieve underlying
      topology information
    */
    Join underlying DHT network;
    /*
      Prepare for choosing the right
      cluster to join
    */
    Retrieve predecessor Id and successor Id;
    Compute the distances $d_1$ and $d_2$;
    Retrieve cluster group size from predecessor
    and successor clusters;
    /*
      Decision making
      - to choose a cluster to join
    */
    If ($d_1 > D$ and $d_2 > D$)
        $n$ forms a new group
    else if ($d_1 < D$ and $d_2 > D$)
        $n$ joins pred group as a member if $g_1 < G$
        otherwise forms a new one;
    else if ($d_1 > D$ and $d_2 < D$)
        $n$ joins succ group as head if $g_2 < G$
        otherwise forms a new one;
    else
        /* both group size <$D$ */
        $n$ chooses a smaller one to join
    /* Cluster information exchange */
    If ($n$ to be a head) {
        $n$ contacts the old head of retrieved Id;
        Retrieve the whole membership list;
        Regenerate the long links if necessary;
    } else if ($n$ to be a member) {
        $n$ sends a message to the target cluster head;
        $n$ retrieves part of the membership list;
    }
}
```

Leave Cluster Protocol (LCP): When node i leaves the P2P system, it informs its neighbor nodes, which are nodes connected to i by cluster links, or some long links if node i is a head node. Node i will inform its neighbor nodes of its departure by sending a close_connection message and terminate its connection. If a node receives a close_connection message, it will perform the following actions:

(a) If the received message is from a neighbor connected by a long link, the receiving node will close the connection and regenerate a new long link to another node in a different cluster.

(b) If the received message is from a neighbor connected by a cluster link, the receiving node will close the connection and reduce the cluster size by 1.

(c) If the received message is from the head node, the receiving node will close the connection. A node will become a new head node within a cluster if the received message is from its predecessor node which was the old head node of the cluster. The new head node retrieves the short link and the long link routing tables from the new head node, in order to maintain message routing capabilities to all the cluster neighbors.

The pseudocode of LCP is as follows:

```
Leave Cluster Protocol
$n$.leave_cluster() {/* For each node $n \in \mathcal{V}$ */
    Prepare the close_connection message
        (n.nodeId, n.headId);
    Send the close_connection message
        to all its neighbors;
}
/* For cluster head */
$n$.close_connection_receive() {
    while(1) {
        Wait until a close_connection message sent
            by node $n'$;
        If ($n$ and $n'$ are in the same cluster) {
            Remove the cluster link;
            Group size reduced by 1;
            Update information for all members;
        } else if (a node in a different cluster) {
            Remove the long link;
            Regenerate one long link;
        }
    }
}
/* For cluster member */
$n$.close_connection_receive() {
    loop {
        Wait until a close_connection message;
        Close the connection if the node in the
        same cluster;
    }
}
```

Stabilize Cluster Protocol (SCP): In the case of a node failure, the SWOP network uses the SCP to recover and to maintain the proper link connectivities. Periodically,[1] each node sends probes to neighboring nodes to make sure that they are still operational. If a neighboring node is an inner node and it does not respond to the probe, the sending node will simply close the connection to the inner node and inform its head node to reduce the cluster size by one. If a neighboring node is a head node and it does not respond to the probe, the node will perform a search to find a new head node.

In order to facilitate recovery from head node failure, we replicate the cluster links from the head node to its successor nodes. The number of cluster links, G, is a system parameter which can be adjusted (the optimal choice of G is left as the future research work). Suppose that there is a head node failure in a SWOP network, the SCP will first ensure that one of the nodes in the network discovers the failure, and is able to find a new node (which, normally, is the failed head node's successor). The new head node will replace the failed head node by executing the OLP. The number of SWOP messages sent for the above action is bounded by the *worst case* average of $(1 + \log_2(m/2))8\ln(3m)/k$ (the proof of this claim will be presented in Section 2.2).

Second, the new head node sends a probe message to verify the old head node's failure. If the failure is verified (i.e., no probe reply is received from the old head node), the new head node begins to take over the head node's responsibilities. The first duty is to send out a head node declare message to each cluster neighbor, which announces itself as the new head of their cluster. The total number of messages needed in this step is $G + 1$, which includes the confirmation and announcement messages.

Thus, the total SWOP messages sent for recovering from a head node failure is $(1 + \log_2(m/2))8\ln(3m)/k + (G + 1)$.

```
Stabilize Cluster Protocol
/* For each node $n \in \mathcal{V}$ */
$n$.connection_probe() {
    while(1) {
        For all neighbors {
            Send a probe message;
            If it is timeout for this message {
                Close the connection to the target
                node of this message;
```

[1] The time scale is on the order of minutes.

```
                If this node is a head node{
                    Search for a new head to replace
                    this node by using OLP;
                }
            }
        }
        Sleep a few seconds;
    }
}
```

Object Lookup Protocol (OLP): The object lookup protocol is responsible for locate a data object in a SWOP network. The object lookup process proceeds in two phases. In phase one, a node asks its cluster neighbor nodes if they have the target object. If any of these cluster neighbor nodes replies positively, then the lookup process terminates successfully. Otherwise, the object lookup process continues into phase two. In phase two, the node first checks its own status within a cluster. If it is the head node, it forwards the lookup request to its long-link neighbor which has the closest distance to the target object. On the other hand, if it is an inner node, it forwards the lookup request to its head node within the same cluster. The head node will then continue the object lookup process recursively as described above. For example, when the long-link neighbor receives the object lookup request, it checks if it is the owner of the object. If not, it will act as if it is the node initiating the lookup, and the data lookup process is repeated. The lookup process takes place in these two phases alternatively until the data object is found.

To illustrate, consider the example shown in Fig. 2. Suppose that node 1 wishes to look up an object whose key value is 16, and node 17 is responsible for managing and maintaining the object. To begin the object lookup, node 1 starts the phase one lookup process in which it asks its neighbor, node 26, if it has object 16. Since node 26 is not the owner of the object, node 1 will receive a negative reply. Node 1 then starts the phase two lookup process and forwards the lookup request to node 26, which is the head node within the cluster of node 1. Node 26 searches along its long link and forwards the object lookup message to its long link neighbor node 14, which is the closest node to the target object 16. Node 14 checks if it has the object 16. Since it does not have the object, node 14 acts as a lookup initiator node, and starts another phase one lookup. Because node 14 is the only node within the cluster, it begins the phase two lookup and forwards the message to the nearest long link neighbor node 17. When the object lookup request reaches node 17, the object is found and the lookup process terminates.

```
Object Lookup Protocol
/*
    Construct a query packet
    (Item Id, Max Hop),
    run op_phase1
*/
/* For each node $n \in \mathcal{V}$ */
$n$.op_phase1(Query_packet P) {
    Send P to all the cluster member $n$ knows;
    if the query is successful, return ownerId;
    If ($n$ is cluster member) {
        Retrieve the whole list from cluster head;
        Send the query to all the cluster member
        not yet queried;
        if success, return the ownerId;
    }
    Forward the query to head $n$';
    $n$'.op_phase2(P);
}
/* For Cluster Head $\in \mathcal{V}$ */
$n$.op_phase2(Query_packet P) {
    Lookup the nearest long link neighbor for
    P.Id
    Forward P to the neighbor $n$''
    $n$''.op_phase1(P);
}
```

Cluster Number Estimation Protocol (CNEP): In order to generate long links for head nodes, one needs to estimate the average number of clusters m in the SWOP network. In the following, we present the Cluster Number Estimation Protocol (CNEP) which achieves the task.

