Efficient Peer-to-Peer Lookup in Multi-hop Wireless Networks

Minho Shin[1] and William A. Arbaugh[2]
[1]Institute for Security, Technology, and Society, Dartmouth College, Hanover, NH, USA
[e-mail: mhshin@cs.dartmouth.edu]
[2]Computer Science Department, University of Maryland, College Park, MD, USA
[e-mail: waa@cs.umd.edu]
*Corresponding author: William A. Abraubgh

Received Jan 06, 2009; revised January 28, 2009; accepted January 29; published February 23, 2009

Abstract

In recent years the popularity of multi-hop wireless networks has been growing. Its flexible topology and abundant routing path enables many types of applications. However, the lack of a centralized controller often makes it difficult to design a reliable service in multi-hop wireless networks. While packet routing has been the center of attention for decades, recent research focuses on data discovery such as file sharing in multi-hop wireless networks. Although there are many peer-to-peer lookup (P2P-lookup) schemes for wired networks, they have inherent limitations for multi-hop wireless networks. First, a wired P2P-lookup builds a search structure on the overlay network and disregards the underlying topology. Second, the performance guarantee often relies on specific topology models such as random graphs, which do not apply to multi-hop wireless networks. Past studies on wireless P2P-lookup either combined existing solutions with known routing algorithms or proposed tree-based routing, which is prone to traffic congestion. In this paper, we present two wireless P2P-lookup schemes that strictly build a topology-dependent structure. We first propose the Ring Interval Graph Search (RIGS) that constructs a DHT only through direct connections between the nodes. We then propose the ValleyWalk, a loosely-structured scheme that requires simple local hints for query routing. Packet-level simulations showed that RIGS can find the target with near-shortest search length and ValleyWalk can find the target with near-shortest search length when there is at least 5% object replication. We also provide an analytic bound on the search length of ValleyWalk.

Keywords: Multi-hop wireless networks, peer-to-peer search, simulations, mathematical analysis

We would like to express our great thanks to the Editor and unknown reviewers' very active and quick review process. We believe that their detailed review has enhanced the quality of our paper.

DOI: 10.3837/tiis.2009.01.001

1. Introduction

Multi-hop wireless network can provide a flexible network service to wireless nodes spread over a certain area. Through a series of direct connections, distant nodes can indirectly communicate when they are beyond their radio coverage. Mesh-like connections between neighboring nodes enables multiple choices in choosing paths to the destination for data packets. Among applications of multi-hop wireless networks are wireless mesh networks, sensor networks, and vehicular ad-hoc networks. However, the lack of a centralized controller in those networks often makes it difficult to design a reliable service. Design of packet routing, for instance, has been a major challenge in multi-hop wireless networks for a couple of decades. Recently, data discovery in multi-hop wireless network has received attention as an application in multi-hop wireless networks [1][2][3][4][5]. In particular, one interesting question is whether it is possible to build a reliable peer-to-peer (P2P) file-sharing system in a wireless domain, likely suggested by the past success of Internet P2P file-sharing [5]. For distributed file sharing, a peer-to-peer lookup service is crucial. Using a P2P-lookup service, each node (or a peer) can request a target object that is stored within some other node (also a peer); each node can act as both a client (requesting data) and a server (serving data), even simultaneously if needed. This paper attempts to take an early step toward the design of a reliable wireless P2P-lookup service.

For the design of a wireless P2P-lookup, it is natural to apply the many existing P2P-lookup solutions originally intended for wired networks. The distributed hash table (DHT) [6][7][8][9][10] is regarded as an efficient and reliable P2P-lookup technology because of its highly structured design. Although DHTs may seem applicable at first, their topology-independent design makes DHTs inappropriate for multi-hop wireless networks. Because of radio interference and limited bandwidth, a multi-hop wireless network is more likely to delay or even fail packet delivery, as data travels more wireless links [11][12][13][14][15][16]. As DHTs disregard the underlying physical topology, they tend to suffer from a long search delay and a low search success rate. This is because a query message that only travels a few links in the DHT structure can travel multiple wireless links in the underlying wireless network. Although many topology-aware DHT schemes exist [17][18][19][20][21][22], DHTs are not primarily concerned with physical locality in their structure.

Other kinds of existing P2P-lookup are loosely-structured and unstructured approaches. Unlike DHTs, these do not build a global search structure. Instead, they guide the query message to the destination with local hints (loosely-structured) [23][24][25] or with blind attempts (unstructured) [26][27]. While the unstructured approach works poorly for multi-hop wireless networks due to link vulnerability, the loosely-structured approach is more adaptable. However, the analytic claims on unstructured and loosely-structured solutions [23][24][26][27] often make topological assumptions that are incompatible with multi-hop wireless networks; small world [28], power law [29], Gnutella network, or random graph models are assumed, while a multi-hop wireless network is best characterized by a random geometric graph [30]. For example, the mixing time analysis of a random walk found in [31][32][24] does not apply to random geometric graphs.

Some past work has explored the wireless P2P-lookup problem. Most works, however, proposed flooding-based solutions [33][34][35][36]. However, flooding scales poorly in multi-hop wireless networks. Many non-flooding solutions assume that nodes are aware of

their locations and use geographic routing [1][2]. Other schemes build on tree structures, either by combining an existing DHT with an existing routing algorithm [3][4], or by designing their own structure [5]. Tree-based routing, however, has a tendency to forward query traffic to the links close to the root of the tree, and cause congestion on those links. Aly and Elnahas [37] proposed a hybrid method; their approach is closer to hierarchical approach than peer-to-peer approach (similar to the first generation P2P services, e.g., *Napster*).

In this paper, we focus on the wireless P2P-lookup problem; given a multi-hop wireless network and a set of objects, we assign each object to a node (or multiple nodes if desired) such that any node can find the object with the following properties:

- Decentralization: every node acts as both data requester and data provider,
- Load balance: each node can store an object with similar probability.
- Efficiency: every node can find (a copy of) an object as close as possible,
- Scalability: the scheme scales well with higher query rates.

In this work, instead of small ad-hoc networks with high mobility, we focus on relatively stable but *large scale multi-hop wireless networks* (such as a wireless mesh network in an urban area consisting of more than one hundred nodes). Large-scale is more challenging for P2P lookup services because flooding- or randomwalk-based schemes hardly work and search-optimization is critical because of large network diameter and inherently high query rates. Therefore, we assume the network has little mobility but query load is high.

We propose two wireless P2P-lookup schemes: a DHT scheme *Ring Interval Graph Search* (RIGS) and a loosely-structured scheme *ValleyWalk*. The RIGS builds a topology-dependent DHT structure such that one-hop neighbors in the structure are also one-hop neighbors in the underlying physical topology. Using the same design principle, ValleyWalk uses a simple heuristic to forward the query message to a neighbor node that is closer to the destination. Unlike RIGS, ValleyWalk is loosely-structured, and a simple hint about the destination guides the message to its destination. Both schemes use only local information and a global parameter.

We evaluate our scheme via packet-level simulations (using ns2 [38]) and make a comparison with the optimal solution and existing schemes. The results show that RIGS always guarantees successful search through near-shortest paths. ValleyWalk also finds the target through near-shortest paths when there are a moderate number of object copies in the network (i.e., 5% object replication). We also provide a mathematical analysis for finding upper-bounds on the search length of ValleyWalk. Our bound is significantly tighter than existing analysis given by Morselli et al. [24]. Our schemes, however, focus on static networks where the topology of the network does not dynamically change over time.

In the next section, we discuss existing P2P-lookup schemes that are not discussed in this section. In the following two sections, we describe RIGS (Section 3) and ValleyWalk (Section 4) in more detail. In Section 5 we present simulation results and in Section 6 we analyze the performance of ValleyWalk. We conclude in Section 7.

2. Related Work

In this section, we discuss existing distributed P2P lookup schemes not discussed in the previous section.

Breadth first search (BFS). Breadth first search requires no structure or topology control; the querying node floods the network within a certain hop distance hoping to hit the target [39]. Although flooding is simple and may find the closest copy of the target, the broadcast query messages can overload the network links. We can limit the flooding area, but finding the

optimal flooding radius is not trivial [27]. Iterative deepening [40], Directed BFS, and Local indices are suggested to reduce the message overhead. The BFS method, however, is undesirable for multi-hop wireless networks, because of the limited bandwidth and interference.

