

# Small-World Overlay P2P Networks: Construction and Handling Dynamic Flash Crowd

Ken Y.K. Hui John C. S. Lui  
Dept. of Computer Science & Engineering  
The Chinese University of Hong Kong  
{ykhui, cslui}@cse.cuhk.edu.hk

David K.Y. Yau  
Computer Science Department  
Purdue University  
yau@cs.purdue.edu

## Abstract

In this paper, we consider how to “construct” and “maintain” an overlay structured P2P network based on the “small-world paradigm”. Two main attractive properties of a small-world network are (1) low average hop distance between any two randomly chosen nodes and, (2) high clustering coefficient. A network with a low average hop distance implies a small latency for object lookup. While a network with a high clustering coefficient implies the underlying P2P network has the “potential” to provide object lookup service even in the midst of heavy object traffic loading, for example, under a flash crowd scenario. In this paper, we present the SWOP protocol of constructing a small-world overlay P2P network. We compare our result with other structured P2P networks such as the Chord protocol. Although the Chord protocol can provide object lookup with latency of  $O(\log(N))$  complexity, 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 and propose an object replication algorithm to handle the heavy object traffic loading situation, e.g. under the dynamic flash crowd scenario. We show that the SWOP network can quickly and efficiently deliver the “popular” and “dynamic” object to all requested nodes. Based on our knowledge, this is the first work that addresses how to handle the “dynamic” flash crowd scenario on 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 Inter-

net. The first generation P2P systems such as the Napster system require a centralized directory service. The second generation P2P system is the unstructured P2P network (e.g., Gnutella, Freenet, . . . , etc) which uses a fully distributed approach for file searching. The major problem of the second generation P2P system is that in searching for an object, the amount of query traffic may be enormous and this leads to the network congestion problem.

For the past few years, researchers are working on distributed, *structured* P2P network. The noted work are Chord, Tapestry and Pastry protocols[13, 17, 20]. By applying the approaches of consistent distributed hashing function and structural routing, these structured P2P networks improve the performance of object lookup and at the same time, also reduce the amount of query traffic into the network. For example, under the Chord protocol, it was shown that the worst case complexity of  $O(\log N)$  link traversal can be achieved, where  $N$  is the number of nodes in a Chord P2P network. The implication is that in finding a particular data object, one does not need to generate enormous amount of query traffic which may overwhelm the network resource.

In this paper, we address two fundamental issues on designing a distributed, structured P2P network. They are:

- How can one further improve the performance of object lookup?
- How can a P2P network handle heavy demand to a *popular* and *dynamic* object such as the flash crowd scenario ? For example, during the September 11 incident, there were large number of requests to the CNN web server that tried to get the most up to date information.

To address the first technical question, we propose to construct a P2P network with the “small-world” paradigm[7, 18]: that the average shortest hop distance between two randomly chosen nodes is around six hops. The second technical question is of importance because some data objects may be very popular and requests to these popular objects may all arrive within a short period

of time. This type of traffic may overwhelm the source node which contains the popular object. Therefore, many users may not be able to access this popular object while only a small percentage of requests can be satisfied. To overcome this problem, we take advantage of the high clustering coefficient effect of a small-world network so that one can quickly *self-organize* and *replicate* the dynamic popular object.

In this paper, we propose a small-world overlay protocol (SWOP) to *construct* and to *manage* a P2P network so that it exhibits the small-world properties. We show that the constructed small-world P2P network has a better performance, as compared to other structured P2P network such as Chord, in performing data object. We also illustrate that the small-world P2P system is robust under heavy traffic loading and on average, it can quickly satisfy requests to the popular and dynamic object.

The balance of this paper is as follows. In Section 2, we present the protocol of constructing and maintaining a small-world P2P network. We also describe the object lookup protocol in which individual client node uses to locate any data object. We show that using the SWOP protocol, one can have a lower average object lookup latency, which is measured by the number of links traversal, as compared to other structured P2P network, i.e., the Chord network. In Section 3, we discuss how the small-world paradigm can be realized in an existing structured P2P network. In Section 4, we present the algorithm to handle heavy traffic loading such as the flash crowd scenario. We consider both the static and the dynamic flash crowd situations. In Section 5, we present the related work and 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 to construct and to maintain a small-world P2P network as well as the corresponding protocol for data lookup. We derive analytically the upper bound for the average object lookup latency. Last, we present the experimental results to illustrate the efficiency of data lookup when comparing with a structured P2P network (i.e., Chord).

The notion of the *small-world phenomenon* was originated from the social science research [10] and it is currently a very active research topic in Physics, Computer Science, as well as in Mathematics [11]. It was observed that the small-world phenomenon is pervasive in many settings such as social community, biological environment and data/communication networks. For example, recently studies show that peer-to-peer networks such as *Freenet* may exhibit the small-world phenomenon[19]. Informally, a small-world network can be viewed as a

connected graph wherein two randomly chosen nodes are connected by just the “*six degrees of separation*”. In other words, the *average shortest distance* between two randomly chosen nodes is approximately around six hops. The implication of this property is that one can locate information stored at any random node of a small-world network with only a small number of links traversal.

One way to construct a network that gives rise to the small-world phenomenon is that: (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*” while links to distant nodes are called “*long links*”. Figure 1 illustrates an example of a small-world network with 11 nodes and six clusters. For example, nodes 9, 10 & 11 form one cluster. They have neighboring links to each other and node 9 has long links to node 6, 14 and 22.

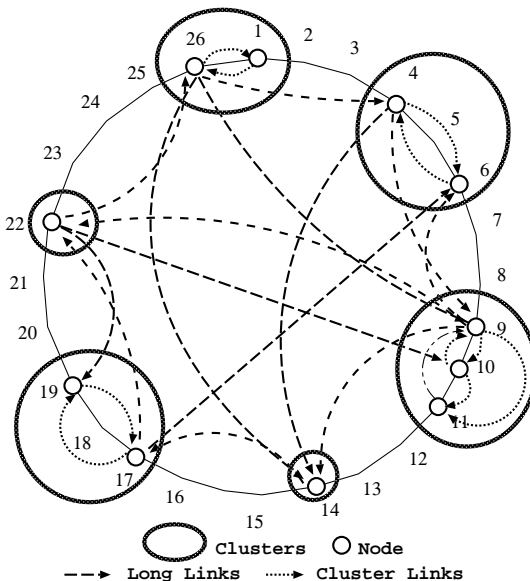


Figure 1: An example of a small-world network with 11 nodes and six clusters.

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  $\mathcal{G} = (V, E)$  denote a connected graph representing the small-world network. There are  $N$  vertices in  $\mathcal{G}$  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  $\mathcal{G}$ , denote as  $\mathcal{H}(\mathcal{G})$ , is equal to

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

In other words,  $\mathcal{H}(\mathcal{G})$  is the ratio of the sum of all shortest paths between any two nodes in  $\mathcal{G}$  and all possible pairwise connections of the connected graph. To define the clustering coefficient, let  $\kappa_v$  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  $C_v = |E(\Gamma_v)| / \binom{\kappa_v}{2}$  where  $|E(\Gamma_v)|$  is the operator of counting the total number of links for all vertices in the set  $\Gamma_v$ . The cluster coefficient of a graph  $\mathcal{G}$ , denote as  $\mathcal{C}(\mathcal{G})$ , is equal to

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

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

In the following, we first describe protocols of constructing and performing data lookup in a small-world P2P network. Then, we provide a mathematical analysis on the worst case average number of links traversal to locate an object in a small-world network. Lastly, we show that when comparing with other structured P2P networks, a small-world P2P network has a *lower* number of links traversal.