The CNEP periodically estimates two values, N and G, which are the average number of nodes and the average cluster size in the network. Then, by applying

$$\text{Average number of clusters} = \frac{\text{Average number of nodes}}{\text{Average cluster size}},$$

the average number of clusters can be estimated.

In terms of implementation, CNEP contains two major components: estimating the average number of nodes N in the network, and estimating the average cluster size of the network. To estimate N, we

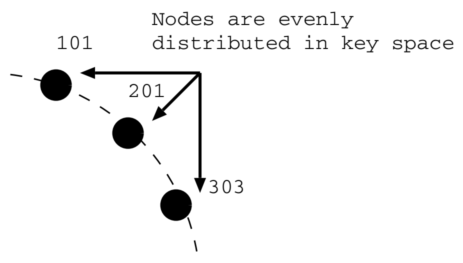

Fig. 3. A snapshot of a SWOP network.

can utilize the distributed hash function's property of evenly assigning IDs. Since nodes are evenly distributed in the hash key space, the average distance of separation between two nodes represents the *compactness* of the underlying network. By comparing the average distance of separation with the total ID space separation, one can proportionally estimate the average number of nodes, N. For instance, a snapshot of the SWOP network is depicted in Fig. 3. Suppose the number of bits for representing the IDs in this network is equal to 10, and the nodes' IDs are as shown in the figure. One can calculate the average number of nodes as:

$$N \approx \frac{\text{hashed space}}{\text{average distance between nodes}} = \frac{2^{10}}{101} \approx 10.$$

To estimate G, the average cluster size of the underlying network, each cluster head node can calculate the average cluster size by collecting a sufficient number of cluster heads' cluster size records. The main challenge of this component is how cluster heads can share their cluster size records. Note that in estimating G, only cluster head nodes are involved. The CNEP can be described as follows: First, each head node keeps track of its own cluster size record and calculates the average distance of separation between two nodes in the cluster. Second, each head node periodically sends a message containing these two pieces of information to its long link neighbors (which are located in different clusters). Third, after each head node has collected a sufficient number of data samples for the two values, the head node can take sample averages for the two values separately. Fourth, each head node calculates the average number of nodes by the average distance of separation obtained in the previous step. Each head node can then estimate the average number of clusters by dividing the average number of nodes by the average number of clusters.

```
Cluster Number Estimation Protocol
/*
  Given Hashed-Space Size HS
*/
/* For Cluster Head ∈ $\mathcal{V}$ */
int $n$.get_ad_between_two_nodes() {
  Retrieve Neighbor nodes' ID;
  Calculate the average distance between
     the nearest nodes;
  Return average distance;
}
$n$.cne_produce() {
  Record start time;
  While(1) {
    Record current cluster size as Ctemp;
    Obtain AD by $n$.get_ad_between_two_
    nodes();
    Send packets to other cluster head nodes
    with
      1. AverageDistance
      2. ClusterSize
    At the same time,
      Receive and store the same kind of pack-
      ets from them;
    Sleep 1 second (can be adjusted)
    If time passes $T$ seconds{
      Compute the average distance in system,
      ADavg;
      Compute the average cluster size, $G$;
      Compute $N$ by HS/ADavg;
      Compute the average cluster number, $C$
      by
        $C = N/G$;
      Construct CNEP packet with value, $C$;
      Reset the start time;
    }
  }
}
/* For each Inner node $n \in \mathcal{V}$ */
$n$.cne_receive() {
  While(1) {
    Wait for Head's CNEP packet;
    Sleep a few seconds;
  }
}
```

2.2. Mathematical analysis

In this section, we derive a theorem based on random analysis to quantify the *worst case* average number of link traversals to locate an object in a SWOP network.

Theorem 1. *Let Y be the non-negative random variable that represents the number of link traversals for object lookup in the SWOP network. We have*:

$$E[Y] \leqslant (1+\log_2(m/2))8\ln(3m)/k, \tag{4}$$

where m and k are the number of clusters and the number of long links of the SWOP network, respectively.

Proof. Let $\mathcal{G}=(V,E)$ be a connected graph representing a SWOP network with $|V|=N$. Let m be the number of clusters in $\mathcal{G}$ and C_i be the number of nodes in cluster i. We have

$$\sum_{i=1}^{m} C_i = N.$$

Let $\overline{C}$ be the average number of nodes within a cluster. Hence, for sufficient large values of N, we have

$$m = N/\overline{C}.$$

We model a connected graph $\mathcal{G}$ with m clusters. We define $d(i,j)$ to be the lattice distance (in the hash key space) between two clusters C_i and C_j, which is equal to $|j-i|$. For two given clusters, C_u and C_v, let us first calculate the probability that there is a long link from C_u to C_v. Since a long link neighbor is chosen according to the probability density function $p(x)=\frac{1}{x\ln(m)}$, the probability that there is a long linkfrom C_u to C_v is $\frac{d(u,v)^{-1}}{\sum_{v\neq u} d(u,v)^{-1}}$. We can express

$$\begin{aligned}\sum_{v\neq u} d(u,v)^{-1} &\leqslant \sum_{j=1}^{m-2}(2)(j^{-1}) = 2\sum_{j=1}^{m-2} j^{-1}\\ &\leqslant 2+2\ln(m-2)\\ &\leqslant 2\ln(3)+2\ln(m-2)\\ &\leqslant 2(\ln(3(m-2))) \leqslant 2\ln(3m).\end{aligned}$$

For the object lookup protocol (OLP) described above, a node forwards an object lookup message from one cluster to another cluster in two phases. These two phases are executed repeatedly until the object is found. When a cluster receives the object lookup message and prepares to forward the message, the cluster is called the *holder* of the current lookup message. We define *cluster movement* as a long link traversal from one cluster to another.

We also define the term *step*, which is the cluster movement that reduces the object lookup distance between the current lookup message's holder and the target object's owner by half. When the object lookup process starts, the process's step is equal to $\log(m/2)$. When the object lookup process ends, the process reaches its last step, $j=0$, in which the distance between the object lookup message and the object owner is at most 2 cluster links apart.

For the object lookup of step j, for $j>0$, the cluster movement from the current lookup message's holder to the target node is greater than 2^j and at most 2^{j+1} cluster movements. Suppose that the object lookup process is moving from step $j+1$ to j. The movement can be seen as the object lookup's need to go from a message holder, which has 2^{j+1} cluster movements to the target node, to its neighbor, which has 2^j cluster movements to the target node. In our analysis, we have to find the probability of such an event. To derive the probability, we first define B_j as the set of nodes having a cluster movement of $2^{j+1}+2^j$, which is less than 2^{j+2} from the target node. Each node in B_j has a probability of at least $(2\ln(3n)2^{j+2})^{-1}$ of being a long link neighbor of the current lookup message's holder. If any of these nodes is the long link neighbor of the current lookup message's holder, the lookup messages can be transmitted to a node with a cluster movement 2^j from the target node. At the same time, one can find the probability that an object lookup can move one step forward from step $j+1$. The message enters B_j with probability at least

$$\frac{2^j}{(2\ln(3m)2^{j+2})} = \frac{1}{8\ln(3m)}.$$

Since there are k links per clusters, the probability of entering B_j is $\frac{k}{8\ln(3m)}$. Let X_j be the total number of cluster movements spent in step j. We have

$$\begin{aligned}E[X_j] &= \sum_{i=1}^{\infty}\Pr[X_j \geqslant i] \leqslant \sum_{i=1}^{\infty}\left(1-\frac{k}{8\ln(3m)}\right)^{i-1}\\ &= 8\ln(3m)/k.\end{aligned}$$

Let Y denote the total number of link traversals spent in the object lookup algorithm. We have

$$Y = \sum_{j=0}^{\log_2(m/2)} X_j.$$

Taking the expectation of both sides, we have $E[Y] \leqslant (1+\log_2(m/2))8\ln(3m)/k$. $\square$

Remark. This theorem is important because we can use it to estimate the proper value of k, so that the SWOP network will have a better object lookup performance than other structured P2P networks. This is considered in the following subsection.

