ABSTRACT

Title of dissertation: Fast Scalable Peer-to-Peer Lookup Services for Multi-Hop Wireless Networks

Min-Ho Shin
Doctor of Philosophy, 2007

Dissertation directed by: Professor William A. Arbaugh
Department of Computer Science

Distribution Date (version) : Jan. 15, 2008

Recent years have seen growing popularity of multi-hop wireless networks such as wireless mesh networks and sensor networks. Such systems require efficient lookup services for reliable system operation such as packet routing, key-discovery, and object lookup. The lack of infrastructure, however, makes the centralized lookup fail to scale in multi-hop wireless networks. For example, consider a citywide wireless mesh network which provides wireless connection service to a number of mobile users. Due to a high volume of user access and inherent vulnerability of wireless links, centralized authentication methods fail to scale. The decentralization of user authentication, however, faces a challenge of *key discovery*; how to find the location of user keys. Motivated from the user authentication problem in wireless mesh networks, this dissertation work aims to provide efficient and scalable distributed lookup services for multi-hop wireless networks.

Employing the notion of peer-to-peer lookup where each node can both query and respond, I present two different methods: Valley-Walk and Rigs. A loosely-

structured scheme VALLEY-WALK strategically places object copies and locates them efficiently only with a minimal local structure. The VALLEY-WALK finds target objects in near-optimal hop counts with a moderate number of copies (e.g., 10% the network size) stored in the network. Without a global structure, however, VALLEY-WALK fails to guarantee the low cost search with a small number of copies.

A tightly-structured scheme RIGS (Ring Interval Graph Search) realizes a Distributed Hash Table (DHT) in multi-hop wireless networks. Experimental study shows the limitations of existing DHTs in mult-hop wireless networks due to its independence of underlying topology. Unlike DHT, RIGS constructs a search structure *Ring Interval Graph* such that queries are forwarded only to local neighbors. RIGS guarantees successful object lookup with near-optimal performance.

Fast Scalable Peer-to-Peer Lookup Services for Multi-Hop Wireless Networks

by

Min-Ho Shin

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2007

Advisory Committee:
Professor William A. Arbaugh, Chair/Advisor
Professor Jonathan Katz
Professor David Lovell
Professor David Mount
Professor Neil Spring

Table of Contents

List of Figures

Chapter 1

Introduction

Recent years have seen growing popularity of multi-hop wireless networks such as Wireless Mesh Networks (WMNs) and sensor networks. A *multi-hop wireless network* consists of multiple nodes communicating only with nearby nodes through wireless medium. Any two nodes separated farther than transmission range can communicate with each other only through intermediate nodes. Many systems exploit the multi-hop wireless network especially when the radio transmission range is limited compared to the entire span of the network. The *sensor network*, for example, is a multi-hop wireless network of spatially distributed sensor devices to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion, or pollutants, at different locations. The formulation of a multi-hop wireless network allows for the sensor network to cover a wide area where no infrastructure is available.

For reliable system operation, most multi-hop wireless networks require efficient resource lookup services such as packet routing, key discovery, and object lookup. Among such lookup services, the key discovery problem in wireless mesh networks motivated this dissertation work. Although I demonstrate the contribution of the work only through the key discovery problem in wireless mesh networks, this dissertation work provides an efficient and scalable solution for generic peer-to-peer

lookup problem in multi-hop wireless networks.

1.1 Wireless Mesh Networks

A wireless mesh network is a collection of *mesh nodes* that provides wireless network service to mobile users. Wireless mesh network is gaining popularity for citywide network service because of the flexibility of network deployment and the resilient performance [1, 2, 3, 4, 5, 6, 7]. For example, Houston, Texas, has deployed about 50 Cisco mesh routers throughout the city and additional 150 mesh routers are planned for deployment. In 2005, Tempe, Arizona, has deployed a citywide wireless mesh network spanning 40 square miles for residents, businesses, visitors, as well as municipal workers [2]. The Tempe network later expanded to the neighboring cities Chandler and Gilbert forming the largest contiguous wireless mesh network in the U.S., encompassing 187 square miles [8]. These mesh networks can provide various applications such as mobile data access, voice over IP, intelligent transportation system, and surveillance camera connection over the whole city. There is a growing number of cities, not restricted to US, which deployed or plan to deploy citywide wireless mesh networks.

The advantage of wireless mesh network for citywide networks is attributed to wireless connection between mesh routers (or mesh nodes), which distinguishes the wireless mesh network from the traditional infrastructure-based wireless networks such as WLANs. In wireless mesh networks, mesh nodes are equipped with multiple radio interfaces; one of which is for the connection to mobile users and the rest is for

(a) MetroMesh by Cisco

(b) Strixsys mesh node

Figure 1.1: Wireless mesh network in metropolitan area

the inter-router connection to form a multi-hop wireless backbone. Benefit of wireless backbone is twofold; flexible management and reliable performance. Without tangled wires between mesh nodes, network deployment and maintenance become economic. For example, an outdoor WMN in a campus or a metropolitan area requires minimal wired infrastructure when we can attach wireless routers on trees or light poles. WMN is also resilient against node failures due to redundant routes by intelligent multi-hop routing protocols [9, 10, 11]. Such routing protocols can dynamically reroute packets according to the network condition such as node failure or link congestion. Multiple channel access and separation of radios for upstream and downstream also improve the performance (called 3rd generation of wireless mesh networks). Figure 1.1 shows an example of a wireless mesh network deployed in a metropolitan area.

One of the important characteristics of such a wide-area system is a huge number of users accessing the network while changing their locations. Each connection requires user authentication to assure the authenticity of the access and the privacy

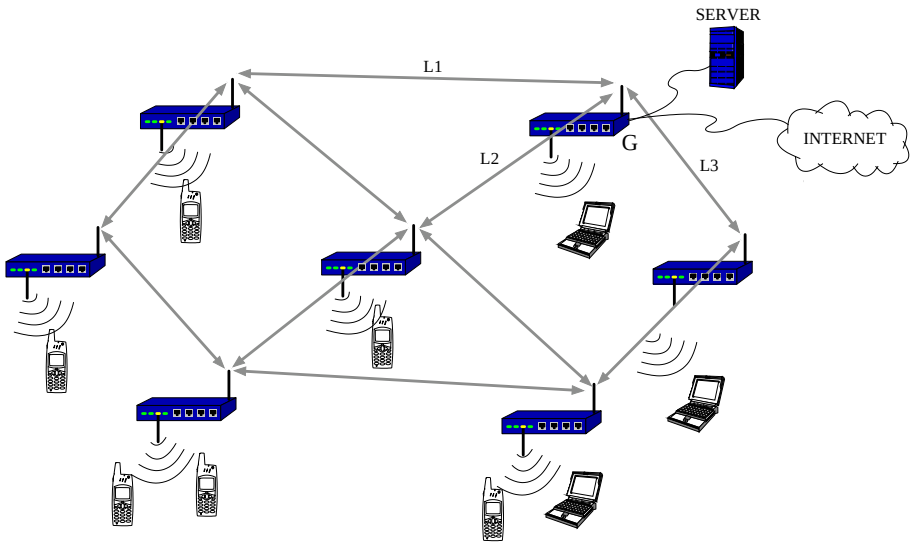


Figure 1.2: A typical wireless mesh network with a central authentication server.

of communication. For reliable services, it is required that the system provides an efficient and resilient authentication service.