## 2.1 Overview

The aim of our small-world overlay protocol (SWOP) is to efficiently locate any object in the network and at the same time, achieve the capability to access *popular* and *dynamic* object under heavy traffic loading. Our SWOP is constructed as a layer on the top of a structured P2P network, where this layer does not affect the functionalities provided by structured P2P network. Moreover, it improves the performance on object lookup.

Let us provide a brief background on the structured P2P protocol. Generally, the structured P2P protocol consists of a consistent hashing function  $h$  (i.e., SHA-1 function) to provide a unique key assignment for each node or each object in the system. With the key’s value, each node can determine its logical position in the system. For example, for a Chord network, 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 routing table (i.e., the finger table in the Chord network), which speeds up the object lookup process. It was shown that the worst case number of links traversal to locate an object is  $O(\log(N))$  number of links traversal[17].

Each node can insert objects into the structured P2P system. Using the same consistent hashing function  $h$ . Each object has a unique key value, for example,  $h(o)$  is

the key value for object  $o$ . Let  $\mathcal{N}(o)$  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 to cache and to maintain the object  $o$ .

For a small-world P2P network, we use a circular ring as our logical representation since it is a representative model in structured P2P networks and it helps to reveal the small-world effect introduced by the SWOP protocol. Let us first define the following parameters for the SWOP:

- Cluster size  $G$  – maximum number of nodes within a cluster.
- Cluster distance  $D$  – maximum hash space distance between two adjacent nodes within a cluster.
- Long Links  $k$  – number of long links of a cluster.

For our small-world P2P network, there are two types of nodes, namely, *head nodes* and *inner nodes*, and two types of links, which are *long links* and *cluster links*. Long links connect two different nodes from 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 and it has cluster links to all nodes within its cluster. On the other hand, 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 (links) for each node is:  $(m * k + G * n) / n$ , where  $n$  is the number of nodes in the SWOP system. (Note that the average number of clusters can be estimated by one of our protocols, CNEP, which will be explained in the next section.)

With the above configuration, an inner node  $i$  can communicate with a target node  $j$  within its cluster either by a cluster link (provided node  $i$  and node  $j$  are connected), or node  $i$  can send a message to its head node, and then the head node will 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, then the head node sends the message using the long link which is the closest to the target node  $j$ . The message may arrive at some nodes which is not within the same cluster of node  $j$ . The procedure repeats until the message is transferred to some node within the same cluster of node  $j$ . In figure 2, it shows an example of small-world overlay P2P network with 11 nodes, six clusters and with SWOP parameters  $G = 3$ ,  $D = 2$  and  $k = 3$ . In the figure, node 1 sends a message to node 17 and we illustrate the object lookup flow.

In the following, we describe several protocols in forming and in maintaining a small-world P2P network. In particular, protocol for events such as node joining, node leaving as well as node failure.

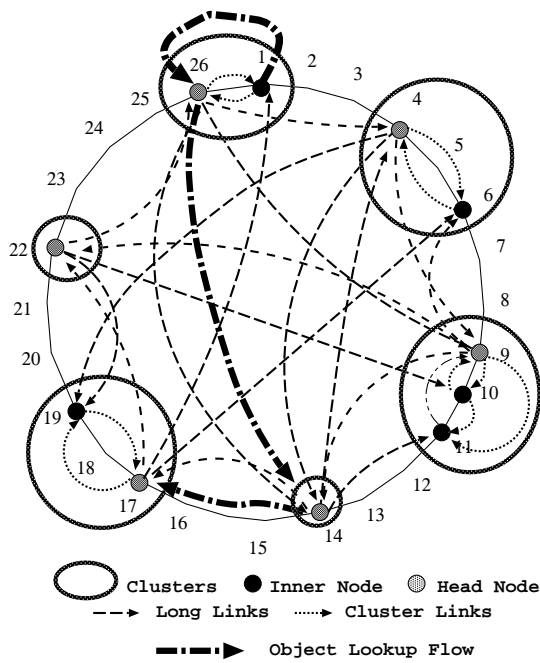


Figure 2: A SWOP network with six clusters,  $G = 3$ ,  $D = 2$ ,  $k = 3$  and the object lookup flow.

**Join Cluster Protocol (JCP):** if a node  $i$  wants to join a small-world P2P network, it uses the consistent hashing function  $h$  to obtain its key  $h(i)$ . At the same time, node  $i$  creates a link to its predecessor node  $a$  and 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) from the predecessor and successor nodes. Let  $d_1$  and  $d_2$  be the distance between the joining node  $i$  and its predecessor and successor node respectively, that is,  $d_1 = h(i) - h(a)$  and  $d_2 = h(b) - h(i)$ . The joining node  $i$  inquires from its predecessor and successor nodes for their respective cluster size. If both nodes  $a$  and  $b$  have 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 this new cluster. Else, it 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  $D$ , then the joining node  $i$  will form a new cluster with itself being the only node in that new cluster. Else, the joining node  $i$  joins the predecessor's cluster (or the successor's cluster) if  $d_1$  (or  $d_2$ ) is less than  $D$  and is smaller than  $d_2$  ( $d_1$ ).

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  $i$  is 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 with its head node. On the other hand, if the joining node  $i$  is the head node, it needs to create cluster links to all inner nodes within the cluster and at the same time, generates  $k$  long links to other nodes in different clusters.

In order to achieve a low average shortest hop count  $\mathcal{H}(\mathcal{G})$  and high cluster coefficient  $\mathcal{C}(\mathcal{G})$ , 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  $\tilde{X}$ , which has the following probability mass function:

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

The implication of the above probability mass function is that it is bias toward 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 links traversal between these two clusters.

---

#### Join Cluster Protocol

```

n.join_cluster() { /* For each node  $n \in \mathcal{V}$  */
  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;
  Computer 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) {
   $n$  contact the old head of retrieved Id;
  Retrieve the whole membership list;
  Regenerate long links if it is necessary;
} else if ( $n$  to be a member) {
   $n$  sends a message to the target cluster head;
   $n$  retrieve part of the membership list;
}
}

```

---

**Leave Cluster Protocol (LCP):** When the node  $i$  leaves the P2P system, it informs its neighboring nodes, which are nodes connected by cluster links, as well as some long links if node  $i$  is a head node. Node  $i$  will inform its neighboring nodes about 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 re-generate 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 links, and the long links routing tables from the new head node in order to maintain messages routing capabilities for all clusters neighbors.

The pseudo code 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 reduces by 1;
      Update information for all member;
    } 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 node failure, the small world P2P network uses the SCP to recover and to maintain the proper link connectivity. Periodically<sup>1</sup>, 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 that inner node and will 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, then the node will perform a search to find a new head node.

In order to facilitate the recovery of the head node failure, we replicate the cluster links from the head node to its successor nodes. The number of clusters links,  $G$ , is a system parameter which can be adjusted (due to the space limit, we leave this as the future research work). Suppose that there is a head node failure in the SWOP system, the SCP will first ensure that one of the nodes in the

---

<sup>1</sup>The time scale is in the order of minutes.