2.3. Experimental results comparing SWOP with other structured P2P networks

In the following, let us compare the object lookup performance between SWOP and Chord, a structured P2P network. We built topology generators for SWOP and Chord, respectively. We conducted simulations to measure the average lookup hop distances between two randomly chosen nodes in the networks, and collected other topological data, including average shortest paths and average clustering coefficients. Our results show that SWOP achieves better performance than its underlying Chord network.

Experiment A.1 (Performance of object lookup): We consider connected graphs with $N = 1000$, 2000, 3000, 4000, and 5000 nodes, respectively. Each node is given one object, for a total of N distinct objects. Each node will perform object lookup 50 times and the target object is randomly chosen among all the available objects. We measure the object lookup performance as the number of link traversals for each network. The SWOP network is configured with parameters $G = 100$, $D = 120{,}000$, and $k = 24$. To get a fair comparison, we set the size of the finger table in Chord to be 24 also. Figs. 4–8 illustrate the probability density functions of the number of lookup link traversals for the two networks, $N = 1000$, 2000, 3000, 4000, and 5000, respectively. Table 1 summarizes the results of object lookup under Chord and SWOP, in which the number of network nodes varies from 1000 to 10000. Observe that the SWOP network has a *lower* average number of lookup link traversals than the Chord network.

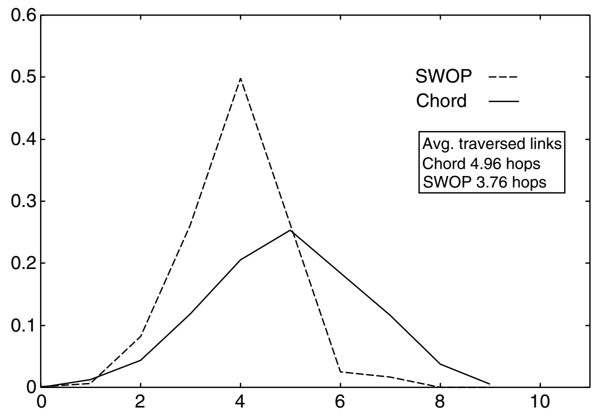

Fig. 4. Probability density function of link traversal for Chord and SWOP with $N = 1000$ nodes.

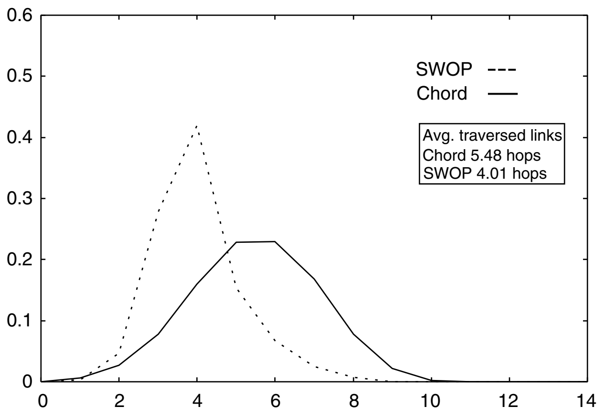

Fig. 5. Probability density function of link traversal for Chord and SWOP with $N = 2000$ nodes.

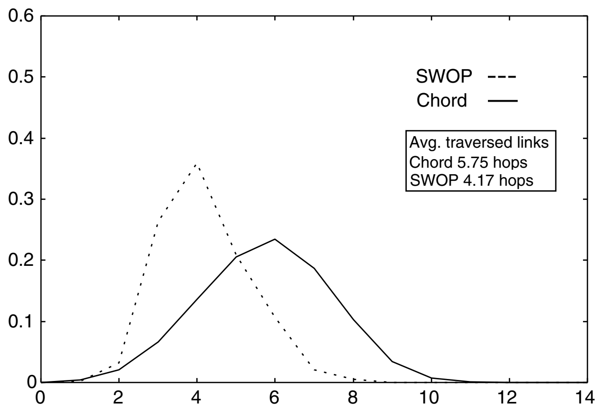

Fig. 6. Probability density function of link traversal for Chord and SWOP with $N = 3000$ nodes.

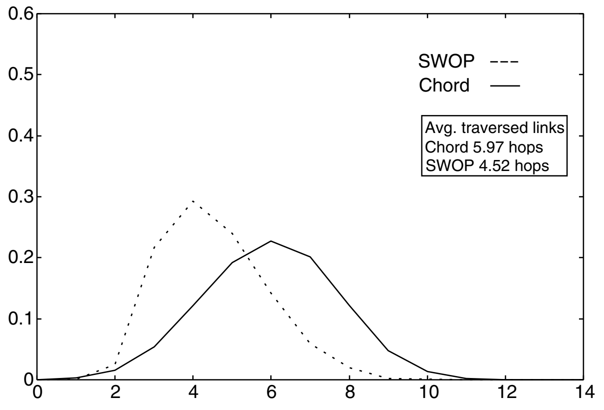

Fig. 7. Probability density function of link traversal for Chord and SWOP with $N = 4000$ nodes.

Experiment A.2 (Effect of object lookup performance under different network sizes and number of

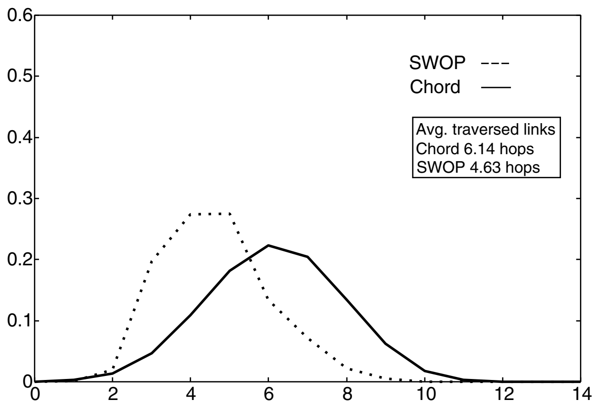

Fig. 8. Probability density function of link traversal for Chord and SWOP with $N = 5000$ nodes.

Table 1
Object Lookup performance (i.e., number of link traversals) and standard deviation of the SWOP and Chord networks, under different values of N

Number of nodes in the system	Avg. traversed links		Standard deviation	
	Chord	SWOP	Chord	SWOP
1000	4.96	3.76	2.41	0.89
2000	5.48	4.01	2.63	1.21
3000	5.75	4.17	2.81	1.33
4000	5.97	4.52	2.87	1.84
5000	6.14	4.63	2.95	1.86
6000	6.26	4.92	3.09	2.31
7000	6.39	5.07	3.10	2.51
8000	6.49	5.32	3.12	2.56
9000	6.53	5.44	3.15	2.91
10000	6.63	5.57	3.20	3.11

long links): We study the object lookup performance further by varying the network size N and the number of long links k for the SWOP network. In order to get a fair comparison, we keep the size of the Chord finger table equal to $\log(N)$, where N is the number of nodes in the network. For SWOP, we vary k, the number of long links, from 4 to 12. Unlike Experiment A.1, we assume that each node has the same number of objects and a node will access any object in the system with equal probability. The results are illustrated in Table 2. They show that (1) by increasing the value of k, one can improve the object lookup performance, and (2) good asymptotic performance can be reached by having a relatively small number of long links k (around 6–8). Since this value is less than $\log(N)$ for a large N, it implies that for SWOP, one achieves a *lower* complexity of connection management, while achieving a better object lookup performance.