Depth first search (DFS). The requesting node sends a single query message to one of its neighbors and each neighbor in turn forwards the query to one of its neighbors, until we hit a copy of the target object. Random-walk is a well-known DFS scheme where each node uniformly chooses a neighbor at random. Although the message overhead is minimal, the response time is high. The *k*-random-walk improves the response time by starting *k* independent random-walks simultaneously, but it can be costly to stop other random-walks when a random-walk finds a target. DFS is also undesirable in multi-hop wireless networks, because of its high response time.

Loosely-structured search. Loosely structured search leaves hints of the target locations throughout the network and uses those hints for searching the object. For example, FreeNet [23] uses history-based hints, Yappers [25] uses partition-based hints, and LMS [24] uses the node-to-object distance in the hash-space. Loosely-structured searching has little or no control over the topology, and thus can work in multi-hop wireless networks without significant modification. Our ValleyWalk uses a similar approach with LMS, but our searching algorithm significantly improved the search performance in multi-hop wireless networks.

Distributed hash table (DHT). The DHT-based scheme adds a significant amount of structure by closely coupling its overlay network topology and the placement of objects. DHTs such as CAN [8], Pastry [9], Chord [7], Tapestry [10], and Kademlia [6] can provide theoretical bounds on the worst-case performance and can guarantee successful search.

3. RIGS: Ring Interval Graph Search

RIGS is fully structured and carefully designed to find the closest target object. RIGS uses a novel search structure *Ring Interval Graph* with *Shortest Interval Searching* algorithm. The structure is built by a distributed algorithm during the initialization phase. Each node is only required to know the local information in order to forward search queries to destinations. RIGS is designed for stable multi-hop wireless networks (i.e., less mobility).

3.1 Hash space

RIGS uses the circular hash space used by Chord [7]. We simplify the original system using the assumption of continuity for the hash space; the hash-space is a continuous real interval [0,1) instead of a discrete set $\{0, 1,..., 2^m\}$. As in Chord, we visualize the hash space as a unit-length circle where 0 and 1 meet at the same point and the values increase in the clockwise direction.

Each object is hashed to the hash space [0,1) and the hashed value is called a *key*. Each node is assigned a *node-id* in [0,1) by the RIGS algorithm. Each key is assigned to the node, viz., *key-holder*, for which the node-ID is equal to or greater than the key value in the hash space. Unlike other consistent-hashing systems that assign node-IDs at random, RIGS carefully chooses the node-ID such that the assignment generates a Ring Interval Graph.

3.2 Ring Interval Graph

In order to assign node-IDs, RIGS builds a distributed data structure called *Ring Interval Graph*. Let us first define *ring interval* and then define Ring Interval Graph. In this paper, v_i denotes both the node itself and the assigned node-ID.

A ***ring interval*** $(a, b]$ is a segment of the hash ring starting at a (excluding a) and ending at b (including b) in the clockwise direction. If a ring interval $(a, b]$ contains zero, the interval is the union of two real intervals $(a,1)$ and $[0,b]$. For example, if $v_8 > v_0 > 0$, then $(v_8, v_0] = \{x \mid v_8 < x < 1 \text{ or } 0 \le x \le v_0\}$. For brevity, we write $x \rightarrow y$ if there is no node-ID assgined in (x, y). A node-set $\{v_i, v_{i+1}, \ldots, v_{i+k}\}$ *induces* the ring interval $(v_{i-1}, v_{i+k}]$ if $v_{i-1} \rightarrow v_i \rightarrow v_{i+1} \rightarrow \ldots \rightarrow v_{i+k}$ where v_{i-1} is the preceding node of v_i. In **Fig. 1**, $\{v_8, v_9, v_{10}, v_{11}, v_0\}$ induces the ring interval $(v_7, v_0]$. Note that one node $\{v_i\}$ induces the ring interval $(v_{i-1}, v_i]$.

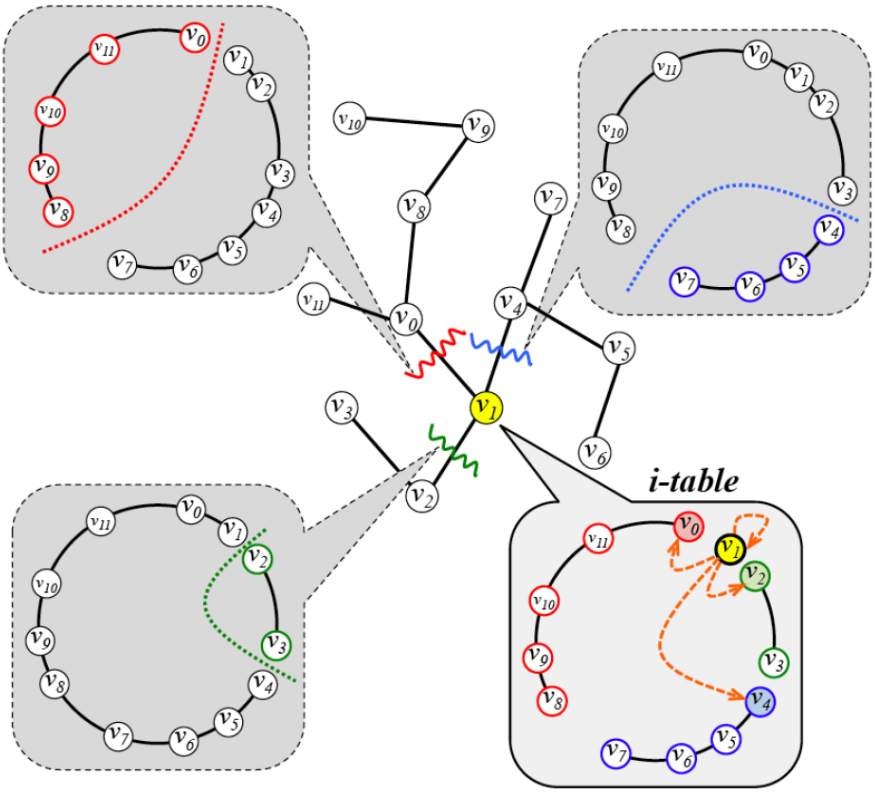


Fig. 1. Ring Interval Graph

Definition 3.1 Given a network graph $G = (V, E)$, a *Ring Interval Graph* (RIG) is an acyclic undirected subgraph $G_{RIG} = (V, E_{RIG})$ where $E_{RIG} \subseteq E$ and each node is assigned a node-ID such that any *one-cut* (removal of one edge) of G_{RIG} partitions the graph into two subgraphs $G_1=(V_1, E_1)$ and $G_2=(V_2, E_2)$, such that V_1 induces R_1, V_2 induces R_2, and R_1 and R_2 partition the hash space (i.e., $R_1 \cap R_2 = \emptyset$ and $R_1 \cup R_2 = (0,1]$).

Fig. 1-(center) shows a Ring Interval Graph of a 12-node graph. Their node-IDs are placed in the hash space at the corners of the figure. Without loss of generality, we assume that the nodes are placed in the order of the node index, in the clockwise direction. In **Fig. 1**-(top left corner), the removal of the edge (v_0, v_1) partitions the ring into two subgraphs with node-set $\{v_8, \ldots, v_{11}, v_0\}$ and $\{v_1, \ldots, v_7\}$. The hash space is partitioned into their induced ring intervals $[v_7, v_0)$ and $[v_0, v_7)$, respectively. Likewise, the figure illustrates the partition of the graph when edge (v_1, v_2) or (v_1, v_4) are removed. As **Fig. 1**-(bottom right coner) concisely illustrates, node v_1 can decide to which neighbor it should forward the query, by checking which interval the

target key belongs to. We build a routing table based on the ring interval information.

Interval table. Given the aforementioned partition information, we build a routing structure as follows. Let $Nb(v)$ denote the neighbor set of node v and $Nb^+(v) = Nb(v) \cup \{v\}$ be the expanded neighbor set. Suppose $u \in Nb(v)$. Then, edge (v, u) partitions the hash-space into two ring intervals, one of which includes v and the other which includes u. Let I_v^u be one of the ring intervals that includes node u. For example, in Figure 1, $I_{v_1}^{v_4} = (v_3, v_7], I_{v_1}^{v_0} = (v_7, v_0]$, and $I_{v1}^{v_2} = (v_1, v_3]$. Let us also define $I_v^v = (v_p, v]$, where v_p is the preceding node of v in the hash space. For example, $I_{v_1}^{v_1} = (v_0, v_1]$. Note that if $k \in I_{v_1}^{v_1}$, then node v_1 has the key and the search terminates. Then, the *interval-table* or *i*-table of node v is ***i*-table(*v*)** $= \{I_v^u \mid u \in Nb^+(v)\}$.