SWOP system discovers this failure and be able to find a new node (normally, it is the failed head node’s successor). The new head node will replace the failed head node by executing the OLP. For the above action, in terms of the number of messages transmitted in the SWOP system, it is bounded by the *worst case* average number of links traversal of  $(1 + \log_2(m/2))8\ln(3m)/k$  (the proof of this claim will be derived in Section 2.2).

Secondly, the new head node sends a probe message to verify the old head node’s failure. If the confirmation is successful, that is, without receiving any reply, the new head node starts the head node’s job. The first duty is to send out a head node declaration message to each cluster neighbor, which announces itself as the new head of their cluster. In this step, the total messages involved is:  $1 + G$ , which includes the confirmation messages and announcement messages.

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

---

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

```

---

**Object Lookup Protocol (OLP):** The object lookup protocol is responsible to locate a data object within a small-world P2P network. The object lookup process proceeds in two phases. In phase one, a node asks its cluster neighboring nodes if they contain the target object. If any of these cluster neighboring nodes replies positively, then the lookup process is terminated. Otherwise, the object lookup process continues and the phase two begins. 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, then the head node will continue the object lookup process recursively as described above. For example, when the long-link neighbor receives that object lookup request, it checks if it is the owner of the object. If not, it will act as if it is the object lookup initiating node and the data lookup process is repeated. An object lookup process continues in these two phases alternatively until the data object is found.

To illustrate, consider an example shown in Figure 2. Suppose that node 1 wants to look up an object whose key value is 16 and node 17 is responsible to manage and maintain this object. To begin the object lookup, node 1 starts the phase one lookup process in which it asks its neighbor, node 26, if it contains item 16. Since node 26 is not the owner of that object, node 1 will receive a negative reply. Then node 1 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 contains the object 16. Since it does not contain that object, node 14 acts as an object 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 is terminated.

---

### 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 this purpose.

The CNEP periodically estimates two values,  $N$  and  $G$ , which are the average number of nodes and the average cluster size in the system. 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, to estimate the average number of nodes  $N$  in the system, and the average cluster size of the network. To estimate  $N$ , we can utilize the property of distributed hash function of evenly assigning IDs. Since nodes are

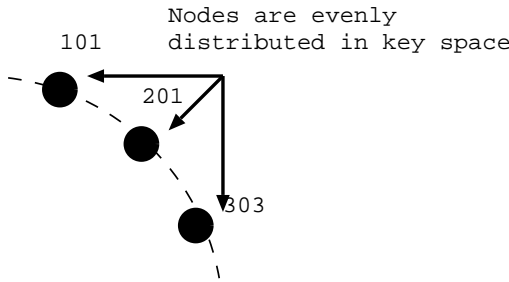


Figure 3: A snapshot of a SWOP network

evenly distributed in the hashed key space, the average distance of separation between two nodes represents the *compactness* of the underlying network, which reflects the number of nodes in the system. By comparing the average distance of separation with the total ID space separation, one can use the proportional calculation to estimate the average number of nodes,  $N$ . For instance, a snapshot of the SWOP network is depicted in Figure 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 sufficient number of cluster heads' cluster size records. The main challenge of

this component is how cluster heads 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 its cluster average distance of separation between two nodes in its own 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 collected sufficient amount of data samples for these two values, the head node can take an average for these two values separately. Fourth, each head node calculates the average number of nodes by the average distance of separation obtained in the previous step. And each head node can now estimate the average number of clusters by dividing the average number of nodes from the average cluster.

### Cluster Number Estimation Protocol

```

/*
    Given Hashed-Space Size HS
*/
/* For Cluster Head ∈ 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 packets 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 ∈ 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 new mathematical theorem based on randomized analysis to quantify the *worst case* average number of links traversal to locate an object under a small-world overlay P2P network.

**Theorem 1** *Let  $Y$  be the non-negative random variable that represents the number of links traversal for object lookup in the SWOP P2P network. We have:*

$$E[Y] \leq (1 + \log_2(m/2))8 \ln(3m)/k, \quad (4)$$

where  $m$  and  $k$  are the number of clusters and the number of long links of the corresponding SWOP P2P network respectively.

**Proof:** Let  $\mathcal{G} = (V, E)$  be a connected graph representing a small-world overlay P2P network with  $|V| = N$ . Let  $m$  be 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  $\bar{C}$  be average number of nodes within a cluster, therefore, for sufficient large value of  $N$ , we have

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

We model a connected graph  $\mathcal{G}$  with  $m$  clusters. We define  $d(i, j)$  as the lattice distance (in the hash key space) between two clusters  $C_i$  and  $C_j$  and it 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 link from  $C_u$  to  $C_v$  is  $\frac{d(u,v)^{-1}}{\sum_{v \neq u} d(u,v)^{-1}}$ . We can express

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

For the object lookup protocol (OLP) described above, a node transfers 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 this message, we call this cluster as the current lookup message's holder. We define cluster movement as the traversal along the long link from one cluster to another cluster.

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. Describing our object lookup process in terms of step, 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 away.

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  $j$ , one can view that the object lookup needs 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 this event. To derive this probability, we first define  $B_j$  as the set of nodes with the cluster movement of being  $2^{j+1} + 2^j$ , which is less than  $2^{j+2}$  from the target node. Each node in  $B_j$  has a probability of at least  $(2 \ln(3n)2^{j+2})^{-1}$  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 of 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 total number of cluster movement spent in step  $j$ . We have

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



Let  $Y$  denotes the total number of links traversal spent by the object lookup algorithm, we have

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

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

**Remark:** The importance of this theorem is that we can use it to estimate the proper value of  $k$  so that the SWOP P2P network has a better object lookup performance than other structured P2P network. Let us consider this in the following subsection.

### 2.3 Experimental results of comparing with other structured P2P networks

In the following, let us compare the object lookup performance between the small-world P2P network and other structured P2P network such as the Chord system[17]. We built topology generators for SWOP and Chord and conducted the simulations on measuring average lookup hop distance between two randomly chosen nodes and collecting the topological data, including average shortest paths and average clustering coefficients. We illustrate that, indeed, the small-world P2P network will have a better performance as compare to the underlying Chord system.

**Experiment A.1 (Performance of object lookup):** We consider a connected graph with  $N = 1000, 2000, 3000, 4000$  and  $5000$  nodes. We insert one object for each node, totally  $N$  distinct objects. Each node will perform object lookup 50 times and the target object is randomly chosen from all inserted objects. We obtain the performance of object lookup, which is measured by the number of links traversal for the Chord and the SWOP network. For the SWOP network, it is configured with parameters  $G = 100, D = 120,000$  and  $k = 24$ . In order to perform a fair comparison, we keep the size of the finger table in the Chord network as 24 also. Figure 4, 5, 6, 7 and 8 illustrate the probability density functions of number of links traversal for object lookup under the Chord and SWOP systems for  $N = 1000, 2000, 3000, 4000,$  and  $5000$  respectively. Table 1 summarizes the effect of object lookup under the Chord and SWOP systems in which the number of nodes in the system varies from 1000 to 10000. It is important to observe that the SWOP network has a *lower* average number of links traversal in object lookup than the Chord network.

