Constructing an Optimal Server Set in Structured Peer-to-Peer Networks

Seung Chul Han and Ye Xia

***Abstract*— To achieve high performance and resilience to failures, a client can make connections with multiple servers simultaneously and receive different portions of the data from each server in parallel. However, selecting the best set of servers from the set of all nodes that have the desired data is not a straightforward task, and the obtained performance can dramatically vary depending on the constructed server set. In this paper, we present two server selection schemes that generate optimal server sets with respect to the degree of interference(DOI) criterion and the worst link stress(WLS) criterion in a structured peer-to-peer network. After examining the correctness of the algorithms, we present simulation results demonstrating the benefits of the optimal server selection schemes. Through the simulation results, we conclude that the optimal selection schemes perform better than the random server-selection scheme in the following aspects: (1) load-balancing in the network and the worst-case link stress, (2) network resource used by the connections, including the number of links and total bandwidth, (3) response time, and (4) throughput of TCP connections.**

***Index Terms*— Peer-to-Peer Network, Parallel Access, Server Selection, Interference, Link Stress**

I. Introduction

IN THE last few years, peer-to-peer (P2P) networks have received significant attention. These P2P networks are distributed and decentralized systems, where each node is independent and equivalent in functionality and there are no nodes with a special responsibility to monitor or supervise the network. There have been immense research efforts on structured P2P networks (e.g., CAN, Chord, Pastry, and Tapestry [9], [10], [11], [12]) that provide a self-organizing substrate for large-scale P2P applications.

As the P2P networks quickly gains popularity, the demand for converting traditional applications onto the P2P networks is also growing fast. In [13], Knutsson, et. al., proposed the use of P2P infrastructure to support a massive multiplayer game (MMG) with thousands of players co-existing in the same game world. In [14], Tang and Xu presented a content-based full text search technique on a decentralized non-flooding P2P information retrieval system. In this paper, we are particularly concerned with the *parallel access* [2], [3], [4] technique and present optimal server-set construction schemes in structured P2P networks.

The conventional way of data downloading is that, when a client wants to obtain a specific object, the client identifies nodes containing the data and selects a server from which it downloads the data. The performance, e.g., the data downloading time, is directly influenced by the load of the server, the condition of the intermediate nodes on the path of the connection and any traffic fluctuations that may affect routing [3]. Even when a well-provisioned server has been chosen, throughput can fluctuate because the network traffic patterns may change during downloading. Moreover, for large volume of multimedia data transfer, a single server may not be able to satisfy the required quality of service due to insufficient resources such as bandwidth and computational power [6]. In order to alleviate these problems, the client could be allowed to connect to multiple servers simultaneously and receive different portions of the data from each server in parallel. Once all portions of the data are received, the client can recover the original data by reassembling the pieces. This technique is called *parallel access* in [2], [3]. According to the results in [2], [3], parallel access can achieve high performance and throughput, and is more resilient to route and link failures and traffic fluctuations than the traditional single-server scheme.

However, we identify one important problem. That is, given a client and a set of nodes containing the desired data, how to form a subset of nodes actually participating in transmitting the data. Constructing the best set of servers is not a trivial problem and the obtained performance can dramatically vary depending on the constructed server set [4].

There has been a number of techniques developed for the server selection problem. In [7], Carter and Crovella presented a dynamic server selection scheme based on instantaneous measurements of the round-trip time (RTT) and available bandwidth. In [6], Hefeeda et. al. proposed a server selection scheme that utilizes the underlying network topology and the performance information of the senders. The scheme constructs and annotates the topology connecting the candidate nodes with the client. Using the annotated topology, the selection algorithm determines the best server set. Similar research works are reported in [1], [2], [3], [8], where a typical strategy is to rank each server by one or more performance metrics (e.g., RTT, hop count, server load, bandwidth) and select the servers according to their ranks. However, most works are not explicitly designed for P2P networks.

Designing a server selection algorithm is not a straightforward task without server and network collaboration. The effectiveness of server selection schemes proposed in the previous works relies heavily on the accuracy of the information provided to the client. However, clients are seldom given accurate information about the server and network conditions in practice because of various reasons, e.g., security and

Manuscript received December 15, 2005; revised July 15, 2006.

The authors are with the Computer and Information Science and Engineering Department, University of Florida, Gainesville, USA (e-mail: {schan,yx1}@cise.ufl.edu).

Digital Object Identifier 10.1109/JSAC.2007.070117.

protocol/computation overhead. Moreover, it is not easy to get appropriate information in time due to frequent changes of the conditions. As a result, most of the previous research efforts only try to solve the server selection problem based on the RTT between the client and each server, but ignore the bandwidth bottleneck and interference created by connections to different servers sharing the same network links. The performance criterion of RTT is insufficient for applications such as media streaming, which is sensitive to bandwidth availability.

These facts motivate us to study techniques for server selection in a structured overlay network, in fact, a hypercube network, which is a special case of Plaxton-type networks [16]. With the assumption that the client can obtain a list of IDs of the nodes that contain the required data, known as the *candidates*, the client is able to determine the entire path from each candidate to the client. The client can then select k candidates, where k is known as the *degree of parallelism*, as actual servers according to a cost-minimizing criterion that reflects network resource usage and interference among the connections from the servers to the client.

This paper is organized as follows. In Section II, some assumptions and definitions are provided. In Section III, we present two optimal server-set construction algorithms, prove their optimality and calculate theoretical running time. We evaluate these algorithms through simulations in Section IV. Finally, the conclusions are drawn in Section V.

II. Preliminaries

The network we consider is a hypercube network with N nodes, numbered $0, 1, \ldots, N-1$. Suppose $N = 2^m$, where m is a natural number. The node IDs can be expressed as binary strings, and two nodes are linked with an edge if and only if their binary strings differ in precisely one bit. The routing rule is as follows. At each node s and for the destination d, let l be the first bit position, counting from the right, that s and d have different values. Then, the next hop on the route to d is the neighbor of s that differs with s in the l^{th} bit. As a result, routing from any node toward the destination node along the network edges can be viewed as changing the ID of the former to that of the latter one bit at each hop. Consider the example where $N = 2^5$, the source node is 10110 and the destination node is 00000. The route consists of the following sequence of nodes: $10110 \rightarrow 10100 \rightarrow 10000 \rightarrow 00000$.

A hypercube network is a special instance of Plaxton-type networks [16], which are broad enough to also include Pastry and Tapestry, and are fundamentally related to Chord and CAN. We note that this particular construction of the overlay network clearly requires that every position of the name space is occupied by a node, which is our current assumption.

A helpful device for visualizing the hypercube and its routing is the embedded tree rooted at node 0, as shown in Fig. 1, consisting of all valid routes allowed by the routing rule from all nodes to node 0. It is known that such a tree is a binomial tree [5]. For the k-level binomial tree B_k, there are exactly $k!/(k-i)!$ nodes at depth i, $0 \leq i \leq k$, and the height of the tree is k. The binomial tree embedded in the P2P network and rooted at node 0 is a labelled tree, where the label of each node is the node's ID. It is easy to verify that the node IDs agree with the following labelling rule.