For example, in **Fig. 1**, ***i*-table**$(v_1) = \{I_{v_1}^{v_0}, I_{v_1}^{v_1}, I_{v_1}^{v_2}, I_{v_1}^{v_4}\} = \{(v_7, v_0], (v_0, v_1], (v_1, v_3], (v_3, v_7]\}$. In practice, each node builds a list of <*val*, *node*> sorted according to *val*, where *val* is the ending value of each interval and *node* is the neighbor node associated with that interval. In **Fig. 1**, *i*-table(v_1) = (<v_0, v_0>, <v_1, v_1>, <v_3, v_2>, <v_7, v_4>).

Basic routing. Given the *i*-table, each node can forward queries to a neighbor who is closer to the destniation on the Ring Interval Graph. Suppose we are given a query for key k. The node first finds an interval that contains the key. For example, node v_1 finds that $k \in (v_3, v_7] = I_{v_1}^{v_4}$. Then, the node forwards the query to the neighbor node associated with that interval. In our example, v_1 forwards to node v_4.

Basic routing, however, limits the traffic on the Ring Interval Graph, which is a spanning subgraph of the original graph. Subgraph routing, however, causes traffic congestion along the subgraph edges, while edges outside the subgraph remain idle. Subgraph routing also fails to find the shortest path and often causes a detour. In **Fig. 1**, the basic routing algorithm will deliver the query of $k \in (v_3, v_7]$ through the path (v_1, v_4, v_5, v_6). However, it is likely that node v_1 and v_6 are one-hop neighbors in the original graph, and thus the query can be delivered in one-hop path (v_1, v_6). In the following section, we describe how RIGS can use the Ring Interval Graph to overcome the drawbacks of basic routing, and achieve near-shortest path routing.

3.3 Shortest Interval Search

Although basic routing with *i*-table can route queries to destinations, the message path is limited to the spanning subgraph, which wastes edges outside the subgraph. *Shortest interval search* enables RIGS to use all the edges in the network and, more importantly, achieve near-shortest routing.

In brief, each node collects *i*-tables of neighbor nodes and for routing, the node selects the shortest ring interval containing the key in those *i*-tables, and forwards the query to the neighbor associated with the interval. Formally, let us define i^*-table of node v as the set of all intervals collected from neighbors' *i*-tables, which is expressed as:

$$i^*\text{-}table(v) = \left\{ I_u^w \mid u \in Nb(v), w \in Nb^+(u) \setminus \{v\} \right\}$$

Given a key k, node v forwards the query to node u^* such that:

$$u^* = \arg\min_{u \in Nb(v)} \left\{ | I_u^w | : I_u^w \in i^*\text{–}table,\ k \in I_u^w \right\}.$$

Fig. 2 illustrates an example of Shortest Interval Search. Suppose we are given a network of 12 nodes $\{v_0, v_1, \ldots, v_{11}\}$. The node-ID assignment is shown in the graph and their values are places in the hash space. In the graph, the solid edges represent the Ring Interval Graph, while the dotted lines are the rest of the edges in the network. Suppose node v_6 has collected the i-tables of neighbor nodes v_1, v_4, and v_5. When v_6 receives a query for k such that $k \rightarrow v_3$, node v_6 checks all the intervals in its i^*-*table* and compares the length of the intervals that contain k: $I^{v_1} = (v_1, v_3], I^{v_4} = (v_7, v_3]$, and $I^{v_5} = (v_6, v_4]$. Since I^{v_1} has the shortest interval size (see **Fig. 2**-(bottom left corner)), node v_6 forwards the query message to node v_1. Using the same algorithm, node v_1 forwards the query to v_3, which is the destination.

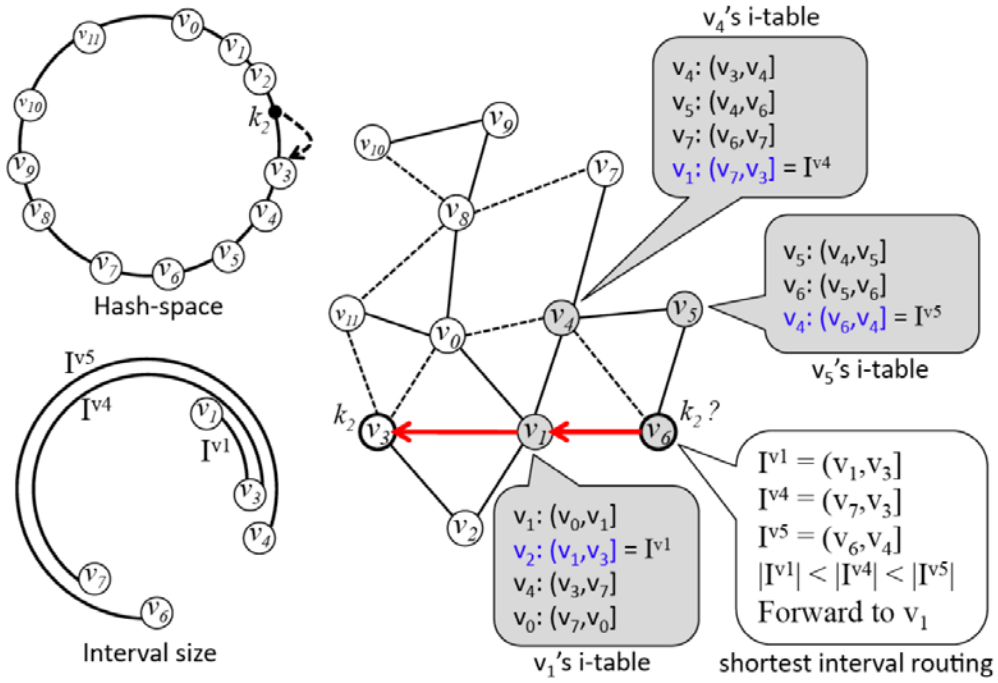


Fig. 2. Shortest Interval Search

The following describes the Shortest Interval Search algorithm, which is executed when node v does not have the key.

1. min := 1
2. **for all** $u \in Nb(v)$ **do**
3. **for all** $I_u^w \in i^*\text{-}table(v)$ **do**
4. **if** $k \in I_u^w$ **and** $| I_u^w |$ < min **do**
5. next-node := u
6. min := $| I_u^w |$
7. **end if**

8. **end for**
9. **end for**
10. *v* sends the query to next-node

Before we explain the rationale behind the algorithm, we first claim that following a shorter ring interval leads to a shorter path on the Ring Interval Graph than longer intervals.

Theorem 3.1 Given a key k and two nodes v and v', let $I_v^u \in i$-table(v) and $I_{v'}^{u'} \in i$-table(v') such that $k \in I_v^u$ and $k \in I_{v'}^{u'}$. Then,

If $I_{v'}^{u'} \subset I_v^u$ and $I_{v'}^{u'} \neq I_v^u$, *then distance*(v', *holder*(k)) < distance(v, *holder*(k))

where *distance*(·) is the hop count between two nodes on the Ring Interval Graph, and *holder*(k) is the node that possesses key k.

*Proof of Theorem 3.1***.** Suppose $v \in I_v^u$ (equivalently, $v = u$). Then $I_{v'}^{u'} = I_v^u$, a contradiction. Therefore, $v \notin I_v^u$ $v \neq u$. Then, the edge (v, u) partitions the network into two node-sets; the nodes whose node-IDs are in I_v^u and the nodes whose node-IDs are in $R \setminus I_v^u$ where R = [0, 1). Any node from one partition cannot go to the other partition without passing the edge (v, u).

Suppose $v' \in R \setminus I_v^u$. Then, every path from v' to the key-holder of k, which is in the other partition, should pass the node v, i.e., $v \in$ *path*(v', *holder*(k)). Since $k \in I_{v'}^{u'}$, the key-holder of k belongs to $I_{v'}^{u'}$, and the path from v' to *holder(k)* is contained in $I_{v'}^{u'}$. Therefore, $v \in I_{v'}^{u'}$. Since $I_{v'}^{u'} \subset I_v^u$, we get $v \in I_v^u$, a contradiction. Therefore, $v' \in I_v^u$.

Here we claim that v' is on the path from v to k, i.e., $v' \in$ *path*(v, *holder*(k)). Suppose that this is not the case; v' is not in *path*(v, *holder*(k)). Let w ($\neq v'$) be a common node of the paths from v to *holder*(k) and from v' to *holder*(k), i.e., $w \in$ *path*(v, *holder*(k)) $\cap$ *path*(v', *holder*(k)). Since $w \in I_{v'}^{u'}$, and v' is not in the path from v to w, v cannot be in $R \setminus I_{v'}^{u'}$, thus $v \in I_{v'}^{u'}$. Since $I_{v'}^{u'} \subset I_v^u$, we get $v \in I_v^u$, a contradiction. Therefore, v' is on the path from v to *holder*(k).