Figure 1.2 illustrates a typical solution for the user authentication with a single dedicated authentication server. In the figure, each mesh node has two wireless interfaces, one for mobile users and the other for the wireless backbone. The *gateway* node (G in the figure) connects the network to outside networks, such as the Internet. The central server performs user authentications and other management tasks. The conventional centralized authentication, however, fails to scale in large wireless mesh networks due to inherent vulnerability of wireless links. With a single centralized server, every authentication message should travel to the server through a multi-hop wireless path. The problem of the centralized authentication is three-fold; low performance, unfair authentication delay, and single point of failure. In Figure 1.2, all authentication messages converge to links around the server degrading the performance and reliability of the authentication traffic as well as data traffic. Since

multi-hop wireless networks exhibit notably different throughput for different path lengths, users near the server can experience better authentication performance than farther users. The centralized authentication also suffers from a *single point of failure*; failure, compromise, or DOS attack of the server or nearby links can cause a system-wide failure or security problems. Replication of the central server is costly; it requires not only hardware cost but also data synchronization and load balancing between servers. Replication merely replicates the same problem in multiple locations.

Decentralization of authentication process can overcome the limitations of the centralized approach. Consider the following design of distributed authentication scheme. Suppose we have m secret keys for user authentication and the network consists of n mesh nodes. We want to store r copies of each key among mesh nodes such that whenever a key k is needed, we can efficiently locate a node with a copy of k. To provide such a key-discovery scheme, we have to address the following problems. First, we need a systematic approach on where to store r copies of each key and how to find them in an efficient and reliable manner. A brute force approach is to store every key at every node. However, a compromise of one node can compromise the entire user keys. The scheme should be able to find keys efficiently even if the number of key copies is relatively smaller than the network size. Second, we need the authentication process secure against key compromise. It is clear that compromising a mesh node is easier than compromising a dedicated central server. One way to circumvent this security weakness is to perform multiple authentications in parallel with multiple mesh nodes so that a compromise of one mesh node does

not imply a compromise of the authentication process involving the node. We can easily accomplish the parallel authentication with existing authentication protocols; let the user perform multiple authentication procedures in parallel with multiple mesh nodes using different keys.

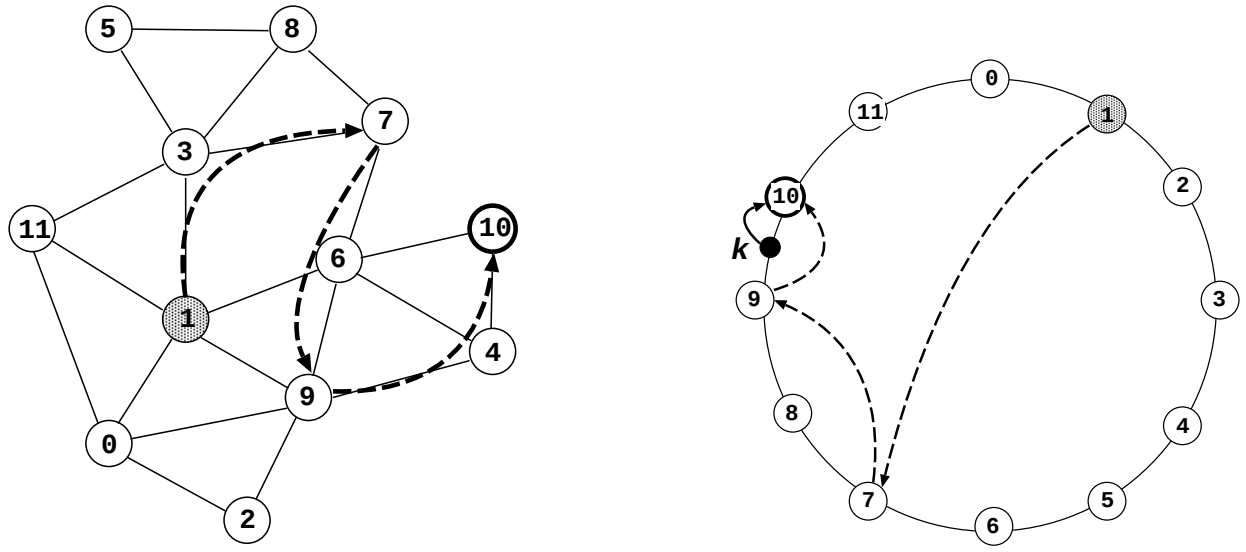
This dissertation work addresses the key discovery problem by addressing a more general problem of peer-to-peer object lookup problem. In the following section, I define peer-to-peer object lookup problem and discuss existing solutions.

1.2 Peer-to-Peer Object Lookup Service

Object lookup service is the fundamental building block of network systems. From the packet routing in the Internet to the distributed file sharing mechanisms in peer-to-peer (P2P) systems, object lookup methods abound in the literature. Especially, the file sharing problem in P2P networks resembles our key discovery problem; place multiple copies of objects (or keys) in the system such that any interested node can efficiently locate a copy of the object. To date, a number of approaches for P2P object sharing have been seen in the literature.

Loosely-structured schemes such as LMS (Local Minima Search) [12] and Yappers [13] achieve better performance with fewer overhead than flooding or random walk based lookup schemes. Such schemes require minimal local structure which hints intermediate nodes to forward toward the destination. Although scalable, these schemes suffer from low success probability in finding the target.

More systematic approach can be found with the *distributed hash table* (DHT)



(a) DHT search on the graph **(b) DHT search on hashing space**

Figure 1.3: DHT in multi-hop wireless network

[14, 15, 16, 17]. DHT maps each object to a node such that any item can be located within a certain number of routing hops using a small per-node routing table. These systems have been used in a variety of distributed applications, including distributed stores [18, 19, 20, 21] and content distribution [22, 23, 24, 25].