**Experiment A.2 (Effect of object lookup performance under different network sizes and number of long links):** We explore the object lookup performance further by varying on the network sizes (e.g., different values

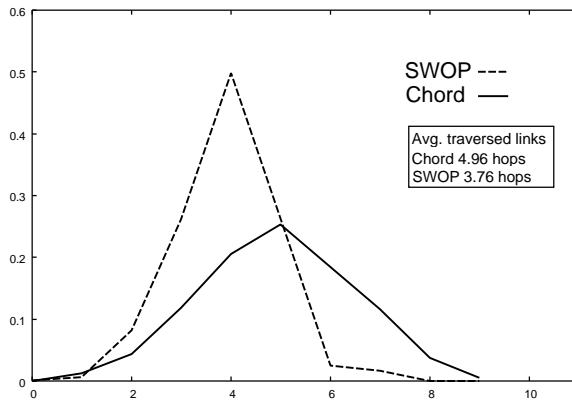


Figure 4: Probability density function of link traversal for Chord and SWOP with  $N = 1000$  nodes.

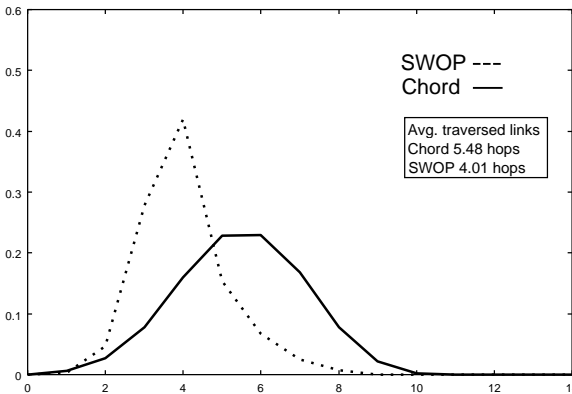


Figure 5: Probability density function of link traversal for Chord and SWOP with  $N = 2000$  nodes.

of  $N$ ) and the number of long links  $k$  for the SWOP network. In order to make a fair comparison, we keep the size of the finger table of the Chord network equal to  $\log(N)$ , where  $N$  is the number of nodes in the P2P network. For the SWOP network, 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 result is illustrated in Table 2 and it shows that by increasing the value of  $k$ , not only can one further improve the object lookup performance, but one also enhances the asymptotic performance that it can be easily reached by fixing a small number of long links, say  $k$  around 6 to 8. Since this value is less than  $\log(N)$  for large  $N$ , it implies that for the SWOP network, one has a *lower* connection management complexity and at the same time, is able to achieve a better object lookup performance.

**Experiment A.3 (Comparison of Clustering Coefficient):** The above results show that the average object lookup performance of SWOP is better than the Chord protocol. The next issue we want to investigate is their corresponding average cluster coefficients. Again, a high

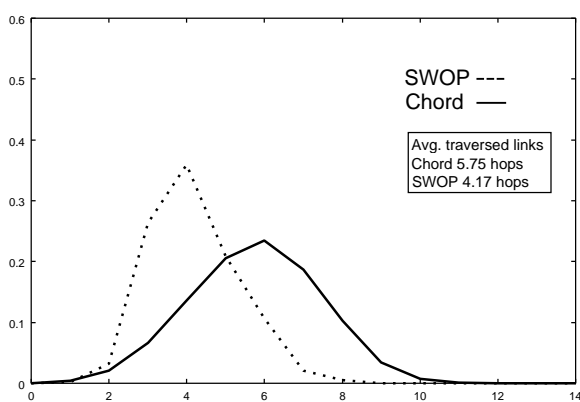


Figure 6: Probability density function of link traversal for Chord and SWOP with  $N = 3000$  nodes.

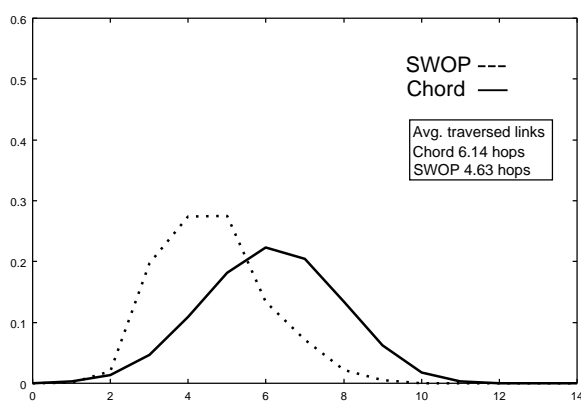


Figure 8: Probability density function of link traversal for Chord and SWOP with  $N = 5000$  nodes.

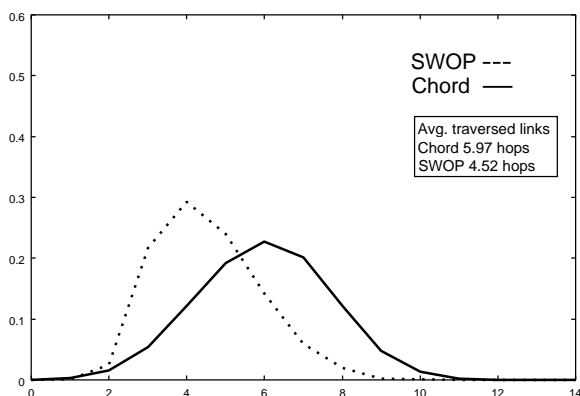


Figure 7: Probability density function of link traversal for Chord and SWOP with  $N = 4000$  nodes.

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

Table 1: Object Lookup performance (e.g. number of link traversal) with standard deviation of SWOP and Chord networks under different values of  $N$ .

### 3 Realizing the Small World Paradigm in an Existing Structured P2P Networks

In general, SWOP can be applied to different structured P2P networks, such as CAN, Pastry and Tapestry. A small world network layer can be constructed on top of different structured P2P networks based on their similarities.

A structured P2P network can be summarized into three major components: a unique key assignment scheme, a characteristic routing table and an effective routing scheme. For our SWOP protocol, it can be implemented as another layer on top of 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 mean to resolve the high traffic problem. This procedure can be carried out in CAN, Pastry and Tapestry as they contain similar key assignment scheme. Based on the grouping criteria, group size and node separation distance, each node can decide a suitable cluster to join and select a appropriate cluster head node. In our

clustering coefficient implies that a higher capability to handle heavy traffic workload. We vary the number of nodes in the overlay network in this experiment. For the Chord network, each node has a finger table of size equal to  $\log(N)$ . For a fair comparison, we also keep the number of long links for the SWOP network with  $k = \log(N)$ . The clustering coefficient is computed based on Equation (2). The result is illustrated in the Table 3. We observe that the SWOP network has a *higher clustering coefficient*. Again, the importance of having a higher clustering coefficient is that the P2P network is more efficient in handling the heavy object lookup traffic, for example, under the flash crowd scenario. We will explore this feature in detail in the following section.

$N$ :	Chord	SWOP $k=4$	SWOP $k=6$	SWOP $k=8$	SWOP $k=10$	SWOP $k=12$
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 2: Object Lookup performance (e.g. number of link traversal) of SWOP and Chord networks under different values of  $N$  and  $k$ .

$N$ : # 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

Table 3: Average Clustering Coefficient

SWOP, the cluster head is selected using the smallest ID, but in general, it can be the smallest ID or vector. Then, different structured P2P networks can apply our selection criteria to choose a suitable cluster head to represent a cluster in a P2P network.

Secondly, one needs to construct the small world routing table. To realize this, we fix each inner node, or cluster member, to connect to nodes within a cluster only and we allow each head node to connect to nodes in other clusters according to harmonic distribution in Equation (3), which is aiming at providing the high degree hubs presenting in a small world network. For different structured P2P networks, provided that each node has the cluster head information, it can maintain a list of short links. Moreover, P2P networks can execute the CNEP protocol as described in the previous section, then nodes can estimate the average number of clusters within a network and they can connect to other clusters based on Equation (3) of using long links.