Lemma 2.1: The m-digit root node ID is 00...0. If a node is the root of an i-level binomial tree, its ID is the same as its parent's ID except the $i+1^{th}$ right most bit, where $0 \leq i \leq m-1$.

By symmetry of the P2P network, there is an embedded binomial tree rooted at every node where the tree paths are the valid routes to the root node. Given the labelled binomial tree rooted at node 0, an easy way to derive the labels for the binomial tree rooted at node $d \neq 0$ is to XOR the node labels of the former tree with d.

Throughout the paper, let us denote the m-level binomial tree rooted at node 0 by $\mathcal{T} = B_m$. Let us denote the ID of node u by $I(u)$. When there is no confusion, we will use u and $I(u)$ interchangeably to denote node u.

From the viewpoint of data transmission, all nodes in the P2P network belong to four categories: the client, the candidate nodes, the server nodes and the irrelevant nodes. Without loss of generality, the root node 0 is understood as the client. The candidate nodes are those that contain the data and the server nodes are those that are eventually selected to participate in transmitting the data. The problem is for the client to select k server nodes from the set of candidate nodes in $\mathcal{T}$.

The following lemmas and corollary are simple consequences of the labelling scheme for the binomial tree.

Lemma 2.2: Suppose u is the parent of node v in $\mathcal{T}$. Then, u and v differ in exactly one bit, and the value of the bit is 0 for u and 1 for v. All bits to the right of the different bit are 0's.

Lemma 2.3: Suppose u is a node in $\mathcal{T}$, and $I(u) = a_1 \ldots a_i\ 0 \ldots 0$, $0 \leq i < m$. Then,
(i) the subtree rooted at u contains all nodes whose IDs fall between $a_1 \ldots a_i 0 \ldots 0$ and $a_1 \ldots a_i 1 \ldots 1$, inclusive;
(ii) $I(u) \leq I(v)$ for all nodes v in the subtree;
(iii) if the subtree contains nodes v and w, then it contains all nodes whose IDs fall between $I(v)$ and $I(w)$;
(iv) if v is a descendant of u, the longest common prefix of $I(u)$ and $I(v)$ is greater than i.

Lemma 2.4 (Routing Lemma): To go from any node toward the root along the tree path corresponds to converting the rightmost bit with value 1 to 0 in each step on the route.

Corollary 2.5: Suppose that, in $\mathcal{T}$, node v's ID is $a_1 \ldots a_m$. Then, a node u is an ancestor of node v if and only if the ID of u has the form $a_1 \ldots a_i 0 \ldots 0$, where $0 \leq i < m$[1], and $I(u) \neq I(v)$.

Definition 2.1: A **common ancestor** of a set of nodes $S = \{s_1, \ldots, s_n\}$, where $n \geq 2$, in a rooted tree is a node u satisfying the condition that, for each i, $1 \leq i \leq n$, either u is an ancestor of s_i or u is s_i.

Definition 2.2: The **lowest common ancestor** (LCA) of a set of nodes $S = \{s_1, \ldots, s_n\}$, where $n \geq 2$, in a rooted tree is the deepest node in the tree that is a common ancestor of all nodes in S. It is denoted by $LCA(S)$ or $LCA(s_1, \ldots, s_n)$.

Definition 2.3: Suppose $S = \{s_1, \ldots, s_n\}$ is a set of nodes in $\mathcal{T}$ and $E = \{e_1, \ldots, e_l\}$ is the set of edges used by the

[1]By convention, when $i = 0$, $a_1 \ldots a_i 0 \ldots 0$ is $0 \ldots 0$.

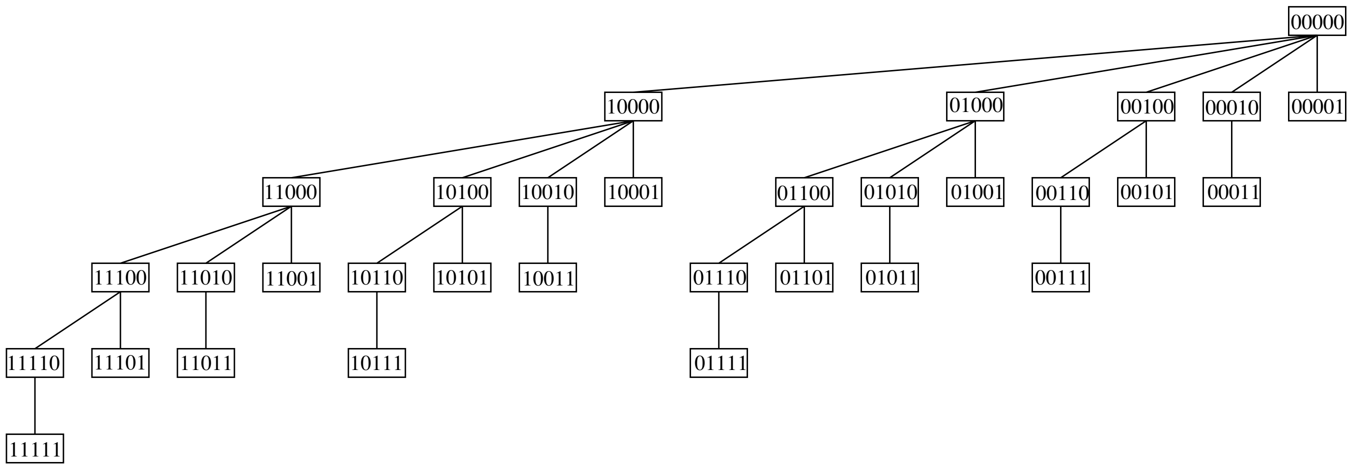

Fig. 1. A labelled 5-level binomial tree B_5

paths from nodes in S to the root, called the S**-paths**. The **degree of interference** (DOI) of nodes $s_1, \ldots, s_n$, denoted by $d(S)$ or $d(s_1, \ldots, s_n)$, is

$$d(s_1, \ldots, s_n) \quad = \quad \sum_{i=1}^{l}(\text{number of } S\text{-paths via } e_i - 1)$$

For example in Fig. 1, if $u = 01100$, $v = 01101$ and $w = 01010$, then $d(u, v, w) = 3$ because the number of S-paths on edge (01100, 01000) is 2 and on edge (01000, 00000) is 3.

III. Optimal Server Selection Algorithms

A. Minimizing DOI

1) Algorithm and Computational Complexity: The problem addressed in this section can be stated as follows. Given the client and a set of nodes containing the data (*the candidate nodes*), select k (*the degree of parallelism*) servers from the candidate set to participate in transmitting the data to the client so that the degree of interference (DOI) among the servers is minimized. For example, in Fig. 1, assume the set of candidates is $S = \{01100, 01110, 10011, 11111\}$ and k is 3. If the client constructs a server set $G = \{01100, 01110, 10011\}$, then $d(G)$, *the degree of interference*, would be 2. On the other hand, if the client constructs a server set $G' = \{01100, 10011, 11111\}$, then $d(G') = 1$.