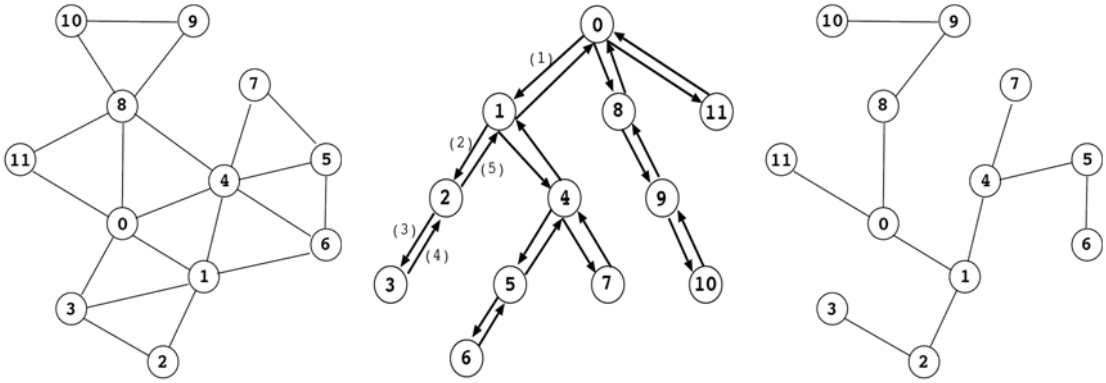
Based on Theorem 3.1, the Shortest Interval Search algorithm always chooses a neighbor that has the shortest ring interval containing the key. Although this choice only guarantees progression towards the destination on the Ring Interval Graph, the use of all available intervals in the vicinity finds short-cut paths beyond the Ring Interval Graph and avoids detours. Experimental results show that such a heuristic can achieve near-shortest paths in various settings.

3.4 Construction

Here we explain how we can construct a Ring Interval Graph of the given network topology. The construction requires a spanning tree of the original graph. We can construct a spanning tree via a depth-first or breadth-first graph traversal. We can also use one of the available distributed spanning-tree algorithms [41][42][43].

Given a spanning tree, we assign each node a node-ID in an increasing order along the depth-first traversal on the spanning tree. We assume that each node v knows the area of the

hash space to which it was assigned, denoted by L_v, and $\sum_v L_v = 1$. For ease of explanation, we assume that the hash space is expanded to $[0,n)$ where n is the number of nodes, and nodes are assigned a uniform unit length interval, that is, $L_v = 1$. Since the assignment requires a depth-first traversal of the graph, we can build a depth-first spanning tree simultaneously if desired. However, in the algorithm description, we assume that a spanning tree is already given.



(a) Original network graph (b) Assignment by DFS (c) Constructed RIG

Fig. 3. Construction of Ring Interval Graph

Fig. 3 shows an example construction of Ring Interval Graph. Suppose we are given a network graph, as shown in **Fig. 3**-(a), where the node-IDs have not yet been assigned. The construction algorithm starts with a randomly-chosen node and assigns zero as its node-ID, then performs a depth-first traversal, as shown in **Fig. 3**-(b). In the figure, the number in parenthesis denotes the order of visit. At the first visit of each node, the algorithm assigns a node-ID, which increases by one at each assignment. The resulting tree becomes a Min-Heap tree structure; all node-IDs of any subtree are greater than the node-ID of the root of the subtree. This property enables the partitioning property of Ring Interval Graph described in Definition 3.1. After assignment, only the edges of the depth-first traversal remain for the Ring Interval Graph. **Fig. 3**-(c) shows the Ring Interval Graph that corresponds to the DFS. The following construction algorithm is executed by v when it is the root node or when it receives message x from node p

1. **if** v is a root node **then**
2. $y := 0$
3. **else**
4. $y := x + L_v$
5. *i-table.add*(<x, p>)
6. **end if**
7. *node-id*(v) := y
8. *i-table.add*(<y, v>)
9. **for all** $u \in Nb(v) - \{p\}$ and u remains unvisited **do**
10. v **sends** u : "y"

11. **wait** *v* **receives from** *u* : "z"
12. *i-table.add*(<*z, u*>)
13. *y* := *z*
14. **end for**
15. **if** *v* is a not root node **then**
16. *v* **sends** *p* : "*z*"
17. **end if**

When node *v* receives *x*, which is the last assigned node-ID, it assigns itself a node-ID by increasing *x* by its assigned area L_v. After adding *i*-table entries for *p* and *v*, it sends the last assigned value *y* to each neighbor and waits until the neighbor finishes assignment of all its descendent nodes. The returning value *z* is the last assigned value in the subtree of *u*. The node *v* repeats this for unvisited neighbors. After assigning all nodes in its subtree, *v* sends *z* to the parent node *p*. The construction ends when the root node receives reply messages from all unvisited child nodes, and it finishes the algorithm without executing line 11.

4. ValleyWalk

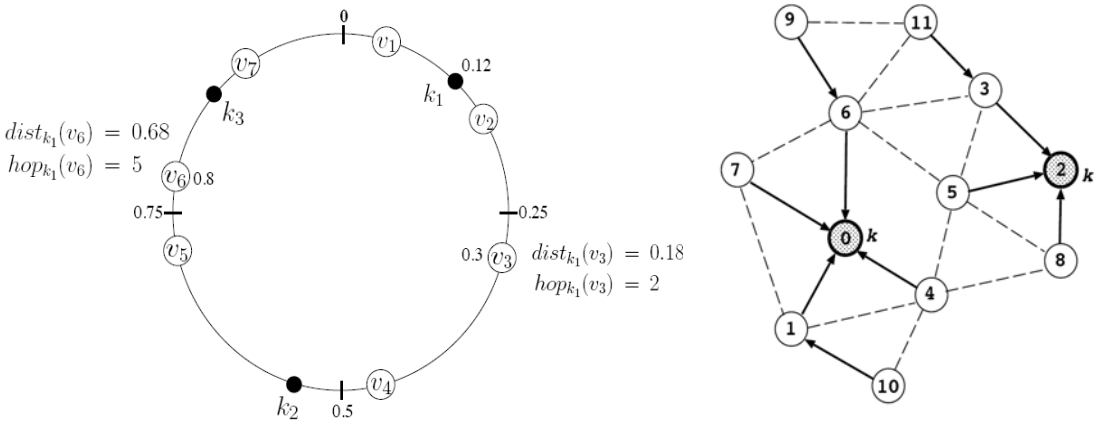
ValleyWalk uses a simple forwarding algorithm with a lightweight local structure; "*each node forwards the query to the neighbor node that is the closest to the target object in the hash-space"*. Because of its minimal structure, ValleyWalk assumes that a moderate number of object copies are available in the network. After describing the algorithm, we discuss the difference between ValleyWalk and LMS [24].

4.1 Hash Space and Definitions

ValleyWalk uses the same hash-space as Chord, and we use the same notation used in Section 3. Unlike RIGS, we hash both objects and nodes to the hash-space, obtaining keys and node-IDs, respectively. Let us define metrics to represent the relationship between nodes and a key. Given a key *k*, we define *key-hop-count* of node *v* with respect to *k*, denoted by $hop_k(v)$, as the number of nodes in the ring interval (*k*, *v*]. We also define *key-distance* of node *v* with respect to *k*, denoted by $dist_k(v)$, as the length of the ring interval (*k*, *v*], that is, the distance from *k* to *v* in the clockwise direction. **Fig. 4**-(a) illustrates the definitions of $hop_k(v)$ and $dist_k(v)$. We state that node *v* is closer to *k* than node *u* if $dist_k(v) < dist_k(u)$. As in Chord, key *k* is stored in the closest node *v*, that is, $hop_k(v) = 1$ or, equivalently, $dist_k(v) < dist_k(u)$[1] for all nodes *u*. In **Fig. 4**-(a), $holder(k_1) = v_2$, $holder(k_2) = v_5$, and $holder(k_3) = v_7$.

Given a key *k*, a node *v* is a *local minimum* with respect to *k* if *v* is the closest to the key among its neighbors, that is, $dist_k(v) < dist_k(u)$ for all $u \in Nb(v)$. **Fig. 4**-(b) shows a network with two local minima, node 0 and node 2, and arrows indicating the closest nodes to the key among neighbor nodes. In LMS, keys are stored in local minima and each node forwards the query to the neighbor who is the closest to the key among neighbors. In the figure, queries are routed in the direction of the arrows. When we reach a local minimum but no key is found, a random walk is performed to restart another local-minima search. In the following, we explain ValleyWalk, which modifies LMS to achieve better performance.