Third, after the formation of small world routing tables, we can use the OLP to perform object lookup directly because the above discussed algorithms are based on short links within a cluster and long links connecting different cluster, which are maintained in small-world maintenance procedure.

Currently, our small world layer groups nodes by comparing similar IDs to form clusters. This scheme can be modified such that other application specific purpose can be achieved. For instance, one can group nodes based on transmission delay or semantic interest etc. For transmission delay, one can measure the network delay between two nodes by sending probe packets and estimate the delay. This kind of group can help reduce the transmission

delay of file transfer. For semantic interest, one can assign a semantic vector, based on file lookup interest, to each node in the network. This type of network may reduce the object lookup latency because users have a tendency to search similar interest within a cluster first.

## 4 Protocols for Handling Flash Crowd

In this section, we address how to handle object access under a heavy traffic loading. An example of a heavy traffic loading is the flash crowd scenario [14, 16] wherein a massive number of users try to access a popular object within a short period of time. The incident of 911 is the prime example of a flash crowd scenario where enormous requests overwhelmed some popular news sites such that only a small number of users could retrieve the object while majority of users could not.

One way to deal with the flash crowd situation is to replicate the popular object to other nodes. This way, access to the popular object can be spreaded out so as to avoid overwhelming the source node. However, one has to address the following technical issues:

- The replication process cannot be self-initiated but rather, driven by a high traffic demand. Or else someone may maliciously replicate many objects in a P2P system.
- Under a structured P2P network, object lookup is carried out by using the object’s key value and only *one* node manages that object. How can one enhance the protocol so that more than one node can store the popular object?

We first divide the study of the flash crowd scenario into two cases, namely, (1) the *static* and (2) the *dynamic* cases. A static flash crowd is a flash crowd phenomenon in which the popular object involved will remain unchanged after its first appearance, e.g., a newly published book or released video. On the other hand, a dynamic flash crowd is a flash crowd phenomenon in which the content of the popular object will change over time after its first appearance, e.g., a frequently updated news article. We first describe algorithms to handle the static flash crowd problem, then we extend the proposed protocols to handle the dynamic flash crowd by adding mechanisms to notify the change/update of different popular objects in order to replicate these popular objects efficiently.

### 4.1 Static Flash Crowd

Under a flash crowd situation, the traffic can overwhelm the source node that contains the popular object. To avoid

this problem, the source node needs to replicate this object to other nodes and in return, these nodes can serve some of the object lookup requests and reduce the traffic to the source node. Note that the object replication has to be "demand driven" or else a node may be able to maliciously replicate objects to all other nodes in a P2P network. In the following, we first describe an access-rate checking function, which is used to estimate the access rate of an object in the system. Note that this function can be applied to the SWOP P2P network or other structured P2P network such as the Chord.

To estimate the access rate of an object, we use the `access_checking` function specified below. This function determines the access rate of object and decides whether the object should be replicated or not. Each node in a P2P system calls the `access_checking()` function every `PERIOD` units of time to estimate the access rates of its managed objects. At all times, each node also keeps track of the number of access to its management objects in the array `access_count[]`. The pseudo code of the `access_checking()` function is shown as follows:

---

### Access checking

```
n.access_checking() /* For each node  $n \in \mathcal{V}$  */
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 the Chord P2P network can process up to  $\lambda_t$  requests per second with a reasonable performance. Whenever a source node discovers that the request rate for an object is  $\lambda_1$  where  $\lambda_1 > \lambda_t$ , the source node starts the replication process by pushing this popular object to *all* its neighboring nodes, e.g., all nodes listed in its finger table. These receiving nodes will cache this popular object for  $T_1$  time units. Using the Chord object lookup protocol, any node that wants to access this popular object may pass through these receiving nodes and can then access the popular object. For a node that caches the popular object, if it receives a lookup rate  $\lambda_2$  for that object, where  $\lambda_2 > \lambda_t$  then in turn it will push this cached popular

object to *all* its neighboring nodes. The motivations of the above algorithm are: (1) the replication process of a popular object is purely demand driven, (2) for a popular object, it will be replicated to other nodes in the Chord system so other nodes can access that object.

We applied the above approach for the Chord system to handle the flash crowd scenario because the nodes in the Chord system are evenly distributed and the fingers of the nodes are evenly spreaded among the system. By pushing the "popular" items to those evenly spreaded finger nodes, one can then distribute the "cached" item evenly in the system, which can increase the hitting probability by a randomly generated query. Thus, the above scheme is used for the Chord system so as to provide a fair comparison between the SWOP system and the Chord system.

**Implementation of replication process in Chord:** The pseudo code of the replication algorithm is illustrated as follows. If the access rate is larger than  $\lambda_t$ , the function `push(int objectId)` will be called. The popular objects will be pushed and replicated to all nodes indicated by the finger table under the Chord system.

---

### Push – Chord version

```
n.push(int objectId)
1. for ( $\forall$  neighboring nodes  $\in$  finger_list) {
2.   transfer 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 take advantage of this property so as to achieve a lower replication time and lower links traversal to obtain the popular object. We also assume that a node in the SWOP network can process up to  $\lambda_t$  requests per second. Whenever a source node receives a request rate for an object being  $\lambda_1$  with  $\lambda_1 > \lambda_t$ , the source node will start the replication process by pushing the popular object its neighbors, e.g., all nodes connected by *long links* of its cluster. All these receiving nodes will cache this popular object for  $T_1$  time units. Any node that wants to access this popular object uses the *Object Lookup Protocol* (OLP) described in Section 2. If the popular object is cached by any node in its cluster, the requesting node can access this popular object quickly. For a node that caches the popular object, if the lookup rate for that cached object is higher than  $\lambda_2$ , in turn, it will push this cached popular object to all nodes connected by long links of its cluster. Note that the main motivation of replicating the popular object via the long links is to propagate information to distant clusters so that

nodes within those clusters can easily access the popular object.

**Implementation of replication process in a SWOP network:** The pseudo code 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 the long links. If the access rate is larger than  $\lambda_t$ , the function `push(int objectId)` will be called. The main idea of this *push* function is to replicate popular object from one cluster  $c_i$  to another cluster  $c_j$  via  $c_i$ 's long links. The popular object will be cached in among the long link neighbors. Since each long link neighbor belongs to another cluster, pushing the popular object to each long link neighbor means spreading or replicating that object to each cluster. Under our SWOP, our OLP can achieve an efficient lookup within a cluster. The detailed pseudo code 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 ( $\forall$  neighbors  $\in$  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 crowd scenario, an additional communication message, *update message*, and an extra data structure for each object's version number, or update counter, are added to static SWOP PUSH algorithm in order to handle the dynamic popular object.

For the static algorithm applying on the SWOP network, an efficient replication has been operated such that each cluster in the system has exactly one node caching the popular object. Consider at this moment, each node with the popular object marks this copy as original, say **version 0**. If an updated version, say **version 1**, is inserted to the system by the source node, we only need a

light weight notification to inform all cached nodes about the newly updated popular object. Again, this notification can be carried out by exploiting the small-world property and the static replication scheme.