Our node selection algorithm, Algorithm 1, takes two arguments, a set of candidates and the degree of parallelism, as inputs and returns an optimal set of servers that has the minimum DOI. The algorithm proceeds in two phases. In the first phase (line 3 through 8), the client sorts the candidates in increasing order of the node ID and calculates the DOI of each candidate with its predecessor in the sorted list. (By convention, the first node's DOI with its predecessor is 0.) In the second phase (line 9 through 14), the client selects k nodes in increasing order of the calculated DOIs.

The running time of the algorithm is as follows. In the first phase, sorting takes $O(n \log n)$ comparisons and calculating all the $d(s_{i-1},\ s_i)$'s takes $O(n)$ operations, each involving comparison of the bits in $I(s_{i-1})$ and $I(s_i)$ by Lemma 3.1 and Lemma 3.3 in the following. In the second phase, one approach is to sort the $d[j]$'s first, and then remove the least element in each step of the loop. This takes $O(k + n \log n)$

Algorithm 1 SERVER-SELECTION-DOI(S, k)

```
1: input: $S = \{s_1, \ldots, s_n\}$, a candidate server set; an integer
   $k \le n$
2: output: $G$, an optimal set with $k$ servers w.r.t. the DOI
   cost
3: $G \leftarrow \emptyset$
4: Sort $S$ in increasing order of the node (server) ID
5: $d[1] = 0$
6: for $i = 2$ to $n$ do
7:     $d[i] = d(s_{i-1}, s_i)$
8: end for
9: for $i = 1$ to $k$ do
10:    Find $s_j$ in $S$ whose $d[j]$ is the minimum
11:    Add $s_j$ to the server set $G$
12:    Remove $s_j$ from $S$
13: end for
14: return $G$
```

s. Since $k \le n$, this approach in fact takes $O(n \log n)$ s. The second approach is to remove the least $d[j]$ from an unordered list in each step of the loop. For ease of presentation, Algorithm 1 only lists this approach (line 9 through 13). This takes a total $O(kn)$ s. Hence, the second phase takes the minimum of $O(n \log n)$ and $O(kn)$ s. Overall, the algorithm takes $O(n \log n)$ s. Operations such as comparison, arithmetic or bit-wise operation of two m-bit numbers take $O(m)$ running time each. Thus, the total running time of the algorithm is $O(nm \log n)$. For practical problems, m is almost always small enough, e.g., 32 or 64. Hence, the practical running time of the algorithm is $O(n \log n)$.

2) Proof of Optimality: Let us denote the *longest common prefix* (LCP) of a set of m-digit binary labels, $R = \{b^1, \ldots, b^k\}$, by $LCP(b^1, \ldots, b^k)$ or $LCP(R)$, and denote the m-digit label equal to the concatenation of $LCP(R)$ with an appropriate number of 0's by $\lambda(R)$ or $\lambda(b^1,\ \ldots,\ b^k)$. For instance, $LCP(01110,\ 01011) = 01$, and $\lambda(01110,\ 01011) = 01000$. For a set of nodes $S = \{s_1, ..., s_n\}$, $LCP(S)$ or $LCP(s_1, \ldots, s_n)$ are the simplified notations for $LCP(I(s_1),\ \ldots,\ I(s_n))$, and $\lambda(S)$ or $\lambda(s_1, \ldots, s_n)$ are the simplified notations for $\lambda(I(s_1),\ \ldots,\ I(s_n))$.

Lemma 3.1: Suppose $S = \{s_1, \ldots, s_n\}$, where $n \geq 2$, is a set of nodes in $\mathcal{T}$. Then, the node ID of $LCA(S)$ is $\lambda(s_1, \ldots, s_n)$.

Proof: Suppose $w = LCA(S)$. Suppose $LCP(S)$ has i digits, for some $0 \leq i < m$. Without loss of generality, suppose that, given two nodes $u, v \in S$, $I(u)$ is $a_1 \ldots a_i 0 * \ldots *$ and $I(v)$ is $a_1 \ldots a_i 1 * \ldots *$, where the wildcard $*$ means the bit value can be either 0 or 1. Such u and v must exist because, otherwise, $LCP(S)$ would have more than i digits. The claim is that $I(w)$ must be $a_1 \ldots a_i 0 \ldots 0$. By Corollary 2.5, such a node w is an ancestor of every node in S, other than itself if $w \in S$. But, any common ancestor of u and v must have an ID of the form $a_1 \ldots a_j 0 \ldots 0$ where $0 \leq j \leq i$. Hence, $w = LCA(u, v)$, and therefore, $w = LCA(S)$. ■

Let us define some partial orders for comparing two node IDs that correspond to the ancestor-descendant relationship of two nodes in $\mathcal{T}$. Given two nodes u and v with IDs $I(u) = x$ and $I(v) = y$, we write $x \prec y$ (or $y \succ x$) if node u is an ancestor of node v; we write $x \preceq y$ (or $y \succeq x$) if either node u is an ancestor of node v or u and v are the same node.

Lemma 3.2: Let x and y be two node IDs. Then,
(i) $x \prec y$ if and only if, after removing all trailing 0's from both x and y, x is a prefix of y and $x \neq y$; $x \preceq y$ if and only if, after removing all trailing 0's from both x and y, x is a prefix of y.
(ii) $x \prec y$ ($x \preceq y$) implies $x < y$ ($x \leq y$, respectively).
(iii) If S is a set of nodes and $R \subseteq S$, then, $\lambda(S) \preceq \lambda(R)$.

Proof: Properties (i) and (ii) are simple consequences of Corollary 2.5. Property (iii) is a result of Corollary 2.5 and Lemma 3.1. ■

Lemma 3.3: Suppose S is a set of two nodes in $\mathcal{T}$. The DOI, $d(S)$, is the depth of $LCA(S)$.

Proof: The paths from both nodes in S to the root, i.e., the S-paths, overlap only on the segment from $LCA(S)$ to the root. ■

Lemma 3.4: Suppose x, y and z are three binary labels of m digits. Suppose that, when interpreted as unsigned integers, $x > y > z$. Then, $\lambda(x, z) \preceq \lambda(x, y)$, $\lambda(x, z) \preceq \lambda(y, z)$, and $\lambda(x, z) = \min\{\lambda(x, y), \lambda(y, z)\}$.

Proof: Suppose $LCP(x, y)$ has j digits and $LCP(x, z)$ has i digits. We claim that $0 \leq i \leq j < m$. Suppose otherwise, i.e., $j < i$. Without loss of generality, let

$$\begin{aligned} x &= a_1 \ldots a_j 1 a_{j+2} \ldots a_i 1 * \ldots * \\ y &= a_1 \ldots a_j 0 * \quad \ldots \quad \ldots \quad * \\ z &= a_1 \ldots a_j 1 a_{j+2} \ldots a_i 0 * \ldots * \end{aligned}$$

This implies $z > y$, contradicting the hypothesis of the lemma. Hence, it must be true that $i \leq j$. In the case $i = j$, x, y and z must have the following form.

$$\begin{aligned} x &= a_1 \ldots a_i 1 * \ldots * \\ y &= a_1 \ldots a_i 0 * \ldots * \\ z &= a_1 \ldots a_i 0 * \ldots * \end{aligned}$$

We see that $\lambda(x, z) = \lambda(x, y)$ and that $\lambda(x, z) \preceq \lambda(y, z)$. In the case $i < j$, x, y and z must have the following form.