DHT, however, has limitations for multi-hop wireless networks. A DHT in a P2P system constructs an overlay network by adding virtual links between nodes regardless of their physical proximity [26]. As a result, a neighbor node in the overlay network can be separated by many hop counts in the underlying network. Since the hop count has a significant impact on delay in multi-hop wireless networks [27, 28, 29, 30, 31, 32], the lack of locality with DHT causes large delay in multi-hop wireless networks. Figure 1.3 shows an example of DHT-based search (e.g., Chord) in a multi-hop wireless network. Figure 1.3-(a) shows the underlying network topology

and Figure 1.3-(b) shows the overlay network where each node is located on a ring-shaped hash space and each node keeps a set of node pointers for its neighbor nodes (not shown in the figure). Suppose node 1 wants to find the key k, which is stored in node 10. In a typical Chord system, the query is forwarded along the path $(1, 7, 9, 10)$ denoted by the dashed arrow lines in Figure 1.3-(b). In this example, the key-holder is found in three hops. In the physical topology, however, the shortest path to the key-holder is just two hops along the path $(1, 6, 10)$ but the query packet actually traveled six hops along the path $(1, 3, 7, 6, 9, 4, 10)$.

Some topology-aware DHT schemes have been proposed [33, 18, 15, 34, 17, 16, 35, 36, 37] but because of different assumptions between wired and wireless networks, their performance improvement in multi-hop wireless networks is not apparent. Topological models of P2P and DHT, such as power-law network [38, 39, 40, 41, 12], are also incompatible with that of wireless networks which is best characterized as a geometric random graph [42].

In this dissertation work, I introduce a notion of *topology-dependent DHT* (TD-DHT) which builds a DHT structure only through local edges. I propose a TD-DHT scheme called *Ring Interval Graph Search* (RIGS), which constructs a novel lookup structure the Ring Interval Graph (RIG) to guide queries toward the destination by narrowing down candidate subgraphs. Nodes can build a RIG in a distributed manner with a minimal global information such as the network size. Simulation results show that the RIGS achieves a near-optimal searching performance.

I also propose a loosely structured scheme VALLEY-WALK which strategically

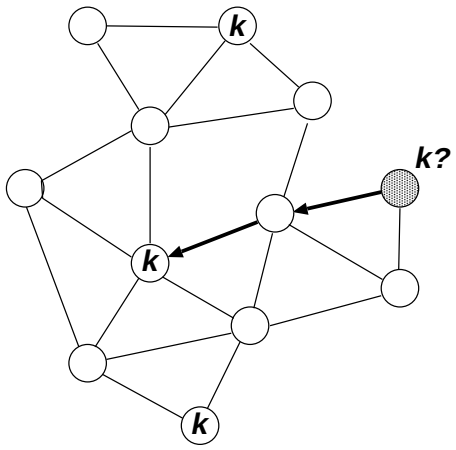


Figure 1.4: Object lookup problem

places object copies and locates them efficiently only with a minimal local structure. The VALLEY-WALK finds target objects in near-optimal hop counts with a moderate number of copies (e.g., 10% the network size) stored in the network. Without a global structure, however, VALLEY-WALK fails to guarantee the low cost search with a small number of copies.

1.3 Problem Definition And Design Goal

Object lookup problem: Suppose we have n nodes and m objects $\{k_1, k_2, \cdots, k_m\}$. Given a parameter $r < n$, we want to distribute r copies of each object in r distinct nodes, called holders, such that, given $i \in \{1, \ldots, m\}$, any node can efficiently locate a holder of the object k_i. Figure 1.4 shows an example of the object lookup problem when three object copies are distributed in the network. Holders of the object are labeled by k in the circle and the grey node wants to find a holder of the object. The arrow lines represent the searching path. We call the number of object copies a

replication number and the ratio of replication number to network size a *replication factor*. The replication number and the replication factor are system parameters which are determined according to system requirements for lookup performance.

I assume *geometric random graph* as network topology. A geometric random graph is a graph probabilistically generated as follows. We place n nodes in the given area uniformly at random. Given a fixed communication range R, any pair of nodes share an edge if and only if their Euclidean distance is less than R. Especially for the mesh network setting, I am also interested in a restricted geometric random graph such that no two nodes are closer than the minimum distance, say $R/2$, and the degree of each node is relatively small so that the network diameter is not too small. This type of topology, which I call a *mesh topology*, is of interest because the wireless mesh network will be carefully deployed with similar properties to minimize inter-node interference and maximize the coverage area.

I assume a uniform query rate for every object. For example, in a wireless mesh network, each user visits the network with the same frequency, or following the same Poisson process with the same inter-arrival time[1]. I also assume that the location of the lookup request is uniform throughout the network such that each node initiates the same amount of queries on average.

The design goals of my key discovery schemes are as follows.

1. Scalable with increasing query rate and network size

2. Efficient lookup by localizing query traffic

[1]With different query rates, we should replicate different number copies for each object. [43] studies optimal replication strategies.

3. Distributed algorithm

First, the scheme should be scalable as the query rate increases. Even if the query rate is fixed for each node, the aggregated query rate in the network grows as the network grows in size and the aggregated query rate impacts on the search performance. Second, the query traffic should be (i) localized and (ii) balanced. The query traffic is localized if it finds the target object in small hops. Lastly, the algorithm should exploit only local information and, if needed, limited global parameters such as the number of nodes or the replication number.

1.4 Thesis Statement

In this dissertation, I support the following thesis: *In multi-hop wireless networks, we can build a scalable and efficient peer-to-peer lookup mechanism, and it can provide a decentralized solution for key discovery problem in wireless mesh networks.* To support this thesis, I design, analyze, and evaluate the following peer-to-peer lookup schemes.

- VALLEY-WALK: a loosely structured peer-to-peer lookup scheme which guides the query to a neighbor which asymptotically approaches to the destination.
- RIGS: a tightly structured lookup scheme which guides the query always toward the destination by forwarding the query to a neighbor which leads to a smaller subgraph containing the target.

1.5 Contribution

This dissertation work contains the following contributions:

- Identify the limitation of existing DHTs with multi-hop wireless networks
- Introduce the notion of topology-dependent DHT (TD-DHT)
- Propose two peer-to-peer lookup schemes for multi-hop wireless networks, both loosely-structured (VALLEY-WALK) and tightly-structured (RIGS).
- Provide an analytical bound on the performance of VALLEY-WALK
- Provide a fast scalable key discovery solution for wireless mesh networks

Chapter 2

Related Work

2.1 Distributed Systems

We can find interesting related works in the literature of distributed systems. The goal of distributed systems is to store huge amount of data in multiple storages and efficiently retrieve the data of interest. Before DHT comes into the picture, most schemes in distributed systems have been focusing on decentralizing index trees.