Table 2
Object Lookup performance (in number of link traversals) for the SWOP and Chord networks under different values of N and k

N:	Chord	SWOP				
		$k = 4$	$k = 6$	$k = 8$	$k = 10$	$k = 12$
1000	6.0	6.0	5.0	4.4	4.2	3.7
2000	6.5	5.9	4.9	4.4	4.1	3.8
3000	6.7	6.0	4.9	4.3	4.1	3.8
4000	6.9	6.3	5.2	4.5	4.0	3.9
5000	7.1	6.2	5.5	4.6	4.3	4.0

Table 3
Average clustering coefficient

N: # of nodes	Chord	SWOP
1000	0.288182	0.560587
2000	0.260332	0.649660
3000	0.250452	0.684012
4000	0.245469	0.704523
5000	0.240776	0.716463

Experiment A.3 (Comparison of Clustering Coefficient): The above results show that the average object lookup performance of SWOP is better than Chord. The next issue we want to investigate is their corresponding average cluster coefficients. Again, a high clustering coefficient implies an increased ability to handle heavy traffic loadings. We vary the number of nodes in the overlay network in this experiment. For Chord, each node has a finger table size equal to $\log(N)$. For a fair comparison, we keep the number of long links for the SWOP network to be also $k = \log(N)$. The clustering coefficient is computed based on Eq. (2). The results are shown in Table 3. We observe that the SWOP network has a *higher clustering coefficient*, showing that the SWOP network is more efficient in handling heavy object lookup traffic. We will explore this feature in detail in the following section.

3. Realizing the small world paradigm in an existing structured P2P networks

In general, SWOP can be applied on top of different structured P2P networks, such as CAN, Pastry, and Tapestry. A small world network layer can be defined above the underlying structured networks which share essential characteristics.

A structured P2P network can be divided into three major components: a unique key assignment scheme, a characteristic routing table, and an effective routing mechanism. SWOP can be readily implemented as a layer over these components. First, we formulate a "cluster" based on the key assignment scheme. We group similar keys together to form a cluster, which can enhance communications between nodes within a cluster, and provide a means to resolve the high traffic loading problem. The grouping procedure can be carried out for all of CAN, Pastry, and Tapestry, which have similar key assignment schemes. Based on the grouping criteria (e.g., group size and node separation distance), each node can decide a suitable cluster to join and select an appropriate cluster head node. In our description of SWOP, the cluster head is selected using the smallest ID. In general, however, it can be the smallest ID or vector. The different structured P2P networks can apply the selection criteria in choosing a suitable cluster head to represent a cluster.

Second, one needs to construct the small world routing table. To realize this, we restrict each inner node, a cluster member, to connect to nodes within the cluster only , and we allow each head node to connect to nodes in other clusters according to the harmonic distribution in Eq. (3). The distribution aims to provide the high degree hubs representing a small world network. For the different structured P2P networks, provided that each node has the cluster head information, the node can maintain a list of short links. Moreover, a P2P network can execute the CNEP protocol as described in the previous section, so that nodes can estimate the average number of clusters within a network, and the nodes can connect to other clusters based on Eq. (3) using long links.

Third, after forming the small world routing tables, we can use the OLP to perform object lookup directly because the corresponding algorithms are based on short links within a cluster, and long links connecting different cluster. These links are properly maintained by the SWOP protocol.

Currently, our small world layer forms clusters of nodes by the nodes' similar IDs. This scheme can be modified to include other application specific criteria. For instance, one can group nodes based on similar communication delays or semantics of contents they hold, etc. Communication delays between nodes can be estimated by sending probe packets. Such grouping can help reduce the delay of network file transfers, for example. For content semantics, one can assign a semantic vector, based on user interests, to each node in the network. Such grouping may reduce the object lookup latency because users will then have a tendency to search for interesting contents within a cluster first.

4. Protocols for handling flash crowd

In this section, we address how to handle object access under flash crowd, heavy traffic loading scenarios. One way to deal with flash crowds is to replicate popular objects among nodes. This way, one can spread out access to the popular objects and avoid overwhelming any single source node. However, one has to address the following technical issues:

- The replication process must be properly driven by a high demand traffic. Otherwise, one may maliciously replicate many objects in the network.
- In a structured P2P network, object lookup is by the object's key value and only *one* node manages that object. How can the protocol be extended so that more than one node can store the popular object?

We divide the study of the flash crowd scenario into two cases: *static* and *dynamic*. A static flash crowd is one in which the popular object in question remains unchanged after it first appears, say a newly published book or a released video. A dynamic flash crowd is one in which the contents of the popular object may change over time after it first appears, say a frequently updated news article. We first describe algorithms to handle the static flash crowd problem. We then extend the algorithms to handle the dynamic flash crowd by adding notifications of object updates, which will allow the popular objects to be replicated efficiently.

4.1. Static flash crowd

To avoid the source node of a popular object from being overwhelmed by high traffic demand, the source node needs to replicate the object among other nodes. Note that the object replication has to be *demand driven*, or else an attacker can maliciously replicate many useless objects all over the network nodes. We now describe a checking function for the access rate, which estimates the rate at which an object is being accessed in the system.

Note that the function can be applied not to a SWOP network, but also to other structured P2P networks like Chord.

The access rate of an object is estimated by the `access_checking` function specified below. By estimating the access rate, the function decides whether the object should be replicated or not. Each node in a P2P system calls the `access_checking()` function for its managed objects every PERIOD units of time. At all times, each node also tracks the number of accesses to its managed objects in the array access_count[]. The pseudocode of the `access_checking()` function is shown as follows:

```
Access checking
$n$.access_checking() /* For each node $n \in \mathscr{V}$ */
1. double access_rate;
2. for ($\forall i \in$objects managed by node $n$) {
3.    /*
      compute access rate for object $i$.
      */
4.    access_rate[i] = access_count[i] / PERIOD;
5.    /* reset access count */
6.    access_count[i] = 0;
7.  }
8.  }
```

In the following, we describe algorithms of replicating static popular objects for the Chord network and for the SWOP P2P network.

Static-Chord Algorithm: Assuming that a node in a Chord network can process up to λ_t requests per second with reasonable performance. Whenever a source node discovers that the request rate for an object is λ_1, where $\lambda_1 > \lambda_t$, the source node starts the replication process by pushing the popular object to *all* of its neighboring nodes, i.e., all the nodes listed in its finger table. The receiving nodes will cache the popular object for T_1 time units. For a node that caches the popular object, if the node receives a lookup rate of λ_2 for the object, where $\lambda_2 > \lambda_t$, the node will in turn push its cached object to *all* the neighboring nodes. The motivations for such an algorithm are: (1) the replication process of a popular object is purely demand driven, (2) a popular object will be replicated at other nodes in the Chord network, so that it can become better accessible.

We use the above approach to handle flash crowds in a Chord network, because the network nodes are evenly distributed, and the fingers of the nodes are evenly spread in the system. By pushing the popular objects to the evenly distributed finger nodes, one also distributes the cached object evenly in the network. The even distribution can increase the hit probability of a randomly generated query. We therefore choose the above algorithm for object replication in Chord, so as to provide a fair comparison between SWOP and Chord.

Implementation of replication process in Chord: The pseudocode of the replication algorithm is described as follows. If the access rate of an object is higher than λ_t, the function `push(int objectId)` will be called. Popular objects will be pushed and replicated at all the nodes indicated in the finger table of a Chord network.