$$\begin{aligned} x &= a_1 \ldots a_i 1 a_{i+2} \ldots a_j 1 * \ldots * \\ y &= a_1 \ldots a_i 1 a_{i+2} \ldots a_j 0 * \ldots * \\ z &= a_1 \ldots a_i 0 * \quad \ldots \quad \ldots \quad * \end{aligned}$$

We see that $\lambda(x, z) \preceq \lambda(x, y)$ and that $\lambda(x, z) = \lambda(y, z)$. In both cases, by Lemma 3.2, we have $\lambda(x, z) = \min\{\lambda(x, y), \lambda(y, z)\}$. ■

Lemma 3.5: Suppose that u, v and w are distinct nodes in $\mathcal{T}$ and that $I(u) > I(v) > I(w)$. Then,

$$d(u, w) = \min\{d(u, v), d(v, w)\}$$

Proof: We will apply Lemma 3.4. Without loss of generality, suppose $\lambda(I(u), I(w)) = \lambda(I(u), I(v))$ and that $\lambda(I(u), I(w)) \preceq \lambda(I(v), I(w))$. By Lemma 3.1, $LCA(u, w) = LCA(u, v)$, and $LCA(u, w)$ is either an ancestor of $LCA(v, w)$ or $LCA(v, w)$ itself. By Lemma 3.3, $d(u, w) = d(u, v)$ and $d(u, w) \leq d(v, w)$. ■

Lemma 3.5 indicates that the closer are two nodes in the name space, the higher DOI they have.

Lemma 3.6: Suppose $(s_1, \ldots, s_n)$ is a sorted list of distinct nodes in $\mathcal{T}$ by the node ID, for some $1 < n \leq 2^m$. Then,

$$d(s_1, s_n) \quad = \quad \min\{d(s_1, s_2), d(s_2, s_3), \ldots, d(s_{n-1}, s_n)\}$$

Proof: Without loss of generality, assume that $(s_1, \ldots, s_j)$ is sorted in decreasing order of the node ID. The proof will be based on induction on n. The case of $n = 2$ is trivial. The base case $n = 3$ is proven in Lemma 3.5.

Let us make the induction hypothesis that the lemma is true for the subset $\{s_1, \ldots, s_l\}$, where $3 \leq l < n$. We will show it is true for $\{s_1, \ldots, s_{l+1}\}$. By Lemma 3.5,

$$d(s_1, s_{l+1}) = \min\{d(s_1, s_l), d(s_l, s_{l+1})\}$$

By the induction hypothesis,

$$d(s_1, s_l) = \min\{d(s_1, s_2), \ldots, d(s_{l-1}, s_l)\}$$

By combining the two inequalities, we are done. ■

Lemma 3.7: Suppose $(s_1, \ldots, s_n)$ is a sorted list of distinct nodes in $\mathcal{T}$ by their IDs, for some $1 < n \leq 2^m$. Then,

$$LCA(s_1, \ldots, s_n) = LCA(s_1, s_n)$$

Proof: Without loss of generality, suppose $(s_1, \ldots, s_n)$ is sorted in decreasing order of the node ID. By Lemma 3.1, we need to show

$$\lambda(s_1, \ldots, s_n) = \lambda(s_1, s_n) \qquad (1)$$

First, we claim that $\lambda(s_1, \ldots, s_n)$ must be equal to the λ-value of some two nodes in the set $(s_1, \ldots, s_n)$. Suppose this is not true. We must have

$$\lambda(s_i, s_{i+1}) \succ \lambda(s_1, \ldots, s_n)$$

for all $i = 1, \ldots, n-1$, which implies

$$\lambda(s_1, \ldots, s_n) \succ \lambda(s_1, \ldots, s_n).$$

Next, (1) is trivially true for $n = 2$. In the general case of $n \geq 3$, suppose that, for some $1 \leq l < k \leq n$,

$$\lambda(s_1, \ldots, s_n) = \lambda(s_l, s_k)$$

But, by Lemma 3.2 and Lemma 3.4,

$$\lambda(s_l, s_k) \geq \lambda(s_l, s_n) \geq \lambda(s_1, s_n)$$

Hence,

$$\lambda(s_1, \ldots, s_n) \geq \lambda(s_1, s_n).$$

Again, by Lemma 3.2,

$$\lambda(s_1, \ldots, s_n) \leq \lambda(s_1, s_n).$$

Hence, equality holds and (1) is true. ■

For each node u in $\mathcal{T}$, denote the depth of u by $l(u)$. By convention, the root has depth 0.

Theorem 3.8: Suppose $S = (s_1, \ldots, s_n)$ is a sorted list of distinct nodes in $\mathcal{T}$ by their IDs, for some $1 < n \leq 2^m$. Then,

$$d(S) = \sum_{i=1}^{n-1} l(LCA(s_i, s_{i+1}))$$

Proof: We will consider the contribution to $d(S)$ by each S-path. Let the S-path from node $s_i \in S$ be P_i. Suppose we add the S-paths to the tree one at a time, in the order P_1, ..., P_n. We will prove the theorem by induction. P_1 does not contribute to $d(S)$. P_2 meets with P_1 at node $LCA(s_1, s_2)$. The contribution to $d(S)$ by P_2 is $l(LCA(s_1, s_2))$. Let us hypothesize that the total contribution to $d(S)$ after P_i is added, for $2 \leq i \leq n-1$, is $\sum_{j=1}^{i-1} l(LCA(s_j, s_{j+1}))$. We claim that, after adding P_{i+1}, the additional contribution to $d(S)$ by P_{i+1} is $l(LCA(s_i, s_{i+1}))$. First, P_{i+1} meets P_i at node $v_i = LCA(s_i, s_{i+1})$. Hence, P_{i+1} overlaps with P_i at the path segment from v_i to the root (Node v_i can be the root itself.). The contribution by P_{i+1} is no less than $l(v_i)$. Consider any path P_j, where $1 \leq j < i$. By Lemma 3.7, $LCA(s_j, \ldots, s_{i+1}) = LCA(s_j, s_{i+1})$. Hence, $LCA(s_j, s_{i+1})$ is either the same as, or an ancestor of, $LCA(s_i, s_{i+1})$. Therefore, if P_{i+1} overlaps with P_j on any edges, those edges must be on the path segment from v_i to the root. In other words, P_{i+1} does not overlap with earlier-added paths on edges that have not yet been accounted for. Hence, the contribution to $d(S)$ by P_{i+1} is precisely $l(v_i)$. We are done with the induction proof. ■

Theorem 3.9: Suppose $(s_1, \ldots, s_n)$ is a sorted list of distinct nodes in $\mathcal{T}$ by their IDs, for some $1 < n \leq 2^m$. Then,

$$d(s_1, \ldots, s_n) = d(s_1, s_2) + d(s_2, s_3) + \ldots + d(s_{n-1}, s_n)$$

Proof: The proof is by applying Theorem 3.8 and Lemma 3.3. ■

Theorem 3.10: Suppose $S = (s_1, s_2, \ldots, s_n)$ is a sorted list of distinct candidate nodes by the increasing order of the node ID and $pred(u)$ is the predecessor of u in S.[2] For a fixed k, where $0 < k \leq n$, let $G = \{g_1, \ldots, g_k\}$ and $G \subseteq S$. Suppose G satisfies the condition that, for any other set $H \subseteq S$ of k elements denoted by $H = \{h_1, \ldots, h_k\}$,

$$\sum_{i=1}^{k} d(pred(g_i), g_i) \leq \sum_{i=1}^{k} d(pred(h_i), h_i) \quad (2)$$

Then, G is a minimal cost set of k elements, i.e., $d(G)$ is the minimum over all k-element subsets of S.