R-tree [44] is a tree data structure for indexing multidimensional objects, a multidimensional variant of B-tree. R-tree is useful for spatial access methods where one can find an object in a two-dimensional space. The first parallelized version of R-tree (Parallel R-tree) is proposed by Kamel and Faloutsos [45] for indexing with a single CPU but multiple disks. The Master R-tree (M-tree) [46] and the Master Client R-tree (MC-tree) [47] attempts to distribute the R-tree structure among multiple independent servers.

In a Master R-tree, a single server maintains all the internal nodes of the R-tree except the leaf level data nodes which are maintained by other servers. A Master Client R-tree is a two-level distributed R-tree that has a single master index on a master server and local client indexes on the other servers. The Master Client R-tree is similar to the Master R-tree in the sense that it declusters leaf level nodes across data servers. However each data server creates its own local index using the

leaf level nodes that are assigned to it. Therefore, the master index does not have to keep the pointers to the data objects in its master index. Instead, it contains the server address where its local index must be searched again in order to get pointers to the data objects. The query processing of the M-tree or the MC-tree is centralized because it requires a dedicated server which maintains global status of the distributed index. Such a central point becomes a bottleneck with high query rates.

The P2PR-tree [48], designed for spatial indexing, decentralizes the R-tree for better scalability than two-level MC-trees. P2PR-tree is hierarchical and performs pruning of the search space by maintaining more information concerning nearby peers than far ones. However, the P2PR-tree is not fully decentralized, because it requires a large number of dedicated servers that maintain part of a static partitioning of the index. Thus, the P2PR-tree is similar to a replicated version of the Master-Client R-tree [49, 47], and has several other problems related to its static partitioning strategy.

The P-tree [50] fully decentralizes the traditional B^+-tree and allows for one dimensional range queries. The P-tree assumes that a peer stores only a single data object, thus in order to store more than one data object each peer needs to be mapped to multiple *virtual peers*. The routing algorithm in a P-tree is based on virtual peers, thus a peer may be accessed multiple times while routing to virtual peers. For this reason, the P-tree is not suitable for a system that stores many data objects.

For fully distributed indexing, distributed systems began to employ distributed

hash tables (DHTs) developed for peer-to-peer networks. I discuss DHT schemes in later sections.

2.2 Data Search In Peer-to-Peer Networks

Data lookup is important in P2P networks especially for file sharing. Once data is inserted in the network, it is replicated in various nodes such that any node requesting the data can find the data with high probability. There are several approaches for data search in P2P networks.

Centralized Search

Napster [51], one of the most famous P2P systems in its early stage, has central directory servers to maintain the list of connected nodes and the files they provide. Since each server maintains independent information, users cannot get global information from one server. With a single server having the complete list of data and their locations, the system does not scale well. Morpheus [52] combines centralized and decentralized approaches with a hierarchical structure. "Super-peer" nodes act as centralized resource for a small number of clients, but these super-peers then connect to each other to form a pure P2P network. Searching in Morpheus is still centralized; Super-peers maintain the list of peers and their shared files and answers the queries from peers looking for files.

Breadth First Search (Flooding)

With unstructured systems such as Gnutella [53], there is neither a centralized directory nor a control over the topology. The typical search method is flooding such that every node within certain hop distance gets the query message. Although flooding finds the data with the shortest distance, the query message can overload the network and finding optimal TTL (time to live) value is not trivial [39]. Iterative deepening [54] (or Expanding ring [39]), Directed BFS, and Local indices improve flooding search. Iterative deepening broadcasts the query packet with increasing TTL until it finds the data. Directed BFS (Breadth-First Search) forwards the query only to a subset of neighbors to reduce the overhead of flooding. Local indices let each node keep a list of nearby nodes, say less than r hops, and their data. Search with local indices let nodes only at certain hop distances process the query using their local indices. Flooding based searching methods are not desirable for multi-hop wireless networks because of limited bandwidth and interference.

Depth First Search

Random-walk is a well-known blind search which forwards a query message to a randomly chosen neighbor at each step until it reaches the data. Detection of loop and use of TTL can improve the performance of random-walk. This search mechanism does not generate as much message traffic as BFS-based algorithms since there is only one message being routed in the network. The response time, however, is high because of the low probability of hitting the data. To reduce the response

time, *k-random-walk* starts independent k random-walks simultaneously expecting to reduce the response time by a factor of k. Each "random walker" stops its walk when TTL equals zero or other walkers already found the data. To *check* other walker's success, each walker should (occasionally) check back with the original requester before walking to the next node. Despite its simplicity and low overhead, random-walk is not appropriate for key discovery problem because of low success probability.

Loosely-Structured Search

In a loosely structured method, the system adds *hints* in placements of the data and searching algorithm uses those hints at each step. FreeNet [40] uses a simple inserting and routing algorithm called steepest-ascent hill-climbing search. FreeNet associates each data being inserted with a globally unique identifier (GUID) using SHA-1 hash function. The searching algorithm in FreeNet is heuristic and there is no analytical bound. FreeNet starts from a uniform random state and evolves into a non-uniform clustered state. Each node builds a routing table using caching, from an initial list of neighbors to the list of data keys stored in known nodes. The routing table grows when either a query to a data succeeds or an inserting data succeeds. To insert a data with a given TTL, each node forwards to a neighbor who is believed to have a data with closest key to the one being inserted until TTL expires[1]. All intermediate and final node stores the data. To find the data, each node performs the

[1] Although this behavior changed in newer versions of FreeNet, I describe this technique because this provides similar insight to proposed schemes.

same forwarding algorithm as the insertion. When a data is found, all intermediate nodes from the querying node to the destination inserts an entry associating the holder with the requested key. Although the idea of choosing a node holding a key closest to the requested data is similar to the proposed VALLEY-WALK there are many differences between them. First, FreeNet does not use node's unique identity in finding the data holders but VALLEY-WALK and DHT schemes extensively utilize the notion of consistent hashing where the relationship between the node key and the data key, hashed on the same key space, plays a main role in determining the data holders. The lack of this property makes the FreeNet hard to analyze. Also the number of node lists at each node can grow arbitrarily and the simulation shows a power-law distribution of the storage [40]. We cannot employ FreeNet like mechanism in multi-hop wireless networks because the next nodes in forwarding path can be arbitrarily far from the current nodes. The lack of systematic approach of finding the data in FreeNet makes the scheme inappropriate for key discovery problem.