```
Push – Chord version
$n$.push(int objectId)
1.   for ($\forall$neighboring nodes $\in$ finger_list) {
2.       send the popular object to
          each neighboring node;
3.       each neighboring node caches
          the received object;
4.  }
```

Static-SWOP Algorithm: Note that a small-world P2P network exhibits a high clustering coefficient. We use this property to achieve a lower replication time and reduce the number of link traversals in obtaining a popular object. We also assume that a node in the SWOP network can process up to λ_t requests per second. Whenever a source node receives a request rate of λ_1 for an object, where $\lambda_1 > \lambda_t$, the source node will start the replication process by pushing the popular object to its neighbors, namely all the nodes connected by *long links* in its cluster. All these receiving nodes will cache the popular object for T_1 time units. Any node that wishes to access the popular object uses the *Object Lookup Protocol* (OLP) described in Section 2 for the access. If the popular object is cached by any node in its cluster, the requesting node can access the popular object quickly. For a node that caches the popular object, if the lookup rate for the cached object is higher than λ_2, the node will in turn push the cached object to all the nodes connected by long links in its cluster. Replication of popular objects via long links is motivated by the desire to propagate information quickly to distant clusters, so that nodes within those clusters can easily access the popular object.

Implementation of replication process in a SWOP network: The pseudocode of the replication algorithm is illustrated as follows. We define *long_links_list* as a set of nodes in different clusters which are directly connected via long links. If the access rate is higher than λ_t, the function `push(int objectId)` will be called. The main idea of the *push* function is to push the popular object from one cluster c_i to another cluster c_j via c_i's long links. The popular object will be cached among the long link neighbors. Since each long link neighbor belongs to a different cluster, pushing the popular object to the long link neighbors replicates that object in each of the clusters. Under SWOP, the OLP can achieve efficient lookup within a cluster. The detailed pseudocode for the *push* function is as follows:

```
Push – SWOP version
n.push(int objectId)
1.  if (n.cluster_status == INNER) {
      /* node n is an inner node */
2.      node n searches its cluster head;
3.      node n sends objectId to it's cluster head;
4.      cluster head registers the replicated item in
        item list;
5.      node n transfers the object to cluster head,
        but the cluster head does not store the
        object in its cache if it did not request
        for it;
6.      cluster head calls the push function again;
7.  } else {/* node n is the cluster head */
8.     /* node n replicates the object.*/
9.     for (∀neighbors ∈long_links_list) {
10.        cluster head transfer the popular object
           to
           each neighbor;
11.        each neighboring node caches the
           received object;
12.    }
13. }
```

4.2. Dynamic flash crowd

To handle dynamic flash crowds, a protocol message, the *update message*, and an *update counter* for each object's version number should be added to the static SWOP push algorithm.

Assume that a popular object has been replicated at exactly one node in each SWOP network cluster, using the static algorithm. Each node caching the object now marks its copy as the original, is version 0. If an updated version, say version 1, is injected into the network by the source node, we only need to use a lightweight notification to inform all the caching nodes of the version update. Notice that the notification process can be carried out efficiently following the paths of the static replication algorithm.

Whenever the source node generates an updated object, additional tasks have to be carried out in the dynamic algorithm. First, an update message is sent by (1) the source node, and (2) each caching node of the updated object. In general, there are two types of message update: (i) those sent to all the cluster neighbors, and (ii) those sent to all the long link neighbors of each cluster head. The first type of message involves the cluster neighbors only, reminding them of looking up the latest version of the popular object. The second type of message involves the long link neighbors. It reminds those neighbors of the new update and sends the updated object to them. Hence, an version update will be propagated from one cluster to another. The second type of message update requires cooperation between the head node and the sending node (if the sending node is not a head node). If a node receives an update message, but does not have a cached copy of the popular object, the node will use the OLP protocol described in Section 2 to retrieve the updated object.

4.3. Mathematical analysis of object replication time

In this section, we mathematically derive the average time, the required memory by each node, and the required bandwidth for replicating a popular object in all the clusters in a SWOP network. We model the spreading process by a continuous time Markov chain (CTMC).

The average replication time: Let $\mathscr{M}$ be the CTMC representing the replication dynamics of a SWOP network. The network has N nodes. The CTMC $\mathscr{M}$ has a state space of $\mathscr{S}$ such that $\mathscr{S} = \{1, 2, \ldots, m\}$, where m is the number of clusters in the network. State i in $\mathscr{M}$ means that the popular object has already been replicated in i nodes in the SWOP network. Since only the source node has the popular object initially, the CTMC $\mathscr{M}$ is in state 1 initially. Define x_1 to be the number of requests needed in time period τ for the source node to replicate the popular object at all its long link neighbors (i.e., $\lambda_1 = x_1/\tau$). After receiving the popular object, the neighboring nodes become the replicated nodes. Similarly, define x_2 to be the number of object requests needed for a replicated node to start replicating the popular object at all its long link neighbors. Let λ be the request rate for the popular object in a SWOP network. The rate of replicating a popular object from the source node is λ_s, which is equal to

$$\lambda_s = (N-1)\lambda\left[1 - \sum_{j=0}^{x_1-1} \frac{e^{-(N-1)\lambda\tau}((N-1)\lambda\tau)^j}{j!}\right].$$

Similarly, let $\overline{C}$ represents the average number of nodes within a cluster. The rate of replicating a popular object from a replicated node is λ_r, which is equal to:

$$\lambda_r = (\overline{C}-1)\lambda\left[1 - \sum_{j=0}^{x_2-1} \frac{e^{-(\overline{C}-1)\lambda\tau}((\overline{C}-1)\lambda\tau)^j}{j!}\right].$$

Let $\boldsymbol{Q}$ be the infinitesimal rate matrix of $\mathscr{M}$. We denote an element in $\boldsymbol{Q}$ as $q_{i,j}$, which is the transition rate from state i to state j, for $i, j \in \{1,2,\ldots,m\}$. Assume that a replicated node will cache the object for an average time of $1/\mu$. Let $v_1 = \lambda_s$ and $v_i = \lambda_r$ for $i \in \{2,\ldots,m\}$. The transition rate matrix of Q can be specified by the following transition events:

Object deletion event:

$$q_{i,i-1} = i\mu \quad \text{for } 1 < i \leqslant m \tag{5}$$

Object replication event: we have two cases to consider:

Case 1: for state $i \in \mathscr{S}$, if $(i + ik) \leqslant m$:

$$q_{i,j} = \begin{cases} \left(\frac{m-i}{m}\right)^{j-i}\left(\frac{i}{m}\right)^{ik-(j-i)} v_i & \text{if } k \leqslant j \leqslant (i+ik) \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Case 2: for state $i \in \mathscr{S}$, if $(i + ik) > m$:

$$q_{i,j} = \begin{cases} \sum\limits_{x=m}^{(i+ik)} \left(\frac{m-i}{m}\right)^{m-i}\left(\frac{i}{m}\right)^{x-(m-i)} v_i & \text{if } j = m \\ \left(\frac{m-i}{m}\right)^{j-i}\left(\frac{i}{m}\right)^{ik-(j-i)} v_i & \text{if } k \leqslant j \leqslant (i+ik) \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Once the rate matrix $\boldsymbol{Q}$ is specified, one can derive the average time to replicate a popular object in all the clusters of a SWOP network using the theory of fundamental matrix in Markov chain analysis [2,12].