[2]By convention, $pred(s_1) = s_1$, and $d(s_i, s_i) = 0$.

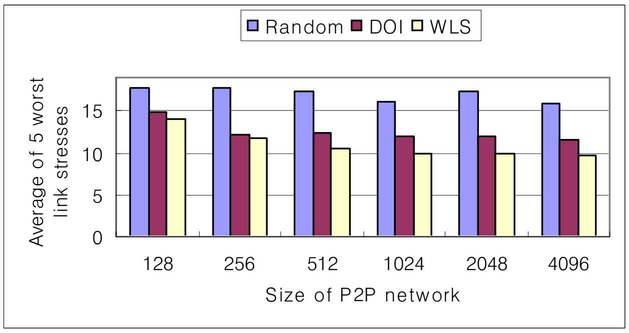

Fig. 2. The average of 5 worst link stresses

Proof: For simplicity, let $c(g_i) = d(pred(g_i), g_i)$. For a list of l numbers, $(z_1, \ldots, z_l)$, where $l \geq 1$, let us sort the list in non-decreasing order of their values. The ordering among repeated numbers is arbitrary but fixed. Let $(z_1, \ldots, z_l)_{(i)}$ denote the i^{th}-entry from the left in the resulting sorted list. If all numbers in the list were distinct, it would be the i^{th}-smallest number in the list. Note that (2) is equivalent to

$$(c(g_1), \ldots, c(g_k))_{(i)} \leq (c(h_1), \ldots, c(h_k))_{(i)} \quad (3)$$

for $i = 1, \ldots, k$.

Let us re-order the nodes in S and denote the resulting ordered list by $(v_1, \ldots, v_n)$, $v_i \in S$ for each i, so that $(c(v_1), \ldots, c(v_n))$ is in non-decreasing order. For each i, $1 \leq i \leq n$, v_i is the node such that $c(v_i) = (c(s_1), \ldots, c(s_n))_{(i)}$. Loosely speaking, node v_i achieves the "i^{th}-smallest" value in $(c(s_1), \ldots, c(s_n))$.

B. Minimizing Worst Link Stress

Consider the server set $\{s_1, v_1, \ldots, v_{k-1}\}$. Clearly, this set is minimal in the sense of (2). We will show that it is also optimal with respect to the DOI cost criterion. (I.e., it has the minimum DOI among all subsets of S with k elements.)

Assume $\{s_{i_1}, \ldots, s_{i_k}\}$, $1 \leq i_1 < \ldots < i_k \leq n$, is optimal with respect to the DOI cost criterion. By Lemma 3.6, for every $1 \leq j \leq k-1$, there exists some a_j, where $i_j + 1 \leq a_j \leq i_{j+1}$ such that $d(s_{i_j}, s_{i_{j+1}}) = c(s_{a_j})$. Clearly,

$$c(v_i) \leq (c(s_{a_1}), \ldots, c(s_{a_k}))_{(i)}, \quad i = 1, \ldots, k-1 \quad (4)$$

In light of (3), we need to show that equality must hold in the above inequality, for all $i = 1, \ldots, k-1$. Otherwise, summing of both side, we get

$$\sum_{i=1}^{k-1} c(v_i) < \sum_{i=1}^{k-1} c(s_{a_i}). \quad (5)$$

Consider the server set $\{s_1, v_1, \ldots, v_{k-1}\}$. Let the corresponding sorted list according to the node ID be $(s_1, v_{i_1}, \ldots, v_{i_{k-1}})$. By Lemma 3.6, $d(s_1, v_{i_1}) \leq c(v_{i_1})$, and $d(v_{i_j}, v_{i_{j+1}}) \leq c(v_{i_{j+1}})$ for $1 \leq j \leq k-2$. By Theorem 3.9,

$$d(s_1, v_1, \ldots, v_{k-1}) = d(s_1, v_{i_1}) + \sum_{j=1}^{k-2} d(v_{i_j}, v_{i_{j+1}})$$

Hence,

$$d(s_1, v_1, \ldots, v_{k-1}) \leq \sum_{j=1}^{k-1} c(v_{i_j}) = \sum_{j=1}^{k-1} c(v_j) \quad (6)$$

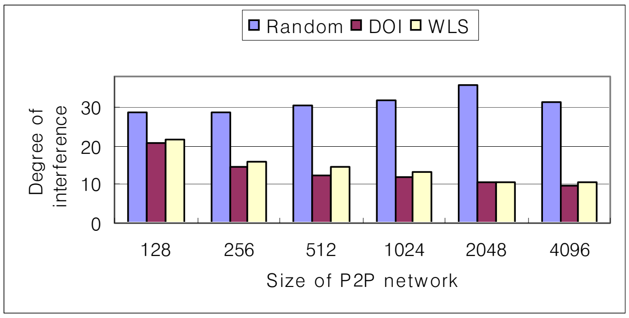

Fig. 3. The degree of interference of the servers

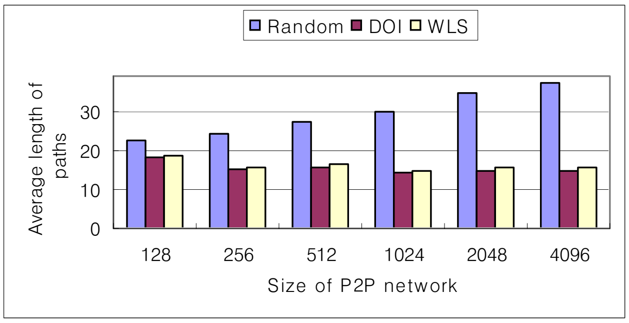

Fig. 4. The average length of the paths from the servers to the client

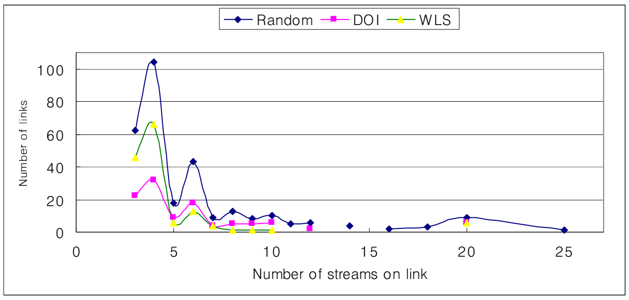

Fig. 5. Distribtuion of link stress

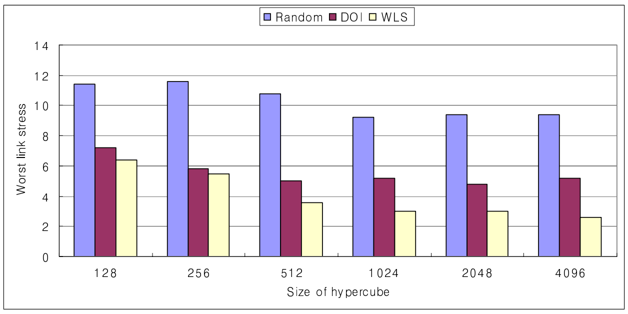

Fig. 6. The worst case link stress of an overlay link

(By the assumption that node v_i achieves the "i^{th}-smallest" value in $(c(s_1),\dots,c(s_n))$, equality must hold in the above. But, this doesn't matter.)