LMS [12] and Yappers [13] achieves better search performance than unstructured methods by adding minimal structure to the network to expedite the search. Yappers (Yet Another Peer-to-PEeR System) partitions the key space of data and nodes into a small number of buckets. It combines structured (local DHT) and unstructured (global flooding). Each node has to keep immediate neighborhood (say 2 hops) and extended neighborhood (say 5 hops). Authors do not present formal analysis. Its flooding mechanism makes Yappers inappropriate to multi-hop wireless networks. I discuss LMS in detail in a later Section 2.4.

Decentralized And structured (DHT)

These systems adds a significant amount of structure by closely coupling its overlay topology and the placement of data. CAN [15], Pastry [16], Chord [14], Tapestry [17], and Kademlia [55] provide such structured systems using hash-like interfaces, often called *Distributed Hash Table* (DHT). The advantage of structured systems are the theoretical bounds on the worst-case performance and guarantee of successful search. Structured systems, however, is not appropriate for the dynamic node membership with frequent join and leave of the nodes. I discuss DHT schemes in more detail in Section 2.3.

2.3 DHT: Distributed Hash Table

In this section, we discuss the distributed hash table (DHT) as a data discovery solution in P2P networks. The limitation of DHT in multi-hop wireless networks will be discussed.

The distributed hash table (DHT) is a class of distributed systems that provide a lookup service similar to a hash table; (*id*[2], *object*) pairs are stored and any participating node can efficiently retrieve the *object* associated with a given *id*. More precisely, DHT maps the object *id* to the node which holds (or supposed to hold) the object. DHT provides only one operation *lookup* for a given *id*, which yields a location of the object with that *id*. The same operation is used for inserting a new object into the network.

[2]I choose to use *id* instead of *key* to avoid confusion with cryptographic keys.

Responsibility for maintaining the mapping from an *id* to an *object* is distributed among the nodes, in such a way that a change in the set of nodes causes a minimal amount of disruption. This allows for DHT to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures. To this end, DHT schemes use consistent hashing [56]; hash the object and the node into the same ID space (or hash space) and assign a node to a subset of ID space so that any object whose *id* belongs to the subset of ID space is assigned to the node in charge of that subset of ID space, called owner/holder. DHT determines which node becomes an owner of a portion of ID space by their "closeness" to the object *id* values. To find the owner of a given object *id*, each node forwards the query to one of its known nodes whose *id* is "closer"to the object in the ID space.

Different DHT schemes define different notions of "closeness" between a node *id* and an object *id*. In Chord [14], the distance between a node *id* and an object *id* is the numeric distance from the object *id* to the node *id* in a clockwise direction along the ring-like ID space. Pastry [16], however, defines the distance as the number of common prefix bits between the node *id* and the object *id*. For an example of Chord, in Figure 2.1, five nodes $\{A, B, C, D, E\}$ are located at $\{0, 1, 3, 5, 6\}$ on the ID space of length 8 and any object whose key belongs to intervals $\{(6, 0], (0, 1], (1, 3], (3, 5], (5, 6]\}$ are stored in $\{A, B, C, D, E\}$, respectively. More precisely speaking, a DHT provides a mapping from the object id to the owner of the object, which in turn provide the object value itself.

What distinguishes different DHT schemes is the construction of ID space and the routing structure over it. Chord builds ID space on a ring-like linear space and

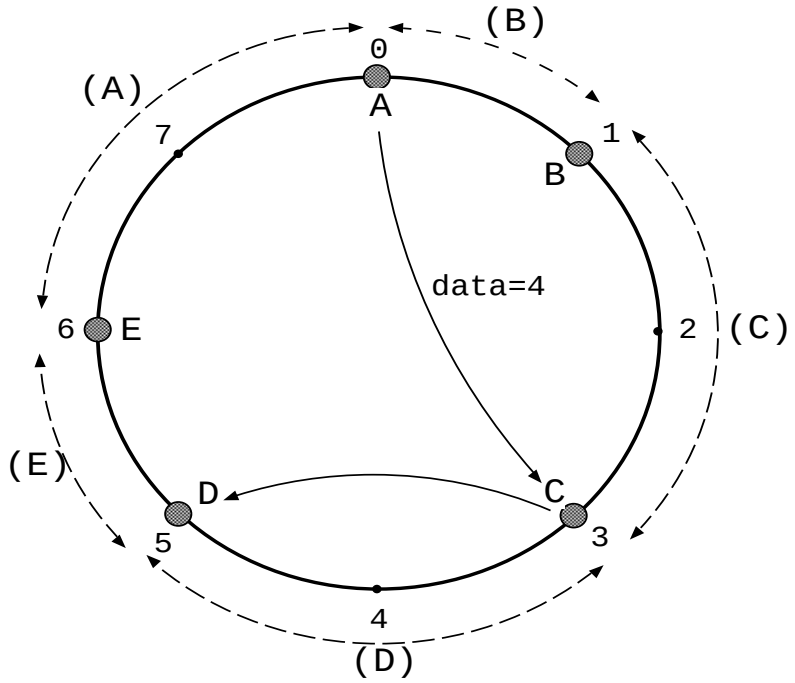


Figure 2.1: The hash space of Chord and its data search.

maintains a routing structure similar to Skiplist [57] where the routing table contains pointers to nodes distant at least halfway around the ID space, a quarter of the way, an eighth of the way, and so forth. Pastry [16], Tapestry [17], and Kademlia [55] use tree-like structures, and CAN [15] uses a multi-dimensional ID space. Note that all DHT schemes build their routing structure to construct an overlay network, a network with virtual links between nodes over an underlying network such as the Internet.

A DHT forms an infrastructure that can be used to build more complex services, such as distributed file systems, peer-to-peer file sharing and content distribution systems, cooperative web caching, multicast, anycast, domain name services, and instant messaging. Applications that use DHT include BitTorrent, eMule, YaCy, and the Coral Content Distribution Network.

2.3.1 Chord

In this section, I describe Chord in detail because the proposed schemes use similar ID space and the notion of "closeness" between nodes and objects.