We first transform the rate matrix $\boldsymbol{Q}$ to a discrete time transition probability matrix $\boldsymbol{P}$ by using the uniformization technique [2,12] such that $\boldsymbol{P} = \boldsymbol{I} + \boldsymbol{Q}/\Lambda$, where $\boldsymbol{I}$ is an identity matrix and Λ is a maximum absolute value for all the entries in $\boldsymbol{Q}$. Since we want to find the average time taken to replicate a popular object, we can consider the state m in $\mathscr{M}$ as an absorbing state; i.e., it is the state in which the popular object has been replicated in all the SWOP network clusters. Let $\boldsymbol{P}_c$ be the square matrix equal to $\boldsymbol{P}$ except that we remove the last row and column (i.e., the absorbing state m) from $\boldsymbol{P}$. The fundamental matrix $\boldsymbol{M}$ can be calculated using

$$\boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{P}_c)^{-1} = \sum_{i=0}^{\infty} (\boldsymbol{P}_c)^i. \tag{8}$$

Let $E[T_c]$ be the average time to replicate the popular object in all the SWOP network clusters, given that only one node (or cluster) has the popular object at time $t = 0$. We can compute $E[T_c]$ by

$$E[T_c] = \left(\frac{1}{\Lambda}\right) e_1 \boldsymbol{M}\boldsymbol{1}^{\mathrm{T}} \tag{9}$$

where e_1 is a row vector of zeroes except that the first entry is one and $\boldsymbol{1}$ is a row vector of all ones.

To quantify the memory requirement, one can measure the expected number of *keys* stored in each node in the SWOP network. The keys can be divided into two categories: topological keys and item keys. For topological keys, an upper bound on their numbers can be calculated by considering the head nodes in the SWOP network, since these nodes have both long links and short links in the topology construction. The value of the upper bound is $(\log N + 2) + (G + k)$, which includes the topological keys for the underlying DHT and the SWOP overlay. For item keys, their upper bound depends on the number of popular objects in the network during a flash crowd period. We expect that the final result of executing the flash crowd handling protocol is that each node has one copy of every popular object. Hence, the bound on memory required for item keys for each node is the number of popular objects in the system. In summary, if there are i popular objects in the SWOP network, the bound on memory required for each node is $(\log N + 2) + (G + k) + i$.

The bandwidth requirement in the network can be measured by the number of messages generated from each node in the network. Without the flash crowd handling protocol, the bandwidth requirement is $n*\ (1 + \log_2(m/2))8\ln(3m)/k$ messages, which means that n nodes apply the OLP to retrieve the popular object. On the other hand, with the flash crowd handling protocol, a caching node can push the popular object to its k long link neighbors. As a result, those nodes inside the k clusters can use the OLP to lookup the popular object by two messages (i.e., within two hops) instead of $(1 + \log_2(m/2))8\ln(3m)/k$ messages. To represent the dynamics using the flash crowd handling protocol, we assume that the system has replicated the popular object N_r times in c clusters, where $c < m$ (m is the average number of clusters in the SWOP network), and each server needs r requests to trigger the replication process. The number of messages generated in the system is bounded by $N_r r(1 + \log_2(m/2))8\ln(3m)/k + 2cG$, where G is the number of cluster members as defined in the previous sections.

4.4. Experimental results for handling flash crowds

In this section, we present experimental results comparing the performance of replicating a popular object in Chord versus in SWOP. We use the topology generator developed in Section 2 to create a small-world P2P network. We then use a discrete event simulator to generate workloads that simulate flash crowd scenarios. In this study, we use a 24-bit hash space.

There are $N = 2000$ nodes in the experimental network. In order to get a fair comparison, each node in the P2P network (both for Chord and SWOP) has no more than 11 long link neighbors. This implies that, in the Chord network, the size of the finger table is 11, and in the SWOP network, the length of the long link neighbor list is $k = 11$. We generate a popular object whose key value is randomly chosen from the 24-bit hash space, and we inject this popular object into the P2P network. Each node in the network generates requests for the popular object at a Poisson arrival rate of λ. A node processes received lookup requests at a Poisson service rate of μ, which is normalized to 1 request/second. If a request arrives to the source node and the source node is busy, that request will be queued. Each node has a finite queue size for storing incoming requests. If the request arrives to a full queue, the request will be dropped.

We perform experiments and measure the *effective replication time* of SWOP and Chord. The effective replication time is measured in the *number of successful requests*. The number of successful requests at time t is defined as the number of nodes that can successfully access the popular object by time t. Note that an object lookup request from a requesting node may fail because the request may be dropped by intermediate nodes or by the source node due to finite queueing capacity. In our study, a request is considered successful only if the requested object can be accessed by time t.

Experiment B.1: Comparison between Chord and SWOP: This is a basic comparison for the static and the dynamic flash crowd scenarios between the Chord and the SWOP networks. We fix the per node average request arrival rate to be $\lambda = 0.003$ requests/second. Under the dynamic flash crowd scenario, the source node which manages the popular object will change the version of the popular object at time $t = 25,50$, and 75 respectively.

Static results: Fig. 9 shows the numbers of successful requests processed as a function of time. The results show that the SWOP network has a much better performance in object replication than Chord. For example, at time $\geqslant 20$, most of the requests for the popular object are successful under SWOP. However, only around 6% of the requests are successful under Chord.

Dynamic results: Fig. 10 illustrates the performance of the Chord and SWOP networks under the dynamic flash crowd situation. It uses the same plot settings as the static results. Since the contents of the popular object are updated at times $t = 25$, 50, and 75, we consider a request to be successful if and only if the most updated version of the requested object can be accessed. Fig. 10 shows that the number of successful requests achieved for the SWOP network is much higher than that for Chord.

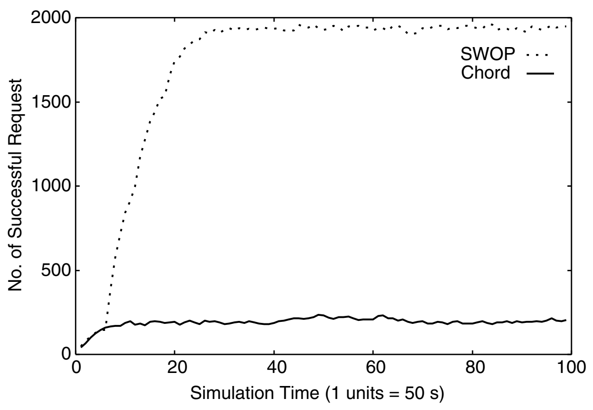

Fig. 9. Comparison of the number of successful requests processed for a popular object in a static flash crowd scenario, with $N = 2000$ nodes.

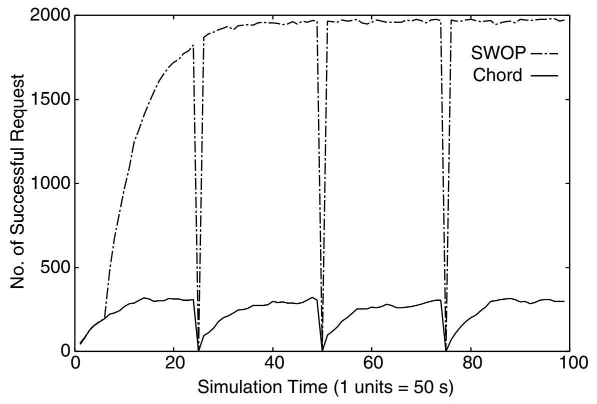

Fig. 10. Comparison of the number of successful requests in a dynamic flash crowd scenario, with $N = 2000$ nodes. Object changes version at $t = 25, 50, 75$.

Moreover, when the object changes at times $t = 25$, 50, and 75, the SWOP network can quickly notify other nodes of the update, such that those nodes can eventually access the most recent version of the popular object. In comparison, the Chord network is not as effective as SWOP in replicating the object. We conclude that SWOP achieves much better performance in a dynamic flash crowd situation.