To implement the above dynamic algorithm, we enhance the static algorithm as described previously, which is the replication scheme for the original version of the popular object. Whenever the source node has an updated version, additional tasks have to be carried out. First, an update message is sent out by (1) the source node and, (2) the cached node of the updated object. In general, there are two types of update messages, which are (i) sending to all cluster neighbors and, (ii) sending to all long link neighbors of each cluster head. The first type of message involves the cluster neighbors only and it reminds the cluster neighbors to lookup the latest updated version of the popular object. On the other hand, the second type of message involves the long link neighbors. It reminds these nodes about the new update and transfers the updated popular object to these nodes. This implies that an updated version will be replicated from one cluster to another cluster. This type of message requires the co-operation between head node and the sending node if the sending node is not a head node. As a receiver of the update message, if a node does not contain a cached copy of the popular object, it will use the OLP protocol as described in Section 2 to retrieve the updated object.

## 4.3 Mathematical analysis of object replication time

In this section, we present the mathematical derivation of the average time, the memory requirement of each node and the bandwidth requirements of replicating a popular object to all clusters in a SWOP overlay P2P network. We model the spreading process by a continuous time Markov chain (CTMC).

**The Average Spreading Time:** Let  $\mathcal{M}$  be the CTMC representing the replication dynamics of a SWOP overlay network. The SWOP overlay network has  $N$  nodes. The CTMC  $\mathcal{M}$  has a state space of  $\mathcal{S}$  such that  $\mathcal{S} = \{1, 2, \dots, m\}$ , where  $m$  is the number of clusters in a SWOP P2P network. State  $i$  in  $\mathcal{M}$  represents that the popular object has already been replicated to  $i$  nodes in the SWOP network. Since the source node contains and manages the popular object initially, the initial state of the CTMC  $\mathcal{M}$  is in state 1. Define  $x_1$  as the number of requests needed within a time period  $\tau$  so that the source node of the popular object will start replicating the popular object to all its long links neighbors (e.g.,  $\lambda_1 = x_1/\tau$ ). After receiving the popular object, these neighboring nodes become replicated nodes. Similarly, define  $x_2$  as the number of object requests needed for a replicated node to start replicating the popular object to

all its long links neighbors Let  $\lambda$  be the request rate of the popular object from each node in a SWOP P2P network. The rate of replicating a popular object from the source node is  $\lambda_s$ , 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  $\bar{C}$  represents the average number of nodes within a cluster. The rate of replicating a popular object from a replicated node is  $\lambda_r$ , which is equal to:

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

Let  $Q$  be the infinitesimal rate matrix of  $\mathcal{M}$  and we denote the element in  $Q$  as  $q_{i,j}$ , which is the transition rate from state  $i$  to state  $j$ , for  $i, j \in \{1, 2, \dots, m\}$ . Assuming that a replicated node will cache the object for an average time of  $1/\mu$ . Let  $\nu_1 = \lambda_s$  and  $\nu_i = \lambda_r$  for  $i \in \{2, \dots, m\}$ . The transition rate matrix of  $Q$  can be specified by the following transition events:

**Object deletion event:**

$$q_{i,i-1} = i\mu \quad \text{for } 1 < i \leq m \quad (5)$$

**Object replication event:** we have two cases to consider:

*Case 1:* for state  $i \in \mathcal{S}$ , if  $(i + ik) \leq m$ :

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

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

$$q_{i,j} = \begin{cases} \sum_{x=m}^{(i+ik)} \binom{m-i}{m} m^{-i} \left(\frac{i}{m}\right)^{x-(m-i)} \nu_i & \text{if } j = m \\ \binom{m-i}{m} j^{-i} \left(\frac{i}{m}\right)^{ik-(j-i)} \nu_i & \text{if } k \leq j \leq (i + ik) \\ 0 & \text{otherwise.} \end{cases} (7)$$

Once the rate matrix  $Q$  is specified, one can derive the average time to replicate a popular object to all clusters in a SWOP P2P network using the theory of fundamental matrix in Markov Chain [2, 12].

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

$$M = (I - P_c)^{-1} = \sum_{i=0}^{\infty} (P_c)^i. \quad (8)$$

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

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

where  $e_1$  is a row vector of zero except the first entry being 1 and  $\mathbf{1}$  is a row vector of all 1's.

To quantify the memory requirement, one can measure the expected number of *keys* stored in each node in the SWOP system. The keys can be divided into two categories: topological keys and item keys. For topological keys, the upper bound of these keys can be calculated by considering the head nodes in the SWOP system since these nodes have both long links and short links for topology construction. The value of this upper bound is  $(\log N + 2) + (G + k)$ , which includes the topological keys for the underlying DHT and the SWOP overlay. For item keys, the upper bound depends on the number of popular objects in the system during flash crowd period. One can expect that the final result after executing flash crowd handling protocol is that each node contains one copy of each popular object. So, the bound of memory requirement for item keys for each node is the number of popular object in the system. In summary, if there is  $i$  popular objects in the SWOP system, the memory bound for each node in the SWOP system 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 SWOP system. 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, the ‘‘cached’’ 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 (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 to  $c$  clusters, where  $c < m$  ( $m$  is the average number of clusters in the SWOP system), and each server needs  $r$  requests to trigger the replication. 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 member as defined in previous sections.

#### 4.4 Experimental results of handling flash crowd

In this section, we present the experimental results of comparing the performance of replicating a popular object

for a Chord and SWOP P2P networks. We use the topology generator developed in Section 2 to form and maintain a small-world P2P network. And we use a discrete event driven simulator to generate workload that simulates a flash crowd scenario.

In our experimental study, we use a 24-bits hash space.

There are  $N = 2000$  nodes in a P2P system. In order to provide a fair comparison, each node in the P2P network (both for Chord and the small-world P2P network) has no more than 11 long link neighbors. This implies that, under the Chord network, the size of the finger table is 11 and, for the small-world SWOP network, the size of the the long link neighbor list is  $k = 11$ . We generate a popular object whose key value is randomly chosen from the 24-bits hash space and we insert this popular object in the P2P network. Each node in the P2P network generates request to this popular object with a Poisson arrival rate of  $\lambda$ . If a node receives a request lookup, it will process the request with a Poisson service rate of  $\mu$ , 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 up. Each node has a finite queue size to store incoming requests. If the request arrives at a full queue, the request will be dropped.

We carried out experiments and measure the "effective replication time" of SWOP and Chord. The effective replication time is reflected by 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 at each node. In our study, a request is successful only if it can access the popular object by time  $t$ . We use this performance metrics for investigating the effective replication time, that is how fast that a popular object can be replicated under the SWOP or the Chord network.

**Experiment B.1: Comparison between Chord and SWOP:** This is a basic comparison of the static and the dynamic flash crowd scenarios between the Chord and the SWOP network. We fix the per node average request arrival rate of the Chord and the SWOP as  $\lambda = 0.003$  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 Result:** Figure 9 shows the number of successful requests as a function of time. The result shows that the SWOP network has a much better performance on object replication than the Chord. For example, at time  $\geq 20$ , most of the requests to the popular object are successful under the SWOP network. However, only around 6% of the requests are successful under the Chord network.