[1] We assume that no two nodes have the same node-ID.



(a) Key distance (b) Local Minima

Fig. 4. Hashing for ValleyWalk and Local Minima

4.2 ValleyWalk

ValleyWalk performs a simple deterministic walk to search or store keys in the nodes. Each node that receives a query for key k forwards the query to one of its unvisited neighbor nodes that is closest to k. Formally, node v forwards the query for k to an unvisited node u^* if

$$u^* = \arg\min_{u \in Nb(v)} dist_k(u) .$$

In order to avoid loops, each node stores recent queries in its cache or records visited nodes within the query. When a node finds that the query has already visited all its neighbor nodes, it forwards the query to a randomly chosen neighbor. **Fig. 5** illustrates the ValleyWalk. On the left are the node-IDs placed on the hash space. Suppose node v_8 receives a query for key k_1. According to the values of node-IDs, v_0, v_1, and v_4 are local minima but only v_0 holds k_1. Choosing the closest neighbor to k_1 at each step, the query message reaches the destination through path (v_8, v_1, v_3, v_0). Note that unlike LMS, ValleyWalk continues with its deterministic walk when it reaches an empty (i.e., no key) local minimum.

As an analogy, we can compare ValleyWalk to a hiking strategy in a mountain where the hiker always chooses the direction towards the lowest area around the current position. The hiking path can be either downhill or uphill (if the hiker has reached the bottom of a basin), and, as a result, the hiker is likely to travel along the valleys between peaks.

Node v executes the following algorithm when it receives the query message for key k.

```
1.  if v = holder(k) then
2.     exit
3.  end if
4.  mindist := 1
5.  next := ⊥
6.  for all u ∈ Nb(v) s.t. u remains unvisited do
7.     if dist_k(u) < mindist then
8.        next-node := u
9.        mindist := dist_k(u)
10.    end if
11. end for
```

12. **if** next-node = $\perp$ **then**
13. pick a random $u \in Nb(v)$
14. **end if**
15. send the query to next-node

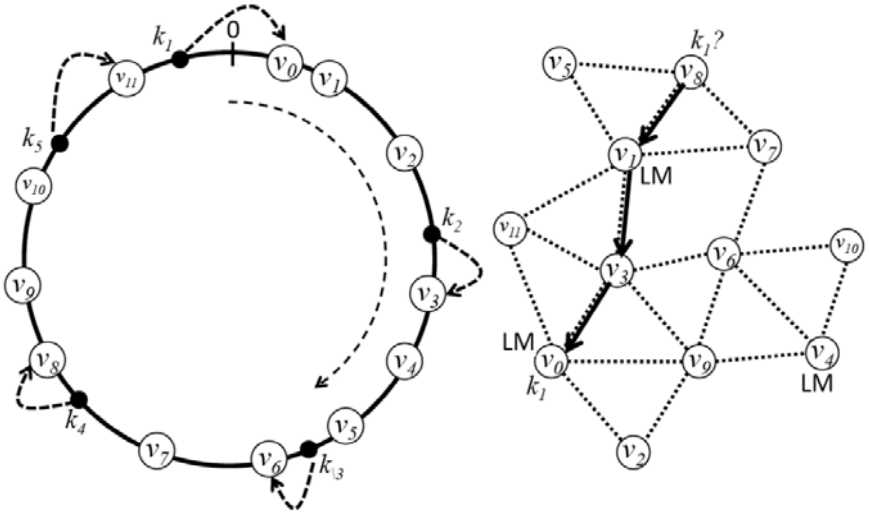


Fig. 5. ValleyWalk

4.3 Key Assignment

Suppose we want to store r copies of each key in the network. Since ValleyWalk always forwards the query to a node with a smaller key distance, ValleyWalk is inclined to move towards local minima. Therefore, it is most efficient to store keys in local minima (as in LMS). However, the number of local minima for a given key depends on the topology and node-IDs. Let LM_k be the set of local minima and $|LM_k|$ be the number of local minima in the network. Since the probability is $1/(d_v+1)$ that node v is a local minimum, where d_v is the degree of node v, the expected number of local minima is:

$$E[|\, LM_k \,|] = \sum_{v \in V} \frac{1}{d_v + 1}$$

where V is the total node-set. Since the number of local minima may not equal the replication number r, we store r copies of each key as follows.

We start with a ValleyWalk and whenever we reach a local minimum we store a copy of the key in that node. If we can find r local minima, we stop. If we can no longer find a local minimum (after a certain number of trials) and we still have copies of the key to store, then we store the remaining copies in randomly chosen nodes with a series of random-walks. As a result, it is possible to have both a local minimum node without a key and a key holder that is not a local minimum.

The discrepancy between the local minima set and the key-holder set can cause overhead of the search algorithm. In particular, a false positive, which involves hitting an empty local minimum, is more harmful than a false negative, which involves hitting a non-local-minimum and unexpectedly finding the key. In order to minimize the false positives, we can intentionally decrease the number of local minima such that $|LM_k| < r$. Since

$E[|LM_k|] \le \frac{|V|}{\delta+1}$ where δ is the minimum degree in the network, we can artificially decrease the expected number of local minima by increasing the minimum node degree. Given a target minimum degree δ^*, nodes that have fewer degrees than δ^* can expand their neighbor sets by adding virtual links with (δ^* - d_v) 2-hop neighbor nodes. Adding a virtual link means that the node creates imaginary edges to the 2-hop neighbors and asks them to send their node-IDs. The expansion of the neighbor set creates an asymmetric relationship between neighbors; node v considers node u as a neighbor, while node u does not consider v as a neighbor. This asymmetry does not affect the correctness of the ValleyWalk. We do not expand the neighbor-set beyond 2-hop neighbors, to avoid creating a local overlay-network and increasing the search length.

4.4 ValleyWalk and LMS

LMS uses both the deterministic walk and random walk. The deterministic walk of LMS forwards the query towards the neighbor with the smallest key distance among neighbors and the current node (whereas ValleyWalk does not compare with the current node). Consequently, LMS stops its search at any local minimum node (whereas ValleyWalk continues the search). When LMS reaches an empty local minimum, LMS performs a random-walk, followed by another deterministic walk, and it repeats this until it hits a target. In order to avoid continuous arrival at the same local minimum (like a black hole), LMS doubles the length of the random-walk each time. The characteristics of a random walk differ between a wired network and a multi-hop wireless network. For instance, the analysis of the mixing time of the random-walk presented in [24] assumes a random graph, which is not the case in multi-hop wireless networks. Our simulation results show that by removing this random-walk from LMS and by replacing the deterministic walk with ValleyWalk, we can achieve a significant performance improvement in wireless P2P-lookup.

4.5 Analysis of ValleyWalk

In this section, we provide an analytic upper-bound of the search length of ValleyWalk. For simplicity, we assume that $|LM_k| \le r$, that is, every local minimum has a key. When $|LM_k| > r$, the additional search length when hitting empty local minima should be considered, and our analysis does not cover that case.

We model ValleyWalk as a time series – a series of independent random variables on [0, 1). Given a key k, let (v_0, v_1, v_2, …) be a search path and (X_0, X_1, X_2, …) be random variables for the key distance of the nodes in the path, that is, X_i=$dist_k(v_i)$. By definition, X_{i+1} is the smallest key distance among v_i's neighbors, and the search stops at v_i if $X_i < X_{i+1}$, otherwise it continues. Therefore, if the search stops at v_l, then $X_0 > X_1 > \ldots > X_{l-1} > X_l < X_{l+1}$. Since the first node is the distance between two random values, X_0 follows a uniform distribution on [0, 1). Suppose we have followed a path (v_0, v_1, v_2, …, v_l) so far, and none of them was a local minimum. Let *revealed set* R_l denote the nodes either on the path or in the neighbor set of the path. That is, $R_l = \{v \mid v \in \bigcup_{i=0}^{l} Nb^+(v_i)\}$. Since $dist_k(v_0) > dist_k(v_1) > \ldots > dist_k(v_{l-1}) > dist_k(v_l)$ and $dist_k(v_i) < dist_k(v_j)$ for all $v_j \in Nb(v_{i-1})$, the node v_l has the smallest key distance among all other nodes in R_{l-1}, that is, $dist_k(v_l) < dist_k(v_j)$ for all $v_j \in R_{l-1}$. Since we want to find the next node $v_{l+1} \in Nb(v_l)$ such that $dist_k(v_{l+1}) < dist_k(v_l)$, we only need to check the nodes in $Nb(v_l) \backslash R_{l-1}$. We call this set *candidate set* at step l+1, denoted by C_{l+1}. Therefore, X_{i+1} is the closest key distance of the nodes in C_{i+1}. Since all candidate sets are disjoint, X_i's are independent. **Fig. 6**-(a) shows the topology, and each node is labeled according to its key-distance. The search follows the path