Chord [14] uses a one-dimensional circular *id* space onto which both nodes and objects are mapped. A node owns all the object *id*s between itself and clockwise preceding node. A chord *id* space can be expressed as $0, 1, 2, \ldots, 2^{m-1}$ for a given parameter $m > 0$. Figure 2.1 shows an example of chord *id* space with $m = 3$ and five nodes $\{A, B, C, D, E\}$ with object *id* segment they own. For example, an object with *id* value 2 belongs to the segment of C and therefore, the object is stored in node C. It is sufficient for each node to keep who is the clockwise successor in order to find an object; each node forwards the query to its successor until the owner node is reached. To expedite search, however, Chord nodes keep a list of m nodes, called finger table such that ith node in node v's finger table is the owner of the object *id* of $id(v) + 2^{i-1}$ where $id(v)$ is node v's *id* and $i = 1, 2, \ldots, m$. For example, node A's finger table has three entries with node pointers B for $i = 1$, C for $i = 2$, D for $i = 3$. When node A wants to find the owner of object 4, it finds the closest node preceding the object 4, which is C in this example. Therefore, A forwards the query to C and C finds that its successor (or the first entry of its finger table) is the owner of the object 4 and forwards to D.

Chord provides the worst case search cost of $O(\log n)$ and guarantees successful search for all queries. To replicate object by r copies, one can store each object at the owner node and $r - 1$ successive nodes from the owner. There are replication or

caching schemes for DHT [16, 18, 58, 59, 60] but they are mainly for load balancing purpose.

DHT schemes, including Chord, can provide a key discovery mechanism by storing pairs of the key *id* and the key-holder's address and search by the key *id* to find a key-holder. The DHT, however, has some drawbacks to be used for multi-hop wireless networks. The important requirements for key discovery problem are to find a key holder in short distance. Most DHTs fail to satisfy this locality requirement for the following reasons. Since existing DHT schemes assume an overlay network, neighboring nodes in an overlay network are not necessarily close to each other. Therefore, a search with a small number of hops in the overlay network can actually have a large number of hops spanning throughout the network. There are some improvements for DHTs to increase the locality of its neighbors [33, 18, 15, 34, 17, 16, 35, 36, 37], but results are limited in multi-hop wireless networks.

2.3.2 Prefix-Tree Based DHT

In this section, I discuss tree-like DHT schemes, Pastry [16], Tapestry [17], and Kademlia [55] to clarify the difference from the proposed RIGS.

Pastry gives each node a randomly chosen ID, indicating its position on an identifier circle. It routes messages with an *id* to the live node with a node ID numerically closest to the *id*, using 128-bit IDs in base 2^b, where b is an algorithm parameter typically set to 4. Imagine a global prefix tree that contains all the nodes in the network. Starting from the root node, representing a null string, i'th

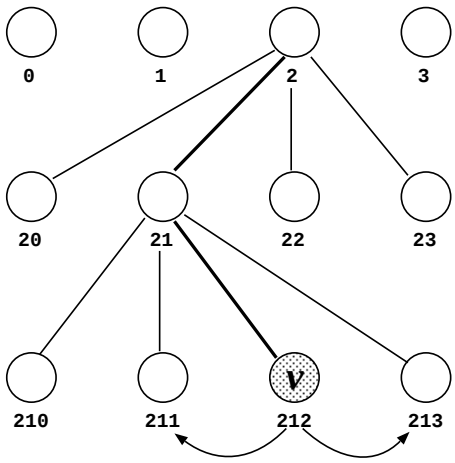


Figure 2.2: The local snapshot of the global prefix tree in Pastry

level has b^i nodes, each of which represents all the nodes with the same prefix of length i. If every node knows this global prefix tree, routing is trivial. Each node, however, stores only a local snapshot of the global prefix tree. Figure 2.2 shows a local snapshot of the prefix tree stored as a form of the routing table of node 213 when $b = 2$. Given the number of nodes N, the tree has $\lceil \log_{2^b} N \rceil$ rows, each with $2^b - 1$ entries. Each entry in row i of the table at node v points to a node whose ID shares the first $i - 1$ digits with node v. Node v also maintains a leaf set L, half of which are leaf nodes closest to and larger than v and the rest of which are leaf nodes closest to and smaller than v. Given the leaf set and the routing table, each node v implements the forwarding step as follows. If the sought *id* is covered by v's leaf set, then the query is forwarded to that node. In general, of course, it will not be, until the query reaches a point close to the target object. In this case, the query is forwarded to a node from the routing table that has a longer shared prefix than v with the sought *id*. Sometimes, the entry for such a node may be missing from

the routing table because the node does not exist, or that node may be unreachable from v. In this case, v forwards the query to a node whose shared prefix with the *id* is at least as long as v's shared prefix, and whose *id* is numerically closer to the object *id* . Such a node must clearly be in v's leaf set unless the query has already arrived at the node with numerically closest ID to the object, or at its immediate neighbor. If the routing tables and leaf sets are correct, the expected number of hops taken by Pastry to route a key to the correct node is at most $\log_{2^b} N$. Tapestry [17] resembles Pastry in terms of the tree structure and routing.

The prefix tree has similar properties with the ring interval graph (RIG) proposed in this dissertation work. However, the prefix trees in Pastry and Kademlia have many differences from RIG. Since prefix trees in those schemes are built as overlay network and each links do not hold locality requirements. However, RIG strictly restricts the routing entries to the physically direct neighbors. Also the construction of prefix trees require the global knowledge but RIG can be constructed through a simple distributed algorithm. Note that we can build RIG by making it a prefix tree.

Kademlia [55] is a peer-to-peer storage and lookup system. Kademlia takes the basic approach of many peer-to-peer systems. Both object *id* and node *id* are from a 160-bit *id* space, and the object is stored on nodes with *id*s close to the object for some notion of closeness. Kademlia uses XOR metric for distance between points in the *id* space. Kademlia treats nodes as leaves in a binary tree, with each nodes position determined by the shortest unique prefix of its *id*. Figure 2.3 shows an example of the binary tree where each leaf node represents a node when the number