Again by Theorem 3.9,

$$d(s_{i_1},\dots,s_{i_k}) = \sum_{i=1}^{k-1} c(s_{a_i}). \tag{7}$$

By (5), (6) and (7), $d(s_1, v_1, \dots, v_{k-1}) < d(s_{i_1},\dots,s_{i_k})$, which contradicts the assumption that $(s_{i_1},\dots,s_{i_k})$ is optimal.

In fact, we have shown not only that equality holds in (4) for all $i = 1,\dots,k-1$, but also that the set $\{s_1, v_1, \dots, v_{k-1}\}$ is optimal with respect to the DOI cost criterion. ∎

Definition 3.1: Suppose $S = \{s_1,\dots,s_n\}$ is a set of nodes in $\mathcal{T}$. The **link stress** of an edge e in $\mathcal{T}$, denoted by $LS(e)$, is the number of S-paths via e. Let E be the set of all edges in $\mathcal{T}$. The **worst-case link stress** (WLS) is defined as $\max_{e\in E} LS(e)$.

Definition 3.2: Given a subset, S, of two or more nodes in $\mathcal{T}$, let us denote the set of LCAs of all subsets of S with two or more nodes by $SLCA(S)$.

The problem is to select k nodes from S so that the WLS is minimized. The key observation is that the link stress of any edge (u, v), where u and v are nodes in $\mathcal{T}$ and u is the parent of v, is no less than that of any edge in the subtree $\mathcal{T}_v$, the subtree rooted at node v. Hence, the edge with the worst link stress must be connected to the root of $\mathcal{T}$.

The actual algorithm that we use minimizes the WLS recursively in the sense that it minimizes the WLS for every subtree rooted at every $v \in SLCA(S)$. The WLS for each such subtree, $\mathcal{T}_v$, is defined over only the edges in $\mathcal{T}_v$. That is, the WLS over $\mathcal{T}_v$ is $\max_{e\in E_v} LS(e)$, where E_v is the set of edges in $\mathcal{T}_v$. The incentives for recursively minimizing the WLS are that, first, the DOI cost can be simultaneously reduced, and second, in the more general problem involving multiple clients, this leads to a better chance of achieving lower link stress or DOI cost for heuristic algorithms built on top of the optimal single-client algorithm.

Although the high-level ideas about algorithm are mostly straight-forward, an efficient implementation is not. The analysis on its running time is not trivial either. Due to space limitation, we will not discuss the details of this algorithm. We simply point out that our implementation of the algorithm has a running time $O(nm)$ in practice. Furthermore, to run the algorithm, each client only needs to know the IDs of the candidates nodes.

IV. Evaluations

In this section, we present simulation results demonstrating the benefits of the optimal server selection schemes. We compare three selection algorithms, the DOI-minimizing scheme (Algorithm 1), the WLS-minimizing scheme (Section III-B) and the random scheme where the client chooses k servers uniformly at random (without replacement) from the list of candidate nodes. We use four performance metrics to evaluate the server selection schemes.

- **Worst case link stress**: We measure the stress of each link by counting the number of streams on the link and report the worst case link stress.
- **Degree of Interference (DOI)**: the DOI of the selected servers
- **Average path length**: The average is taken over the paths from the selected servers to the client.
- **Distribution of link stress**: We show the number of links in the network that contain j streams, for $j \geq 1$. This reflects how well the link stress are balanced over the links.

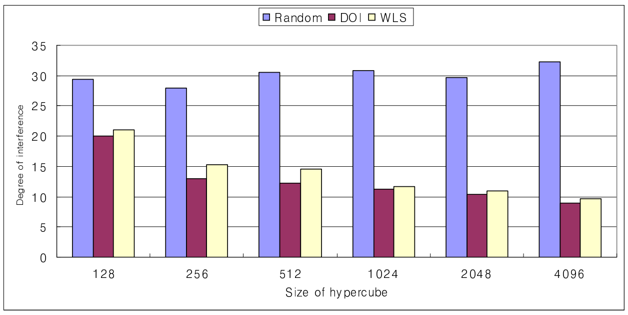

Fig. 7. The degree of interference of the servers for the overlay network

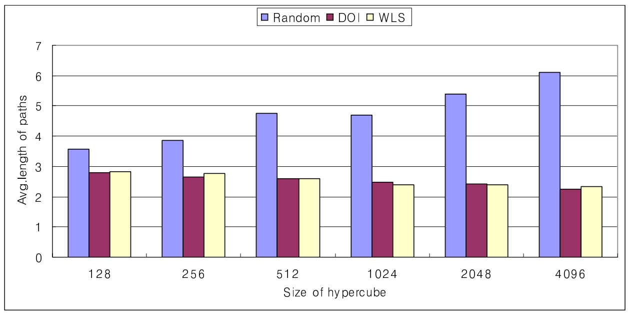

Fig. 8. The average length of the paths from the servers to the client at the overlay network

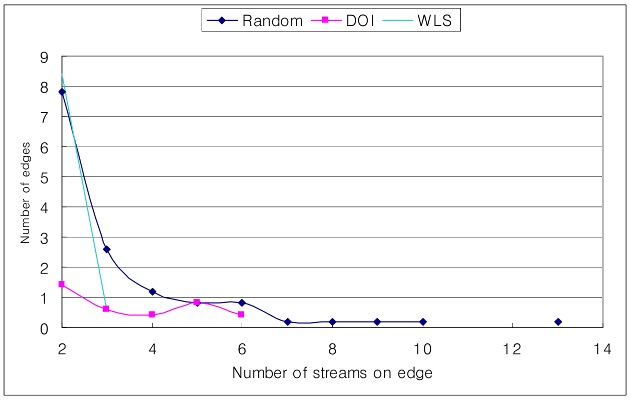

Fig. 9. Distribution of link stress at the overlay network

A. Single Client Case

In this experiment, we compare the goodness of server selection schemes for a single client. Note that the optimality of the two optimal server selection schemes are with respect to the overlay hypercube network. That is, the link stress and DOI are for each overlay virtual link, which is an end-to-end IP path. We will show by simulation that, despite being implemented at the overlay level, these algorithms lead to reduced link stress and DOI for the physical links at the underlay network as well.

Our simulations were performed on the Transit-Stub model [15] using a 2-level hierarchy of routing domains with transit domains that interconnect lower level stub domains. The simulations run on network topologies consisting of 4200 nodes. The average diameter of the network is 10. We vary the size of the overlay P2P networks from 128 to 4096 nodes and, from each network, a reasonable size (25%) subset of all nodes is chosen as the candidate set. The candidate nodes are uniformly distributed over the 4200 nodes. The client selects $k = 20$ servers from the candidate set. The simulator counts the number of streams on each physical link and assigns a constant delay to each link. It does not model either queuing delay or packet losses. We run each experiment with different random number generator seeds and present the average of the results obtained.