(v_0, v_1, v_2). The path selection is explained by **Fig. 6**-(b); at each step, we chose the smallest key distance (gray circles) among the candidate set (white or gray circles). The empty circles are revealed nodes in previous steps, so they are excluded in the candidate set. In the figure, R_2={9,6,11,0,3,5,7} and $Nb(v_2)$={1,4,5,6,7}, so $C_3=Nb(v_2)\backslash R_2$={1,4}. Similarly, C_0={9}, C_1={6,11}, and C_2={0,3,5,7}. Therefore, by choosing nodes with the smallest key-distance in candidate sets, we chose v_0 ($dist_k(v_0)$=9), v_1 ($dist_k(v_1)$=6), v_2 ($dist_k(v_2)$=0), and v_3 ($dist_k(v_3)$=1). Since $dist_k(v_3)>dist_k(v_2)$, we stop at v_2.

Consider the following thought experiment. Let Δ be the largest node degree in the network. Then, the candidate set cannot be larger than Δ. Imagine the case (viz., Δ-case) when each candidate set is as large as Δ ($|C_i| = \Delta$). Since a node is less likely to have the smallest key distance among a larger set of nodes then among smaller sets, the expected length of the search increases as the candidate-set expands. Therefore, the expected search length of the original case is no longer than the Δ-case, thus the Δ-case provides an upper-bound of the search length.

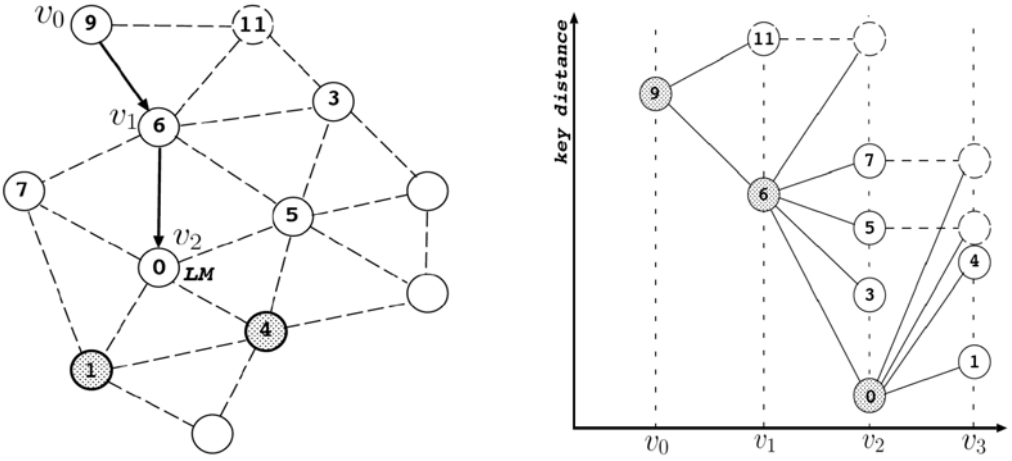


Fig. 6. Analysis of ValleyWalk

Theorem 4.1 (Tail-bound of ValleyWalk) Given k>0, the *tail-bound of search length L* of ValleyWalk is:

$$\Pr(L \geq k) \leq \frac{1}{k!}\sum_{i=0}^{k}\binom{k}{i}\frac{(-1)^i}{\Delta i+1}$$
$$\leq \frac{1}{k!}$$

Proof of Theorem 4.1. Let L_Δ be a random variable for the search length in Δ-case and Y_0, Y_1, Y_2, … be the random variables for key distances along the search path in Δ-case. Let $\hat{f}(y_0, y_1, ..., y_k)$ be the joint *pdf* of Y_0, Y_1, …, Y_k. Then,

$$\begin{aligned}\Pr(L \geq k) &\leq \Pr(L_\Delta \geq k)\\ &= \Pr(Y_0 \geq Y_1 \geq \ldots \geq Y_k)\\ &= \int_0^1\int_0^{y_0}\int_0^{y_1}\cdots\int_0^{y_{k-1}} \hat{f}(y_0, y_1, y_2, \ldots, y_k)dy_k \cdots dy_2 dy_1 dy_0.\end{aligned}$$

Since Y_i's are independent, $\hat{f}(y_0, y_1, \ldots, y_k) = f_0(y_0) f_1(y_1) \cdots f_k(y_k)$ where $f_i(y_i)$ is the *pdf* (probability density function) of Y_i. Because Y_0 follows the uniform distribution on [0, 1), $f_0(y_0) = 1$. Since the key-distance of each node also follows the uniform distribution on [0,1) and Y_i ($i > 0$) is the smallest key-distance among Δ nodes, *pdf* of Y_i is $f_i(y_i) = f(y_i) = \Delta(1-y_i)^{\Delta-1}$ for $i>0$ and the *cdf* (cumulative distribution function) of Y_i is $F_i(y_i) = F(y_i) = 1-(1-y_i)^{\Delta}$.

Since $\int_0^{y_{k-1}} f(y_k)dy_k = F(y_{k-1})$ and $\int_0^{y_{i-1}} f(y_i)\frac{F(y_i)^j}{j!}dy_i = \frac{F(y_{i-1})^{j+1}}{(j+1)!}$, it follows that:

$$\begin{aligned}
\Pr(L \geq k) &\leq \int_0^1 f_0(y_0)\int_0^{y_0} f(y_1)\int_0^{y_1} f(y_3)\cdots\int_0^{y_{k-2}} f(y_{k-1})\int_0^{y_{k-1}} f(y_k)dy_k dy_{k-1}\cdots dy_2 dy_1 dy_0 \\
&= \int_0^1 f_0(y_0)\int_0^{y_0} f(y_1)\int_0^{y_1} f(y_3)\cdots\int_0^{y_{k-2}} f(y_{k-1})F(y_{k-1})dy_{k-1}\cdots dy_2 dy_1 dy_0 \\
&= \int_0^1 f_0(y_0)\int_0^{y_0} f(y_1)\int_0^{y_1} f(y_3)\cdots\int_0^{y_{k-3}} f(y_{k-2})\frac{(F(y_{k-2}))^2}{2!}dy_{k-2}\cdots dy_2 dy_1 dy_0 \\
&\vdots \\
&= \int_0^1 f_0(y_0)\frac{(F(y_0))^k}{k!}dy_0 \\
&= \int_0^1 \frac{(1-(1-y_0)^{\Delta})^k}{k!}dy_0
\end{aligned}$$

If we integrate the Binomial expansion $(1-(1-y_0)^{\Delta})^k = \sum_{i=0}^{k}\binom{k}{i}(-1)^i(1-y_0)^{\Delta i}$, it follows that:

$$\Pr(L \geq k) \leq \frac{1}{k!}\sum_{i=0}^{k}\binom{k}{i}\frac{(-1)^i}{\Delta i+1}$$

We can simplify the formula by using the finite difference technique. Let $Df(x) = f(x+1) - f(x)$. Then, by equating $D^k f(x)$ and $D^k(1/x)$ and replacing x with $1/\Delta$, it follows that:

$$\begin{aligned}
\Pr(L \geq k) &\leq \frac{1}{\left(\frac{1}{\Delta}+1\right)\left(\frac{1}{\Delta}+2\right)\cdots\left(\frac{1}{\Delta}+k\right)} \\
&\leq \frac{1}{k!}
\end{aligned}$$

Morselli et. al [24] showed a tail bound of $\frac{\Delta}{2^{k-1}}$ for the same problem, and our bound is significantly tighter. **Fig. 7** compares our analysis with Morselli's analysis and packet-level simulation results. The simulation parameters are the same as those in Section 5. The topology had an average degree of 6.5 and a maximum degree of 11 (thus $\Delta = 11$). The variance of the node degree caused the gap between the simulation and analytic bound, and the gap becomes negligible when the variance of the degree becomes small. The average search length of the simulations was 1.49, and the average of our analysis was 1.54.