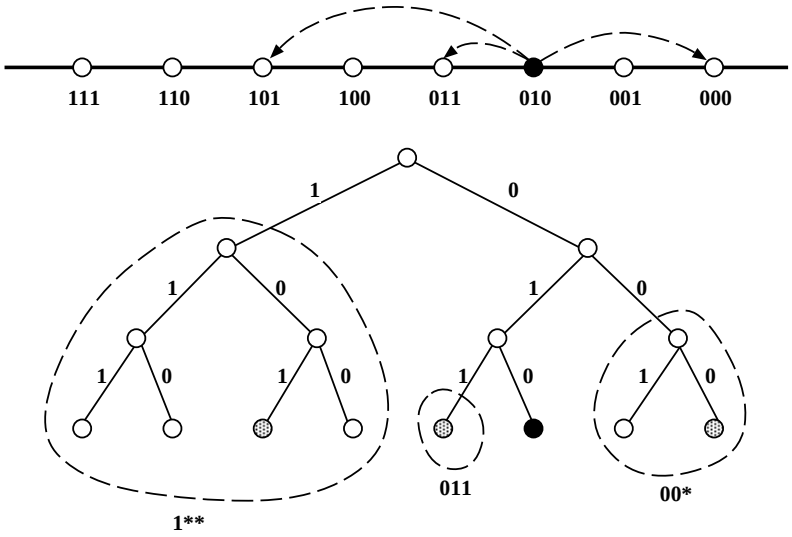


Figure 2.3: The binary tree in Kademlia

of nodes are 8. The line above the tree represents the linear *id* space with nodes *id*s in binary notations. For any given node, say node 010 in the example (black node in the figure), we divide the binary tree into a series of successively lower subtrees that dont contain the node. The highest subtree consists of the half of the binary tree not containing the node. The next subtree consists of the half of the remaining tree not containing the node, and so forth. In Figure 2.3, each subtrees are enclosed by dashed lines labeled with the prefix that represents the subtree. The Kademlia protocol ensures that every node knows of at least one node in each of its subtrees, if that subtree contains a node, shown as grey nodes in the figure. The dashed arrows shown on the *id* space denotes the routing table of node 010.

Every node having this structure, any node can locate any other node by its *id*; Forward the query to the node in the routing table such that the subtree the node belongs to is the smallest one containing the target *id*. Figure 2.4 shows an

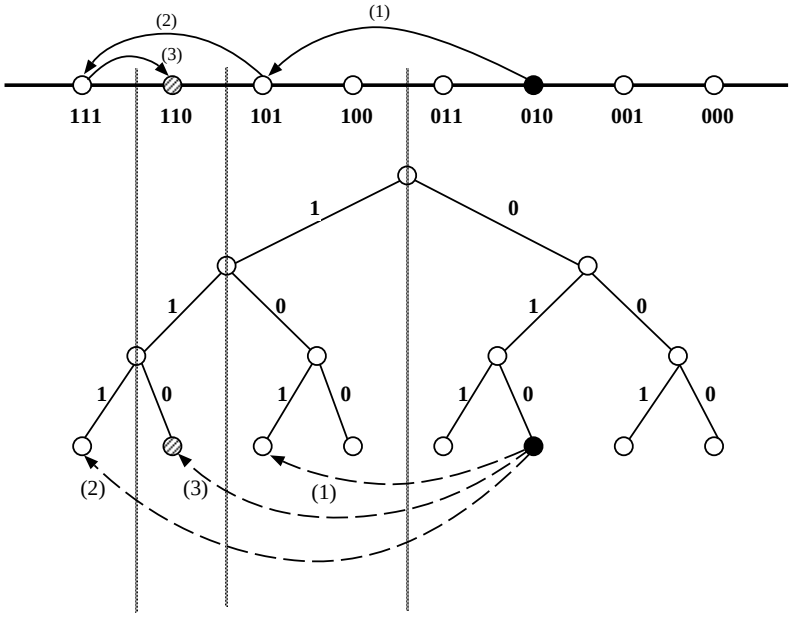


Figure 2.4: Searching in Kademlia

example of node 010 locating node 110 by successively querying the best node it knows of to find contacts in lower and lower subtrees; finally the lookup converges to the target node. When node 010 queries node 101, which is the only node that belongs to a subtree containing the target node, node 101 looks up its own routing table and returns with the contact information of the node, 111, which belongs to the smaller subtree containing the target (message (1) in the figure). In turn, node 010 queries node 111 and finally gets the address of node 110.

2.4 LMS: Local Minima Search

Local Minima Search (LMS) [12] is a loosely structured data lookup protocol in P2P. LMS performs a random-walk followed by a deterministic walk. Data insertion and lookup follows the same procedure. Authors provides an asymptotic bound on

the search cost as $O(n \log n/(grd_h))$ and the size of the state at each node as $O(d_h)$. In the above notations, g is the eigenvalue gap of a random walk over the given graph, r is the number of replications, and d_h is the parameter that defines the range of *neighborhood* used for the deterministic walk. LMS does not guarantee successful search; the probability of successful search depends on the user/node key distribution, topology, the number of replications, and the number of trials.

Since our work borrows the idea of the deterministic walk in LMS, we describe on how LMS performs the deterministic walk in the following. LMS uses consistent hashing; we hash all the key *id*s and all the mesh node *id*s into a unit length interval $[0, 1)$[3]. Imagine a unit length ring such that 0 and 1 meet at the same point. Define the distance between a key k and a node v, denoted by $d(k, v)$, as the shortest distance between k and v along the ring. At each step of the deterministic walk, the node v forwards the request to one of its neighbor v' that minimizes $d(k, v')$ over the d_h hop neighborhood, i.e, all the nodes within d_h hop. A node v is a *local minimum* for the key if v itself has the smallest key distance in the neighborhood. We insert data only in local minima. If a probe message reaches a local minima v without the data, v notifies the requesting node of the failure. The requesting node then starts another random walk with a doubled TTL. Since the number of replications can differ from the number of local minima, LMS can suffer from performance degradation because of empty local minima.

Our algorithm VALLEY-WALK modifies the deterministic walk of LMS such

[3]In the original paper of LMS, authors use unique identifiers of keys and nodes generated uniformly at random from an ID space of $\{0, 1\}^\lambda$ where λ is large enough to avoid collisions. We change this to a unit interval for the ease of representation

that it effectively samples a series of local minima without reporting back and restarting from the originator. While LMS takes alternations of random walk and the deterministic walk, VALLEY-WALK nicely integrates both of them and achieves better performance. VALLEY-WALK can also perform a topology control to minimize the number of empty local minima.

2.5 Limitation Of DHT In Multi-Hop Wireless Networks

DHT schemes in P2P networks have limitation for the object lookup problem in multi-hop wireless networks.

First, the network delay has different characteristics in wired networks and in multi-hop wireless networks. Note that the communication delay between nodes in a wired network does not greatly differ from location to location. With such assumptions, DHT schemes build a topology-independent *overlay network* which assigns virtual links between two hosts without knowing the underlying physical topology. However, because of the interference between wireless links, the distance in the underlying topology has greater impact in a multi-hop wireless network than a wired one. Suppose we construct a P2P overlay network on a multi-hop wireless network. Since the virtual links of a DHT can have arbitrarily large hop counts, I call it *hidden cost of a virtual link*, the route following the virtual link may not be far from the fastest route.