Experiment B.2: Comparison of queue size: In this experiment, we investigate the effect of different queue sizes on the number of successful requests processed. We keep the same experimental parameters as in Experiment B.1, except that we vary the queue size of each node from 20 to 50. The results are shown in Fig. 11. From the figure, observe that the SWOP network is not too sensitive to the node's queue size. Even when one uses a small queue size of 20, the SWOP network can maintain a good performance and a high successful request rate. Thus, system resources for each node in the SWOP network can be reduced without affecting the performance during flash crowds. Last, the figure illustrates that the SWOP network performs much better than the Chord network under the flash crowd scenarios.

Experiment B.3: Variation of object request rate: In this experiment, we further examine the performance of handling dynamic flash crowds when the object request rate is varied. We keep the same experimental settings as in Experiment B.1. We vary the per node object request rate λ from 0.001 to 0.005. The results are shown in Fig. 12. They show that SWOP performs the best at he arrival rate of 0.003. The reason is that when the request rate is greater than 0.003, the aggregate request rate is higher than the service rate of individual nodes that cache the popular object within a cluster. On the other hand, when the request rate is smaller than 0.003, the aggregate request rate is small enough to achieve a very high number of successful requests. To handle the higher request rates, we may need to dynamically control the cluster size (e.g., to make a smaller cluster) in order to achieve a higher rate of successful requests. How to perform the dynamic control is a subject for further research.

Experiment B.4: Variations of number of long link neighbors (k): This experiment investigates the performance when we vary the number of long link neighbors k in the network. We vary the topology of the SWOP network such that each node has $k = 7$, 9, or 11 long links. In this experiment, we

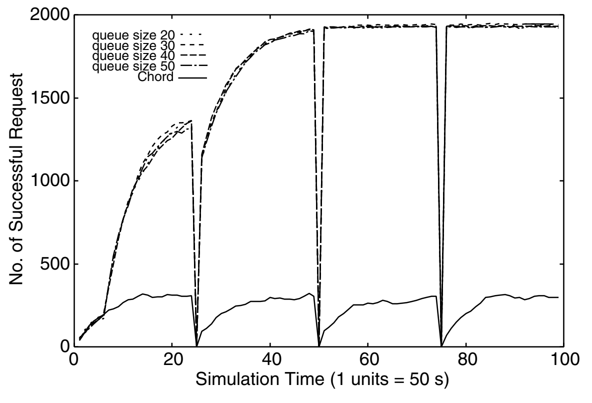

Fig. 11. Effect of queue size on the number of successful requests for the SWOP and the Chord networks. Object changes version at $t = 25, 50, 75$.

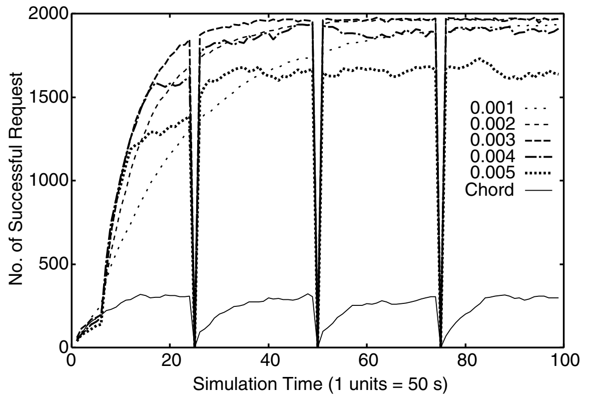

Fig. 12. Effects of varying the per node request rate λ on the number of successful requests for the SWOP and the Chord networks. Object changes version at $t = 25, 50, 75$.

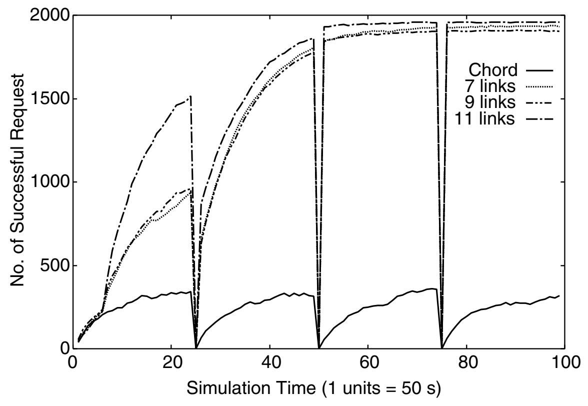

Fig. 13. Effects of varying the of number of long links k for the SWOP and the Chord networks. Object changes version at $t = 25$, 50, 75.

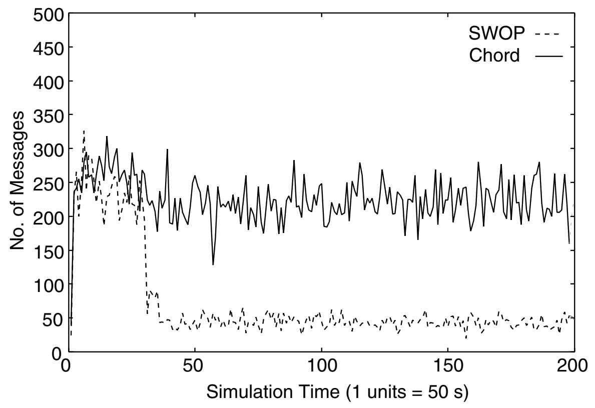

Fig. 14. Effect of the traffic loading for static flash crowd of SWOP and the Chord network.

keep the finger table size of the Chord network to be 11. The results are shown in Fig. 13, which show that even when the number of long link neighbors in SWOP is reduced, the number of successful requests under SWOP is *significantly better* than that of the Chord network. For example, observe that when SWOP has $k = 7$ long links, the number of successful requests at the beginning is less than the cases of $k = 9$ or $k = 11$ long links. However, the number of successful requests quickly catches up. Also, the replication rate is much better than Chord. The results imply that the SWOP network only needs to manage a small number of long links compared with Chord, but it still has highly robust performance under dynamic flash crowds.

Experiment B.4: Amount of traffic generation: In this experiment, we examine the effects of traffic loadings, in terms of the number of messages generated in the network to handle flash crowds. We compare the difference in traffic loadings between Chord and SWOP under the same flash crowd scenario.

We record the number of messages generated in an experiment. For both SWOP and Chord, we set $N = 2000$ nodes, $\lambda = 0.003$ requests/second for each node, and $\mu = 1$ for the source node and each object caching node. At the start of the simulation, a popular object is generated. After that, each node generates requests at a rate of λ to look up the popular object. In this simulation, the total number of generated messages over each time interval is counted, and the simulation ends at time 200.

The results are shown in Fig. 14. They show that, in the static flash crowd scenario, the traffic loading of the SWOP network decreases substantially, while the Chord network still suffers a high traffic loading. We conclude that the SWOP network has a better performance in handling traffic bursts because it makes use of the clustering property to reduce the lookup distances of requests.

5. Related work

The small world phenomenon was first observed by Milgram [10], where the interesting six degrees of separation in a social network is described. Kleinberg [6,7] provides a theoretical framework for analyzing graphs with small-world properties. In [3], the authors study the broadcast mechanism of communication in a small-world network. In [1], the authors present the conditions for switching from a regular lattice to a small-world graph. In [5], the authors propose a system architecture to handle the replication and version updates of popular files. Compared with their work, our system makes use of the small world structure to reduce the spreading distance of popular files significantly. Moreover, our system is built on top of a DHT network, thus preserving many attractive properties provided by DHT. In [19], the authors propose a scheme for storing data in an *unstructured* P2P network, such as Freenet, such that the P2P network exhibits some of the small-world properties. Compared with their work, our solution focuses on structured networks which use consistent hashing. Moreover, we provide techniques for resolving the dynamic flash crowd problem, which are not applicable in an unstructured network because an object can be stored in many different nodes. In [15], the authors propose a small world overlay structure on top of an *unstruc-*

tured and decentralized peer-to-peer network, such as Gnutella. In contrast, we address structured and decentralized peer-to-peer networks, such as Chord. In [4], the authors propose to form a small-world P2P network for scientific communities. However, the details of creating and managing such a network are not clearly discussed.