Fig. 2 shows the average number of streams on the five most stressed links, for different sizes of the P2P networks when the random scheme, DOI-minimizing scheme and WLS-minimizing scheme are used respectively. In all cases, the WLS-minimizing scheme generates the best results, and the random performs the worst. The optimal schemes are demonstrated to be effective in reducing the bottleneck stress. Improvement of up to 38% was observed. The WLS shows no clear trend as the P2P network size increases under the random scheme and the DOI-minimizing scheme. It decreases under the WLS-minimizing scheme due to the decrease in the density of the servers in the candidate set.

Fig. 3 shows the DOI of all servers for different sizes of the P2P networks. As we would expect, the DOI-minimizing scheme has clearly produced the best results over all the cases. It is interesting to notice that the WLS-minimizing scheme generates only slightly higher DOI than the DOI-minimizing scheme. The random scheme has up to three times as much DOI as the other two schemes.

In Fig. 4, we compare the average length of the paths from each selected server to the client. We see that the optimal schemes offer much shorter average path length than the random scheme. This means that the two optimal schemes can perform better than the random scheme in terms of response time due to shorter round trip time (RTT). If data transfer is conducted over TCP connections, shorter RTT typically leads to higher throughput. Furthermore, since the number of paths is constant, the two optimal schemes use much fewer links, hence, less network resource, than the random scheme.

Fig. 5 plots the distribution of the link stress when the size of the P2P network is 4096 nodes. Note that the curve for the random scheme slowly decreases and has a longer tail than the other two schemes. In the case of the WLS-minimizing scheme, most links have less than 10 streams on it. This suggests that, if we identify “load” with the number of streams on an link, the WLS-minimizing scheme is the best from the load-balancing point of view.

As a comparison, we show the simulation results for the same performance metrics at the overlay network in Fig. 6, 7, 8 and 9. In these figures, a link or an edge refers to an overlay link, which is an end-to-end path in the underlay network. The path length is measured in terms of the number of overlay links.

B. Multiple Clients Case

In this experiment, the P2P network has 4096 nodes, a quarter of which are candidate nodes. There are 100 clients, each selecting 20 servers from the shared candidate set. The candidate nodes and the clients are both uniformly distributed over the 4096 nodes. For server load control, if a node is selected 30 times by the clients, it is automatically removed

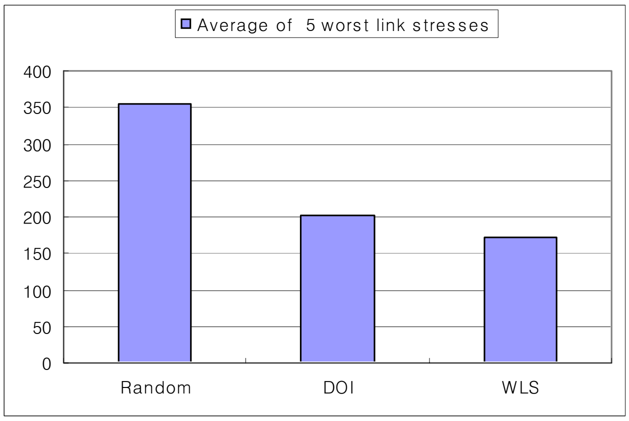

Fig. 10. The average of 5 worst link stresses for the case of 100 clients

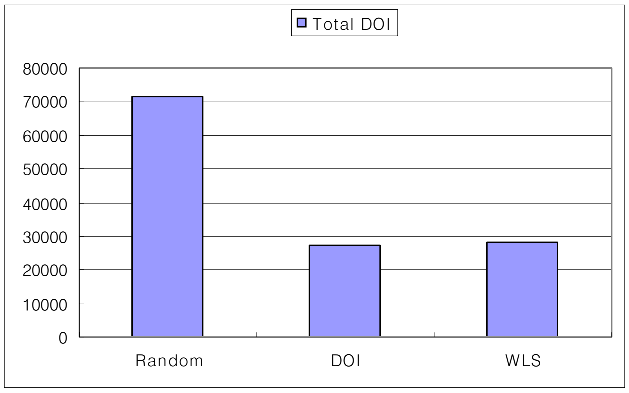

Fig. 11. The degree of interference of all connections for the case of 100 clients

from the candidate set. We expect that two good heuristic algorithms are for each client to independently apply the DOI-minimizing scheme or the WLS-minimizing scheme, which are only optimal if there is only one client. All performance metrics are with respect to the underlay physical network.

Fig. 10 depicts the average number of streams on the five most stressed links for the three selection schemes. The WLS-minimizing scheme still achieves the best result (173), as compared to the DOI-minimizing scheme (202) and the random (355.8) scheme. The former two schemes are still very effective at reducing the stress at the bottleneck link for the multiple clients case.

Fig. 11 shows the total DOI of all connections between each and every client and its selected servers. The DOI-minimizing scheme yields the lowest DOI, much lower DOI than the random scheme, and slightly lower than the WLS-minimizing scheme.

Fig. 12 compares the average path length of the connections between each and every client and its selected servers. The average path lengths of the DOI-minimizing and WLS-minimizing schemes are both 15 and that of the random scheme is 38.2. The former two schemes still perform much better than the random scheme, leading to lower response time and less network resource requirement.

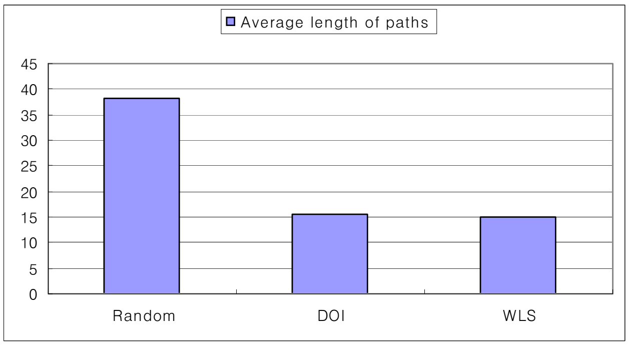

Fig. 12. The average path length of connections for the case of 100 clients

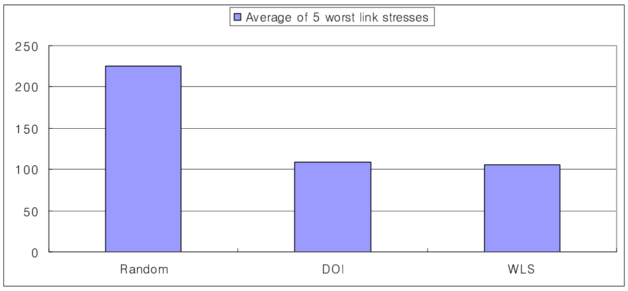

Fig. 13. The average of 5 worst link stresses for the case of 100 clients. The overlay network has a name space size 4096 and contains 2048 nodes.