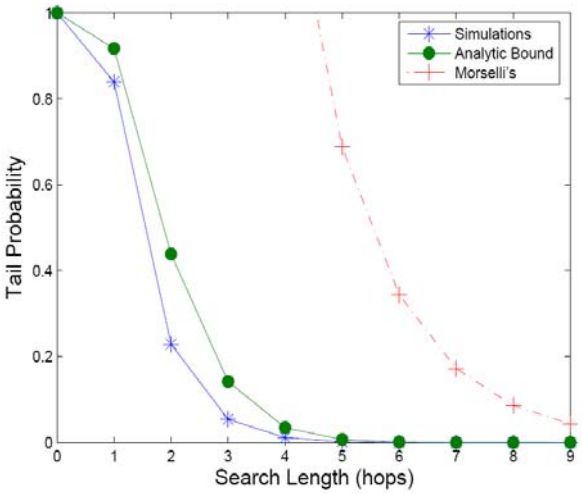


Fig. 7. Analysis Results

5. Performance Evaluation

By packet-level simulations, we evaluate the performance of RIGS and VALLEYWALK. We compare our algorithms with the optimal solution and other existing solutions.

5.1 Methodology

We used ns2 [38] for our simulations. We uniformly placed 100 nodes at random in a 1,000 by 1,000 m^2 rectangular area (denoted by density 1.0 in the figures). The wireless communication range was set to 250m. After an initialization period for gathering neighbor information (30 seconds), each node generates a series of queries with random keys, following a Poisson distribution of a fixed rate. We varied the total query rate from 20 to 100 queries per second. We also varied the replication number from 1 to 30 nodes. We repeated the simulations with 20 different random seeds and averaged the results.

5.2 Algorithm

We compared the following algorithms.

1. **RIGS**: Ring Interval Graph Search proposed in Section 3. We used BFS spanning tree for the simulations but the result using the DFS spanning tree was similar.
2. **VALLEYWALK**: Proposed scheme in Section 4.
3. **LMS**: Local Minima Searching proposed for P2P overlay networks [24]. Rather than following the recommendation in their paper, we used a random walk of length two, instead of three, because it worked better in our simulations. We added loop detection.
4. **CHORD**: As described in [7]
5. **OPTIMAL**: Each node chooses the closest key copy from the current position and forwards the query along the shortest path.

Because only ValleyWalk and LMS provide a native replication scheme, we introduced a simple replication method for RIGS, CHORD, and OPTIMAL. Given a replication number r and key k, we created equidistant r virtual keys $V = \left\{ k + \frac{i}{r} \mid i = 0,1,\ldots,r-1 \right\}$. Each node

forwards the query to a neighbor who is the closest to any of the virtual keys. i.e., the next node u^* has the minimum smallest distance to any virtual key among neighbors.

5.3 Metrics

We evaluate the performances of the lookup algorithms with the following metrics. Let us define three different lengths. Suppose node v tries to find key k for which r copies are stored in nodes $\{u_1, u_2, \ldots, u_r\}$ and u_1 is the closest from v. Suppose an algorithm finds a key at u_i through path p. *Search length* is the length of the path p; *shortest search length* is the shortest path from v to u_i; *optimal search length* is the shortest search path from v to u_1. Then:

- **Search-overhead**= $\frac{search\ length}{optimal\ search\ length}$, performance compared to optimal solution
- **Locality-overhead**= $\frac{shortest\ search\ length}{optimal\ search\ length}$, closeness of the found key
- **Detour-overhead** = $\frac{search\ length}{shortest\ search\ length}$, overhead due to detour

Tail bound of searching length: the probability that the search length is larger than K hops, that is, $Pr(search\ length \leq K)$.

5.4 Results

We first varied the *replication factor* from 1% to 30%, and compared the algorithms in terms of the overhead. Then, we compared the scalability as the query rate increases.

Search-overhead. **Fig. 8**-(a) compares the search-overhead of the algorithms. The search-overhead of OPTIMAL is, by definition, always one. The performance of RIGS is almost OPTIMAL. ValleyWalk has about five times the overhead of OPTIMAL when there is only one key in the network, but the overhead rapidly approaches one as replication increases. LMS has a similar pattern, but it has about 18 times the OPTIMAL hop count when there is only one replication, and about two times OPTIMAL at best. CHORD, although fully-structured, is eight times less efficient than OPTIMAL. Replication did not help CHORD compensate for its inherent design defect in multi-hop wireless networks.

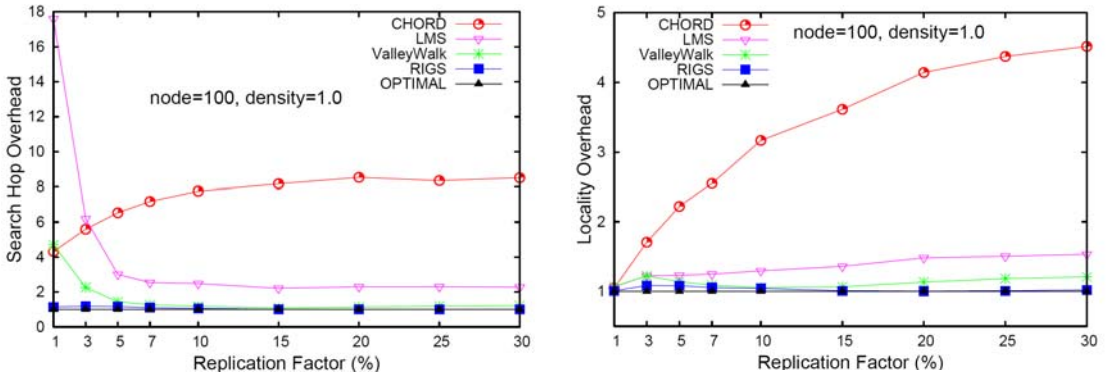
Locality-overhead. **Fig. 8**-(b) compares the locality overhead. Both RIGS and ValleyWalk found very close key-holders and LMS maintained a distance of at most one and half times the OPTIMAL. CHORD ignored the locality and replication did not help.

Detour-overhead. **Fig. 8**-(c) shows how much the search path digresses from the shortest path to the found key. RIGS took the near shortest path and ValleyWalk approached the shortest path when replication was 5% or greater. LMS took about 17 times longer than the shortest path with 1% replication, and reduced its detour overhead down to 1.5 times the optimal solution when replication increased. CHORD took the longest detour to destinations for most replication factors.

Tail bound. **Fig. 8**-(d) shows the tail bound of the search length. For OPTIMAL and RIGS, most queries (95%) took about three hops, and for ValleyWalk they took five hops. However, for LMS, most queries (95%) took eight hops, while CHORD took more than 20 hops.

Scalability. In order to compare the scalability, we varied the total query rate, the sum of the per-node query rates, from 20 to 100 queries per second. As the query rate increases, wireless links become overloaded and packets are dropped. We measured the scalability by the success ratio of a network protocol between the querying node and a key-holder. The protocol starts when the querying node finds a key-holder and the protocol requires five round-trip

messages to finish. A protocol instance succeeds when the querying node finds a key-holder and all the subsequent messages are delivered. The protocol success rate is the ratio of the number of successful protocol trials to the total number of queries generated. **Fig. 9** shows that RIGS and ValleyWalk scale well, even with high query rates. We varied the network size and network density but observed similar results; those figures are not presented due to the space limit.



(a) Search overhead (b) Locality overhead

(c) Detour overhead (d) Tail bound

Fig. 8. Comparison of Searching Performance

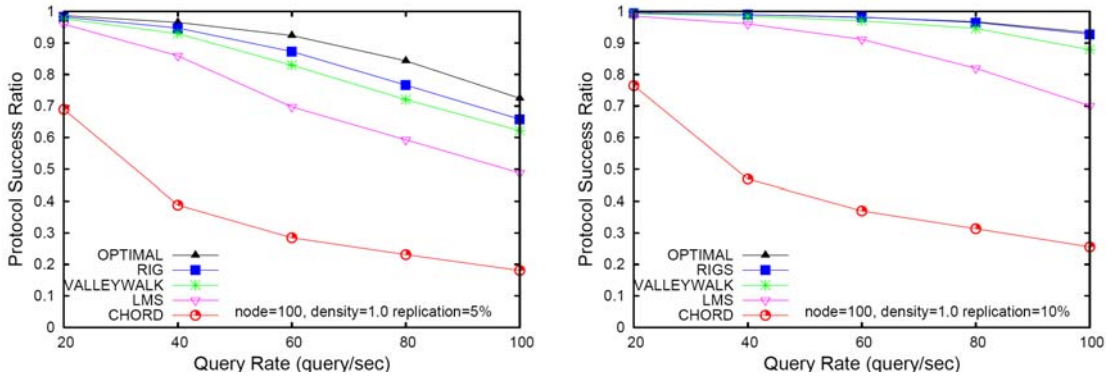


Fig. 9. Comparison of Scalability

6. Conclusion

In this paper, we presented a novel approach to the peer-to-peer lookup problem in large-scale and stable multi-hop wireless networks, and proposed a fully-structured topology-dependent DHT scheme Ring Interval Graph Search (RIGS) and a loosely-structured scheme ValleyWalk. Simulation results show that RIGS achieves near-optimal search performance, even if there is only one object copy in the network. ValleyWalk can also achieve a near-optimal search performance when replication is 5% or greater. The comparison shows that our schemes are significantly more efficient than the existing P2P-lookup schemes.