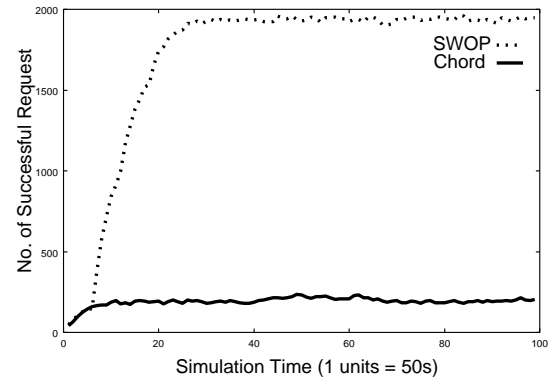


Figure 9: Comparison of the number of successful requests to the popular object under the static flash crowd with  $N = 2000$  nodes.

**Dynamic Result:** Figure 10 illustrates the performance of both P2P networks under the dynamic flash crowd situation. It uses the same plot setting as the static result. Since the content of the popular object is updated at time  $t = 25, 50$  and  $75$ , we consider a request is successful and only if it can access the most up to date version of the popular object. Figure 10 shows that the number of successful request for the SWOP network is much better than that of the Chord. Moreover, when the object changes its content at time  $t = 25, 50$  &  $75$ , the SWOP network can quickly notify other nodes so that other nodes can eventually access the most recent version of the popular object. On the other hand, the Chord network is not as effective as the SWOP in replicating the object. Thus, the SWOP network has a much better performance during the dynamic flash crowd situation.

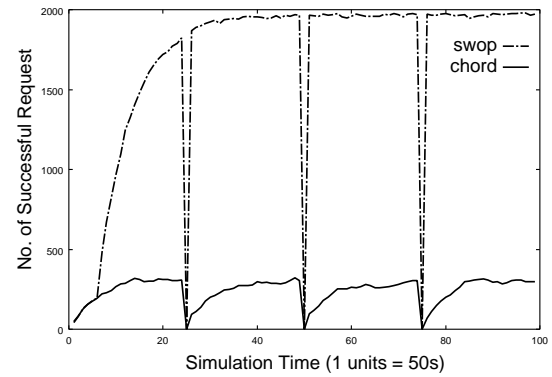


Figure 10: Comparison of number of successful requests under the dynamic flash crowd with  $N = 2000$  nodes. (object changes its version at  $t = 25, 50, 75$ .)

**Experiment B.2: Comparison on Queue Size:** In this experiment, we investigate the effect of different queue size on the number of successful request. We keep the same experiment configuration as in Experiment B.1 ex-

cept that we vary the queue size of each node. from 20 to 50 and the result is shown in Figure 11. From the figure, we observe that the SWOP network is not very 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 high successful request rate. Thus, the system resources for each node in the SWOP network can be reduced without affecting the performance during the flash crowd situation. Lastly, the figure also illustrates that the SWOP network performs much better than the Chord network under the flash crowd scenario.

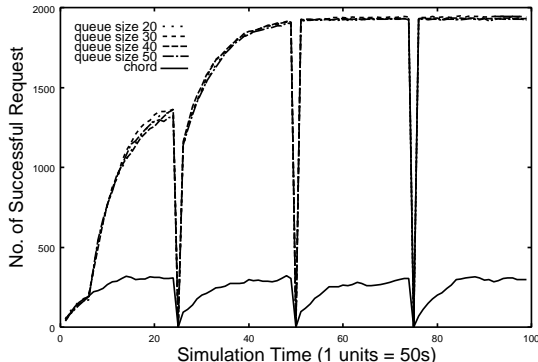


Figure 11: Effect of queue size on the number of successful request for the SWOP and the Chord network (object changes its version at  $t = 25, 50, 75$ .)

**Experiment B.3: Variation on object request rate:** In this experiment, we further examine the performance on handling dynamic flash crowd when object request rate is being varied. We keep the same configuration as in Experiment B.1. We vary the per node object request rate  $\lambda$  from 0.001 to 0.005. The result is illustrated in Figure 12. Result shows that the SWOP performs the best under the arrival rate of 0.003. The reason is that when the request rate is greater than 0.003, the aggregated request rate is higher than the service rate for the individual node which caches the popular object within a cluster. On the other hand, when the request rate is smaller than 0.003, the aggregated request rate is small enough so as to achieve a very high number of successful request. To handle higher request rate, we may need to dynamically control the cluster size (e.g., to make a smaller cluster) in order to achieve a high successful object access rate. Due to the lack of space, we leave this as one of our future work.

**Experiment B.4: Variation on number of long link neighbors ( $k$ ):** This experiment investigates the performance when we vary the number of long link neighbors  $k$  in a small-world P2P network. We adjust the topology of the SWOP network such that each node in SWOP has  $k = 7, 9$  or  $11$  long links. In this experiment, we keep the finger table size of the Chord network to be 11. The result is illustrated in Figure 13, which shows that even when the

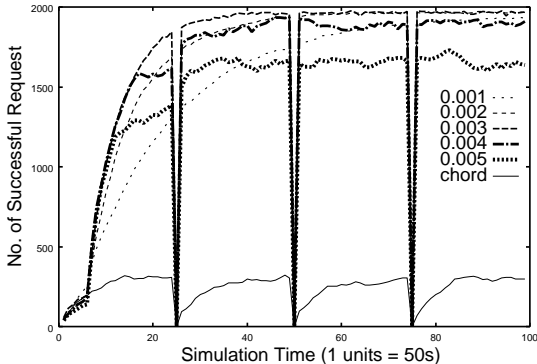


Figure 12: Effect on the variation of per node request rate  $\lambda$  on the number of successful request for the SWOP & the Chord network (object changes its version at  $t = 25, 50, 75$ .)

number of long links neighbors in SWOP is reduced, the performance on the number of successful requests under the SWOP network is *significantly better* than that of the Chord network. For example, one can observe when the SWOP has  $k = 7$  long links, the number of successful requests at the beginning is less than  $k = 9$  or  $k = 11$  long links of SWOP, but the number of successful requests can quickly catch up. Also, the replication rate is much better than the Chord network. The implication of this experiment is that the SWOP network only needs to manage a small number of long links than the Chord network but it is still very robust under the dynamic flash crowd situation.

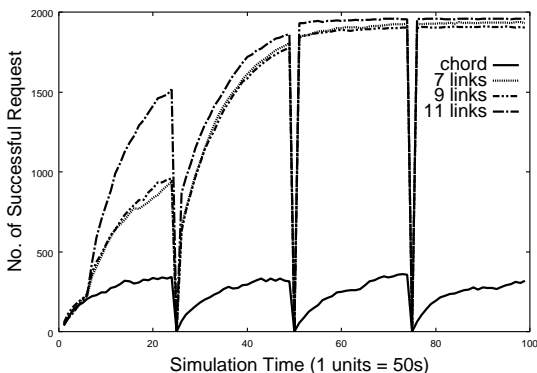


Figure 13: Effect of the variation of number of long links  $k$  for the SWOP & the Chord network. (object changes its version at  $t = 25, 50, 75$ .)

**Experiment B.4: Amount on traffic generation:** In this experiment, we examine the effects on traffic loadings, in terms of the number of messages generated in the network to handle flash crowd. For both the SWOP and the Chord networks, we compare the differences on their traffic loadings when they are facing the same flash crowd scenario.



We employ the simulation as described in previous experiments with the modification that we record the number of message generated. We set parameters  $N=2000$  nodes for both the SWOP and the Chord networks,  $\lambda = 0.003$  requests/second for each node and  $\mu = 1$  for the each source and object cached node. Before the simulation starts, a popular object is generated and after that, each node generates a request, with the rate  $\lambda$  to lookup that popular object. In this simulation, for each time interval, the total number of generated messages is counted, and the simulation ends at the simulation time of 200.