C. Name Space Not Fully Occupied

In this section, we consider the situation where the name space is not fully populated by nodes. This is the most likely scenario for an ad-hoc P2P network with frequent node joins and leaves. We modify the routing as follows. When a message for destination d arrives at a node, the next node will be the first available node on the hypercube path to d. This results in an increase in the routing table size. However, one can substitute routing-table-based routing by on-demand routing. The hypercube path to the destination is probed before the data messages are sent, and the route is either encoded in the messages themselves or cached at the intermediate nodes.

The server selection strategy is, given the list of candidate, each client selects servers as if the network is a hypercube. In our simulation experiments, we again use a transit-stub model with 4200 nodes for the physical network. The overlay network has a name space size 4096, but only contains 2048 actual nodes, uniformly distributed throughout the name space. There are 100 clients, each selecting 20 servers from the shared candidate set of 1024 nodes. For our experiment, we constructed 10 instances of the random network. Fig. 13, 14 and 15 show the resulting worst link stress, DOI and average path length, respectively, each being the average value over the 10 network instances. Compared with the results for the fully-occupied hypercube in Section IV-B, the performance gain of the WLS-minimizing and DOI-minimizing schemes has not deteriorated.

V. Conclusions

In this paper, we have envisioned a structed P2P network and a routing protocol as described in Section II, and specified two node selection schemes that generate optimal solutions

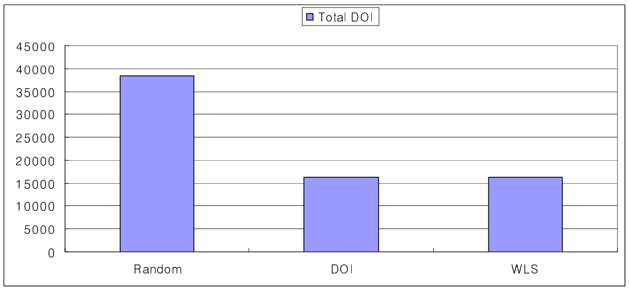

Fig. 14. The degree of interference of all connections for the case of 100 clients. The overlay network has a name space size 4096 and contains 2048 nodes.

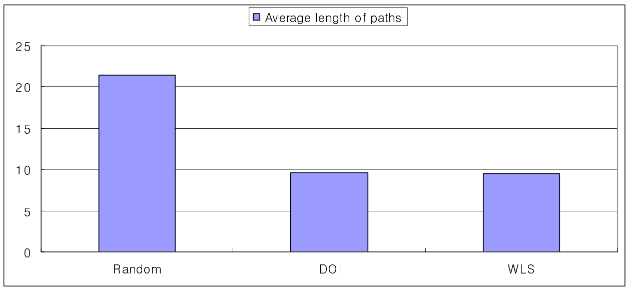

Fig. 15. The average path length of connections for the case of 100 clients. The overlay network has a name space size 4096 and contains 2048 nodes.

with respect to the degree of interference (DOI) criterion and the worst link stress (WLS) criterion. These optimal schemes do not require measurement of some performance metrics or the network topology. The only assumption is that the client can obtain the list of IDs of the candidate nodes that contain the required data. Both schemes exploit the recursive property of the embedded binomial tree. The DOI-minimizing scheme has a running time $O(nm \log n)$ in theory and $O(n \log n)$ in practice, where n is the number of nodes in the candidate set and 2^m is the size of the network. The WLS-minimizing scheme has a running time of $O(nm^2)$ in theory and $O(nm)$ in practice. After examining the correctness of the schemes, we have presented simulation results to demonstrate the benefits of the optimal server selection schemes. We conclude that the optimal selection schemes perform significantly better than the random node selection scheme in the following aspects: (1) load-balancing in the network and the worst-case link stress, (2) network resource used by the connections, including the number of links and total bandwidth, (3) response time due to shorter RTT, and (4) throughput of the connections, if TCP is used. The methodologies proposed in this paper can be extended in several directions, including applications to peer-to-peer file downloading or streaming, multicast group member selection, or edge-mapping in content distribution networks.

REFERENCES

[1] Micah Adler, Rakesh Kumar, Keith Ross, Dan Rubenstein, Torsten Suel and David Yao. Optimal Peer Selection for P2P Downloading and Streaming. *Proc. Of IEEE INFOCOM 2005.*

[2] Pablo Rodriguez, Adreas Kirpal and Ernst Biersack. Parallel-Access for Mirror Sites in the Internet. *Proc. Of IEEE INFOCOM'00, Mar. 2000.*

[3] A. Miu and E. Shih. Performance Analysis of a Dynamic Parallel Downloading Scheme from Mirror Sites Throughout the Internet. *url:http://nms.lcs.mit.edu/ aklmiu/comet/paraload.html, Dec. 1999.*

[4] A. Zeitoun, H. Jamjoom and M. El-Gendy. Scalable Parallel-Access for Mirrored Servers. *In the Proc. of 20th IASTED, 2002.*

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to Algorithms, Second Edition. *McGraw-Hill 2002.*

[6] Mohamed Hefeeda, Ahsan Habib, Boyan Botev, Dongyan Xu and Bharat Bhargava. PROMISE:Peer-to-Peer Media Streaming Using CollectCast. *In Proc. of ACM Multimedia 2003.*

[7] Robert Carter and Mark Crovella. Server selection using dynamic path characterization in wide-area networks. *IEEE INFOCOM 1997.*

[8] Anees Shaikh, Renu Tewari and Mukesh Agrawal. On the Effectiveness of DNS-based Server Selection. *IEEE INFOCOM 2001.*

[9] R. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. *In Proc.of ACM SIGCOMM, 2001.*

[10] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *In Proc.of ACM SIGCOMM, 2001.*

[11] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: A global-scale overlay for rapid service deployment. *IEEE JSAC, 2004.*

[12] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM ICDSP, 2001.*

[13] Bjorn Knutsson, Honghui Lu, Wei Xu and Bryan Hopkins. Peer-to-Peer Support for Massively Multiplayer Games. *INFOCOM, 2004.*

[14] C Tang, Z Xu, S Dwarkadas. Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. *In Proc.of ACM SIGCOMM, 2003.*

[15] Ellen W. Zegura, Ken Calvert and S. Bhattacharjee. How to Model an Internetwork. *Proceedings of IEEE Infocom '96.*

[16] C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the 9th Annual ACM SPAA, 1997*

Seung Chul Han received the BS degree in Computer Science and Engineering from the Sogang University, Seoul, Korea, in 1995, and the MS degree in Computer Science from Purdue University in 2003. He is currently working toward the Ph.D degree in Computer and Information Science and Engineering at the University of Florida. His primary research interests include computer networks and operating systems.

Ye Xia received the BA degree in 1993 from Harvard University, the MS degree in 1995 from Columbia University and the PhD degree in 2003 from University of California, Berkeley, all in Electrical Engineering. Between June 1994 and August 1996, he was a member of the technical staff at Bell Laboratories, Lucent Technologies in New Jersey, where he worked in performance evaluation of a shared-memory ATM switch and studied traffic control for ATM networks. Since August 2003 to the present, he is an assistant professor in the Computer and Information Science and Engineering department at the University of Florida. His primary research interest is in communication networks.