References

[1] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, S. Shenker, "GHT: A Geographic Hash Table for Data-Centric Storage," In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, 2002.

[2] Filipe Araujo and Luus Rodrigues and Jorg Kaiser and Changling Liu and Carlos Mitidieri, "CHR: A Distributed Hash Table for Wireless Ad Hoc Networks," In *Proceedings of the Fourth International Workshop on Distributed Event-Based Systems* (*DEBS*) (*ICDCSW'05*), Washington, DC, USA, 2005.

[3] Himabindu Pucha and Saumitra M. Das and Y. Charlie Hu, "Ekta: An Efficient DHT Substrate for Distributed Applications in Mobile Ad Hoc Networks," In *Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications*, 2004.

[4] Zahn, Thomas and Schiller, Jochen, "MADPastry: A DHT Substrate for Practicably Sized MANETs," In *Proc. of 5th Workshop on Applications and Services in Wireless Networks* (*ASWN2005*), 2005.

[5] H. Sozer, M. Tekkalmaz, and I. Korpeoglu, "A peer-to-peer file sharing system for wireless ad-hoc networks," In *The Third Annual Mediterranean Ad Hoc Networking Workshop* (*Med-Hoc-Net*), 2004.

[6] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," In *Proceedings of IPTPS02*, Cambridge, USA, Mar. 2002.

[7] R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," In *SIGCOMM*, San Diego, CA, Sept. 2001.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network. In *SIGCOMM*, vol. 31, pp. 161-172, Oct. 2001.

[9] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," In *IFIP/ACM International Conference on Distributed Systems Platforms* (*Middleware*), pp. 329-350, Nov. 2001.

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

[11] R. Bruno, M. Conti, , and E. Gregori, "Mesh networks: Commodity multihop ad hoc networks," *IEEE Wireless Communications*, vol. 43, Mar. 2005.

[12] N. Chang and M. Liu, "Revisiting the ttl-based controlled flooding search: optimality and randomization," In *Mobi-Com*, pp. 85-99, New York, NY, USA, 2004.

[13] Z. Cheng and W. B. Heinzelman, "Flooding strategy for target discovery in wireless networks," *Wireless Networks*, 2005.

[14] A. Gamal, J. Mammen, B. Prabhakar, and D. Shah, "Throughput-delay trade-off in wireless networks," In *Proceedings of IEEE INFOCOM*, 2004.

[15] J. Jun and M. Sichitiu, "The nominal capacity of wireless mesh networks," *IEEE Wireless Communications*, Oct. 2003.

[16] J. Li, C. Blake, D. S. D. Couto, H. I. Lee, and R. Morris, "Capacity of ad hoc wireless networks," In *MobiCom,* pp. 61-69, New York, NY, USA, 2001.

[17] M. Castro, P. Druschel, Y. Hu, and A. Rowstron, "Exploiting network proximity in distributed hash tables," *FuDiCo*, 2002.
[18] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," In *SOSP*, pp. 202-215, New York, NY, USA, 2001.
[19] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of dht routing geometry on resilience and proximity," In *SIGCOMM*, pp. 381-394, New York, NY, USA, 2003.
[20] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao, "Distributed object location in a dynamic network," In *SPAA*, pp. 41-52, New York, NY, USA, 2002.
[21] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," In *SPAA*, pp. 311-320, New York, NY, USA, 1997.
[22] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," In *INFOCOM*, 2002.
[23] I. Clarke, S. G. Miller, T.W. Hong, O. Sandberg, and B.Wiley, "Protecting free expression online with freenet," *IEEE Internet Computing*, vol. 6, no. 1, pp. 40-49, 2002.
[24] R. Morselli, B. Bhattacharjee, A. Srinivasan, and M. A. Marsh, "Efficient lookup on unstructured topologies," In *PODC*, pp. 77-86, New York, NY, USA, 2005.
[25] P. Ganesan, Q. Sun, and H. Garcia-Molina, "Yappers: A peer-to-peer lookup service over arbitrary topology," In *INFOCOM*, San Francisco, California, USA, 2003.
[26] G. H. L. Fletcher, H. A. Sheth, and K. Brner, "Unstructured peer-to-peer networks: Topological properties and search performance," *Lecture Notes in Computer Science*, vol. 3601, pp. 14-27, 2005.
[27] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," In *Proceedings of the 16th annual ACM International Conference on supercomputing*, 2002.
[28] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440-442, June 1998.
[29] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman, "Search in power-law networks," *Physical Review*, E, vol. 64, no. 4, 2001.
[30] M. Penrose, "Random Geometric Graphs," *Oxford Studies in Probability*, Oxford University Press, USA, July 2003.
[31] C. Avin and G. Ercal, "On the cover time and mixing time of random geometric graphs," *Theoretical Computer Science*, vol. 380 no. 1-2, pp. 2-22, 2007.
[32] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Mixing times for random walks on geometric random graphs," *SIAM ANALCO*, Vancouver, 2005.
[33] A. Klemm, C. Lindemann, and O. P. Waldhorst, "A special-purpose peer-to-peer file sharing system for mobile ad hoc networks," in *Vehicular Technology Conference* (*VTC*), 2003.
[34] A. Duran and C.-C. Shen, "Mobile ad hoc p2p file sharing," In *Wireless Communications and Networking Conference 2004*, *IEEE WCNC*, vol. 1, pp. 114-119, 2004.
[35] Z. J. Haas, J. Y. Halpern, and L. Li, "Gossip-based ad hoc routing," *IEEE/ACM Transactions on Networking*, vol. 14, no. 3, pp. 479-491, 2006.
[36] C. Lindemann and O. P. Waldhorst, "A distributed search service for peer-to-peer file sharing in mobile applications," In *Proceedings of the Second International Conference on Peer-to-Peer Computing*, pp. 73, Washington, DC, USA, 2002.
[37] S. Aly and A. Elnahas, "Sustained service lookup in areas of sudden dense population," *Wireless Communication and Mobile Computing,* vol. 8, no. 1, pp. 61-74, Sep. 2006.
[38] ns2. http://www.isi.edu/nsnam/ns.
[39] Gnutella. http://www.gnutella.com.
[40] B. Yang and H. Garcia-Molina, "Efficient search in peer-to-peer networks," In *PODC*, pp. 77–86, New York, NY, USA, 2005.
[41] N. Li, J. Hou, and L. Sha, "Design and analysis of an MST based topology control algorithm," In *IEEE INFOCOM*, 2003.
[42] V. Rodoplu and T. H. Meng, "Minimum energy mobile wireless networks," *IEEE JSAC*, vol. 17,

no. 8, pp. 1333-1344, Aug. 1999.
[1] R. Wattenhofer, L. Li, P. Bahl, and Y.-M. Wang, "Distributed topology control for power efficient operation in multi-hop wireless ad hoc networks," In *IEEE INFOCOM*, Apr. 2001.

Minho Shin is a Postdoctoral Research Fellow of Institute of Security, Technology, and Society (ISTS) at Dartmouth College. He earned his M.S. and Ph. D in Computer Science from the University of Maryland, College Park, USA in 2003 and 2007, respectively. He earned his B.S. in Computer Science and Statistics from the Seoul National University, Seoul, Korea in 1998. His research interests are in wireless networks, wireless network security, and user privacy in people-centric sensing. He has filed several patents in U.S., Korea, and India, and he has refereed articles for many journals and conferences.

William A. Arbaugh is an assistant professor in the Department of Computer Science at the University of Maryland, College Park. His research interests include information systems security and privacy with a focus on wireless networking, embedded systems, and configuration management. He received a BS from the United States Military Academy at West Point, an MS in computer science from Columbia University, New York, and a PhD in computer science from the University of Pennsylvania, Philadelphia. He is on the editorial boards of the IEEE Computer and the IEEE Security and Privacy magazines.