Recent research work on structured P2P networks can be found in [9,13,17,20]. The main feature of this body of work is in providing some form of topological structure for an overlay, so as to efficiently look up a data object without generating a lot of query traffic. Compared with their work, the SWOP protocol provides better performance in object lookup. We also provide an efficient way to replicate popular and dynamic objects, thereby resolving the general flash crowd problem. Ulysses [8] is a structured P2P based on the concept of butterfly network and a shortcut intuition. The proposed P2P protocol achieves a low number of link traversals for performing an object lookup. However, the performance depends on a stable topology. If a query is routed between nodes which are in the middle of a join or leave operation, the performance is not clear. Moreover, their work considers only moderate traffic loadings, and does not address flash crowd issues. In comparison, SWOP can achieve a low number of link traversals in object lookup, while exploiting the high cluster coefficient in handling dynamic flash crowds. Based on Chord, the Cooperative File System has been proposed as a new peer-to-peer read only storage system. The system helps alleviate flash crowd problems, but it has high storage overhead in storing the file information. In comparison, SWOP has low storage overhead of $O(G+k)$ routing table entries in handling the flash crowd problem.

In [16], the authors propose a protocol to handle the static flash crowd problem in a P2P network. An elegant analysis of the static flash crowd problem is presented in [14]. Our work more generally solves the dynamic flash crowd problem, which is important because many popular network contents are dynamic in nature.

6. Conclusion

Small-world network is an active area of research in the social sciences, physics, and mathematics, among other fields of study. A small-world graph has two important properties: a low average hop distance between two randomly chosen nodes, and a high clustering coefficient. In this paper, we have presented a set of protocols for creating and managing a small-world structured P2P network. We show that the proposed network does achieve both the small average hop distance and high clustering coefficient properties. We demonstrate how the low average hop distance between two random chosen nodes can reduce the number of link traversals in object lookup. A high clustering coefficient, on the other hand, is important in handling the flash crowd problem. In addition, we have designed and evaluated a protocol for replicating popular, possibly dynamic, objects in response to user demands. We have reported extensive experimental results that compare the performance of our small-world P2P network with the Chord structured P2P network. We show that the small-world network achieves a lower object lookup latency and has robust performance under heavy traffic loadings.

Acknowledgements

We would like to thank the anonymous reviewers for their insightful comments. The work of John C.S. Lui was supported in part by an RGC Grant. David K.Y. Yau was supported in part by the National Science Foundation under grant numbers CCR-9875742 (CAREER) and CNS-0305496.

References

[1] A. Barrat, M. Weight, On the properties of small-world network models, The Eur. Phys. J. 13 (2000) 547–560.
[2] U. Bhat, Elements of Applied Stochastic Processes, John Wiley & Son, New York, 1984.
[3] F. Comellas, M. Mitjana, Broadcasting in small-world communication networks, Proc. 9th Int. Coll. Struct. Inform. Commun. Complexity 13 (2002) 73–85.
[4] A. Iamnitchi, M. Ripeanu, I. Foster. Locating data in (small-world?) peer-to-peer scientific collaborations, in: First International Workshop on Peer-to-Peer Systems, Cambridge, MA, March, 2002.
[5] B.W. Ian Clarke, O. Sandberg, T.W. Hong, Freenet: a distributed anonymous information storage and retrieval system, Lecture Notes Comput. Sci. 2009 (2001) 46+.
[6] J. Kleinberg, Navigation in a small-world, Nature (2000) 406.
[7] J. Kleinberg, The small-world phenomenon: an algorithmic perspective. cornell computer science technical report 99-1776, 2000.
[8] A. Kumar, S. Merugu, J.J. Xu, X. Yu, Ulysses: a robust, low-diameter, low-latency peer-to-peer network.
[9] D. Malkhi, M. Naor, D. Ratajczak, Viceroy: A scalable and dynamic emulation of the butterfly.
[10] S. Milgram, The small world problem, Psychol. Today 2 (1967) 60–67.

[11] M.E.J. Newman, I. Jensen, R.M. Ziff, Percolation and epidemics in a two-dimensional small world, Phys. Rev. E 65 (2002).

[12] E. Parzen, Stochastic Processes, Holden-Day, San Francisco, California, 1962.

[13] A. Rowstron, P. Druschel, Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems, in: 18th IFIP/ACM Int. Conference on Distributed System Platforms, November 2001, pp. 329–350.

[14] D. Rubenstein, S. Sahu, An analysis of a simple P2P protocol for flash crowd document retrieval, Columbia University, Technical report EE011109-1, November, 2001.

[15] S.S. Shashidhar Merugu, E. Zegura, Adding structure to unstructured peer-to-peer networks: the use of small-world graphs, J. Parallel Distributed Comput. 65 (2) (2005) 142–153.

[16] A. Stavrou, D. Rubenstein, S. Sahu. A lightweight, robust P2P system to handle flash crowds, IEEE ICNP, November, 2002.

[17] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Chord: a scalable Peer-To-Peer lookup service for internet applications, pp. 149–160.

[18] D. Watts, Small-Worlds: The Dynamics of Networks Between Order and Randomness, Princeton University Press, 1999.

[19] H. Zhang, A. Goel, R. Govindan, Using the small-world model to improve freenet performance, in: Proc. IEEE Infocom, 2002.

[20] B.Y. Zhao, J. Kubiatowicz, A.D. Joseph, Tapestry: an infrastructure for fault-tolerant wide-area location and routing, pages Tech. Report, UCB/VSD–01–1141, UC Berkeley, 2001.

Ken Y. K. Hui received his B.Eng. (with first-class honors) in Computer Engineering in July 2002 and M.Phil. in Computer Science and Engineering in July 2004, both from the Chinese University of Hong Kong. He is now working as a researcher in the Chinese University of Hong Kong. His research interests are in the network protocol design, network security and network system architecture design. His personal interests include general reading and sports.

John C. S. Lui worked in the IBM T.J. Watson Research Laboratory, as well as in the IBM Almaden Research Laboratory before joining the Chinese University of Hong Kong (CUHK). He has been a visiting professor in Computer Science Departments at UCLA, Columbia University, University of Maryland at College Park, Purdue University, University of Massachusetts at Amherst and Universit degli Studi di Torino in Italy. Currently, he is the chair of the Computer Science & Engineering Department at CUHK and he leads the Advanced Networking & System Research Group. His research interests span both in systems as well as in theory/mathematics with the emphasis on the robustness, scalability, and security issues on the Internet. John received various departmental teaching awards and the CUHK Vice-Chancellor's Exemplary Teaching Award in 2001. He is on the editorial boards on various international journals and an elected member of the IFIP WG 7.3. He is currently on the directorial board of the ACM SIGMETRICS. His personal interests include films and general reading.

David K. Y. Yau received the B.Sc. (first class honors) degree from the Chinese University of Hong Kong, and the M.S. and Ph.D. degrees from the University of Texas at Austin, all in computer sciences. From 1989 to 1990, he was with the Systems and Technology group of Citibank, NA. He was the recipient of an IBM graduate fellowship, and is currently an Associate Professor of Computer Sciences at Purdue University, West Lafayette, IN. David received an NSF CAREER award in 1999, for research on network and operating system architectures and algorithms for quality of service provisioning. His other research interests are in network security, value-added services routers, and mobile wireless networking. David is a member of the ACM and IEEE. He serves on the editorial board of the *IEEE/ACM Transactions on Networking* and as technical program co-chair of the 2006 IEEE International Workshop on Quality of Service (IWQoS).