The result is illustrated in Figure 14, which shows 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. This shows that the SWOP network has a better performance on handling traffic burst in the flash crowd scenario because it makes use of the clustering property to reduce requests' lookup distances.

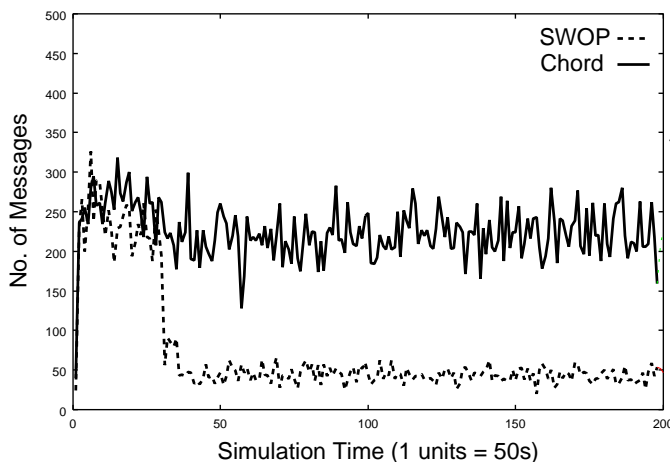


Figure 14: Effect of the traffic loading for static flash crowd of SWOP & the Chord network.

## 5 Related Work

The small world phenomenon was first observed by Milgram[10] and he discovered the interesting six degrees of separation in social network. Kleinberg [6, 7] provided the theoretical framework in analyzing graphs with small-world properties. In [3], authors studied the broadcast mechanism of communication in a small-world network. In [1], authors presented the condition of switching from a regular lattice to a small-world graph. In [5], authors proposed a system architecture to handle the popular files replication and their version updates. Comparing to our work, our system makes use of the small world structure which can reduce the popular file spreading distance significantly. Moreover, our system is built on top of the

DHT network, which preserves many attractive properties provided by the DHT network. In [19], authors proposed a scheme of storing data in an *unstructured* P2P network such as Freenet so that the P2P network may exhibit some of the small-world properties. Comparing to our work, our proposal is concentrated on structured networks which use consistent hashing function. Moreover, we apply it to resolve the dynamic flash crowd problem which is not applicable in an unstructured P2P network because an object can be stored in many different nodes. In [15], authors proposed the small world overlay structure on top of the *unstructured* and decentralized peer-to-peer network, such as Gnutella. In contrast to our work, we focus on the *structured* and decentralized peer-to-peer network, such as Chord. In [4], authors proposed to form a small-world P2P network for scientific communities. However, the detail of creating and managing a small-world P2P network was not clearly specified.

Recent research work on structured P2P network can be found in [9, 13, 17, 20]. The main feature of these work is to provide some form of data/topological structure for an overlay network so as to efficiently lookup a data object without generating a lot of query traffic. Comparing to our work, the SWOP protocol provides a better performance in object lookup and we propose an efficient way to “replicate” popular and dynamic objects and as a result, it helps the system to resolve the dynamic flash crowd problem. Ulysses[8] is a structured P2P based on the concept of butterfly network and shortcut intuition. The proposed P2P protocol achieves a low number of link traversal for performing an object lookup. However, the performance depends on a stable topology. If a query is routed between nodes which are doing a join or a leave operation, the performance is not known. Moreover, it considers only moderate traffic and does not address how to resolve heavy traffic workload like the flash crowd scenarios. In comparison, the proposed small-world P2P network can achieve a low number of link traversal for object lookup and one can also take the advantage of the high cluster coefficient effect to handle the dynamic flash crowd problem. For the Chord project, the Cooperative File System was proposed for a new peer-to-peer read only storage system, which helps to handle the flash crowd problem. But it has a large storage overhead to store the file information. In our small-world P2P network, it only requires a small amount of additional data storages of  $O(G+k)$  routing table entries to handle flash crowd problem.

In [16], the authors proposed a protocol to handle the static flash crowd problem on a P2P network. An elegant analysis of the static flash crowd problem was presented in [14]. Our work focuses on the dynamic flash crowd problem since many popular objects may have a time varying content.

## 6 Conclusion

The small-world effect is an active field of research in social sciences, Physics and Mathematics. A small-world graph has two important properties: low average hop distance between two randomly chosen nodes and high clustering coefficient. In this paper, we propose a set of protocols to create and manage a small-world structured P2P network. We show that the proposed small-world P2P network contains both a small average hop distance and a high clustering coefficient properties. We demonstrate how the low average hop distance between two random chosen nodes can reduce the number of link traversal for object lookup. A high clustering coefficient provides the "potential" advantage to handle the flash crowd problem. We propose a protocol to replicate popular and dynamic object so as to handle the dynamic flash crowd problem. Experiments were carried out to compare the performance of the proposed small-world P2P network with other structured P2P system (e.g., Chord). We show that the small-world P2P network has a lower object lookup latency and can satisfy many users who are requesting for the popular object.

**Acknowledgment:** We would like to thank the anonymous reviewers for their insightful comment. The work of John C.S. Lui was supported in part by RGC Grant and 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 and M. Weight. On the properties of small-world network models. *The European Physical Journal*, 13:547–560, 2000.
- [2] U. Bhat. Elements of applied stochastic processes. *John Wiley & Son, New York*, 1984.
- [3] F. Comellas and M. Mitjana. Broadcasting in small-world communication networks. *Proc. 9th Int. Coll. on Structural Information and Communication Complexity*, 13:73–85, 2002.
- [4] A. Iamnitchi, M. Ripeanu, and I. Foster. Locating data in (small-world?) peer-to-peer scientific collaborations. *First International Workshop on Peer-to-Peer Systems, Cambridge, MA.*, March, 2002.
- [5] B. W. Ian Clarke, Oskar Sandberg and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes In Computer Science*, 2009:46+, 2001.
- [6] J. Kleinberg. Navigation in a small-world. *Nature*, 406, 2000.
- [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, and X. Yu. Ulysses: A robust, low-diameter, low-latency peer-to-peer network.
- [9] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly.
- [10] S. Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.
- [11] M. E. J. Newman, I. Jensen, and R. M. Ziff. Percolation and epidemics in a two-dimensional small world. *Phys. Rev. E65*, 2002.
- [12] E. Parzen. Stochastic processes. *Holden-Day, San Francisco, California*, 1962.
- [13] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *18th IFIP/ACM Int. Conference on Distributed System Platforms*, pages 329–350, Nov,2001.
- [14] D. Rubenstein and 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 and E. Zegura. Adding structure to unstructured peer-to-peer networks: the use of small-world graphs. *Journal of Parallel and Distributed Computing*, 65(2):142–153, Feb,2005.
- [16] A. Stavrou, D. Rubenstein, and S. Sahu. A lightweight, robust p2p system to handle flash crowds. *IEEE ICNP*, November, 2002.
- [17] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. pages 149–160.
- [18] D. Watts. *Small-worlds: The dynamics of Networks between order and randomness*. Princeton University Press, 1999.
- [19] H. Zhang, A. Goel, and R. Govindan. Using the small-world model to improve freenet performance. *Proc. IEEE Infocom*, 2002.
- [20] B. Y. Zhao, J. Kubiawicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. pages Tech. Report, UCB/VSD–01–1141, U.C. Berkeley, 2001.