Another limitation of DHTs in multi-hop wireless networks is that a DHT highly depends on the underlying network layer for the packet delivery. In other

words, packet forwarding at each link should rely on the underlying routing scheme. Unlike with wired network such as the Internet, the routing in multi-hop wireless network is costly. For example, when we use a reactive routing scheme such as DSR due to high mobility, at each overlay hop, the routing scheme need to flood the network to locate the next node, or it requires some control packets for routing. Since our scheme only makes an edge between physically local neighbors, the scheme do not suffer from the additional packet routing cost[4].

P2P topologies follow the graph models such as small world network [61], power law network [38], Gnutella graph, and random graph, and most research on P2P searching algorithms assume such topologies [39, 40, 41, 12]. However, multi-hop wireless networks form random geometric graphs [42]. In random geometric graph, each node is located uniformly at random and two nodes are connected only if their Euclidean distance is at most R. Random geometric graph exhibits different characteristics from the topologies that appear in P2P overlay networks. For example, random walk shows a unique mixing time property in random geometric graphs [62, 63, 12].

2.6 Wireless Peer-to-Peer System

Some past work provides peer-to-peer lookup service in wireless networks. Most works, however, focuses on searching algorithms based on optimized flooding. ORION [64] employs a searching algorithm similar to AODV routing scheme

[4]one of proposed schemes may want to forward to 2-hop neighbors, which do not requires serious routing protocol.

through link-layer flooding. Duran and Shen [65] propose a flooding-based searching algorithm with enhancement by filtering and gossiping [66]; flooding of queries are suppressed by local files of intermediate nodes and probabilistic query drops. PDI (Passive Distributed Indexing) [67] actively uses caching of query results to reduce network-wide flooding. Flooding-based searching, however, is not scalable with large network size and highly frequent query rates.

Sozer et al. [68] proposed an interesting file sharing scheme in multi-hop wireless networks. It constructs a global tree structure such that each node is assigned a portion of the hash space through the incremental formation of the multi-hop wireless network. At the beginning of the network, there exists only one node and the entire hash space is assigned to that node. Thus, every object belongs to the node. When a node joins the network as a child of an existing node, the parent node divide its assigned hash space into half and gives one half to the child node. The parent node remembers the hash spaces given to its children so that it can later route queries to an appropriate child node, who can in turn forward the query to its child.

Sozer's scheme holds several drawbacks. Apart from its limitation of only incremental construction, it fails to achieve even assignments of the hash space; the size of hash space assigned to a node depends on the network topology and the order of node joins. The more hash space a node is assigned to, the more data the node should store in its storage, sacrificing its resource. Simulation results show that Sozer's scheme assigns hash space more than three times biased than our algorithm Rigs. Its tree-routing also limits the routing performance compared to

graph-routing, as in RIGS.

Chapter 3

Valley-Walk: A Loosely-Structured Peer-to-Peer Lookup Service

In this dissertation work, I propose a loosely-structured P2P lookup scheme Valley-Walk and a tightly-structured scheme Rigs. I describe Rigs in Section 4. The structure of Valley-Walk is minimal and purely localized. Each local structure requires only local knowledge such as one-hop information. Because of this locality, there is no guarantee that each step advances toward the destination; we only probabilistically bound the required search length.

The Valley-Walk is a simple deterministic walk with occasional random walk when needed. Unlike *random walk* which blindly chooses the next node among neighbors, Valley-Walk gives a strong incentive to one of the neighbors which is believed to be closer to the destination. Valley-Walk tries to compensate the limitation of random walk – blind searching – and that of DHT – independence of topology. Valley-Walk adds a slope to random walk so that the walk has tendency toward closer nodes to the destination to increase possibility of hitting the target. Because of its simple distributed characteristic, Valley-Walk is significantly robust against network dynamics, such as mobility, node join/leave, and failures on nodes or edges.

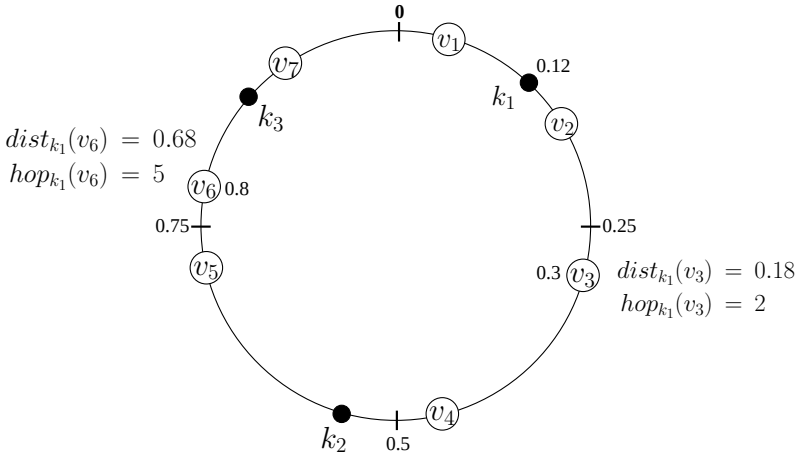


Figure 3.1: Hash space

3.1 Hash Space

VALLEY-WALK follows the same hash space as in CHORD. Although such ring-like hash space will be implemented by a 2^m long discrete space, for the ease of exposition, I describe the hash space as a real value unit length circle. Suppose we have an undirected graph $G = (V, E)$ where V is the set of nodes and E is the set of edges. Let $n = |V|$ be the network size. Assume that each node v is assigned a node identity $id(v)$ chosen uniformly at random from $[0, 1)$, the *id*-space, and each object k is assigned an identity $id(k)$ chosen uniformly at random from $[0, 1)$. It is common in the literature that the identity of the object is called a *key* and the *id*-space is called a *key space*. Although I mean a cryptographic credential by *key* in the context of key-discovery problem, I will also use *key* for the object identity and the actual meaning can be determined by the context.

Imagine the *id*-space as a unit-length ring where 0 and 1 meet at the same point and the value increases in the clockwise direction. We can now locate nodes

and objects on the *id*-space of unit length ring. The Figure 3.1 shows an *id*-space with 7 nodes $v_1, v_2, \ldots, v_7$ and 3 objects k_1, k_2, k_3. The numbers by the nodes and objects are their *id* values. We often use v and k for $id(v)$ and $id(k)$, respectively, when the meaning is clear from the context.

We define two metrics, key-distance and key-hop-count. The *key-distance* of node v for object k, denoted by $dist_k(v)$, is the distance from k to v along the ring in the clockwise direction. The *key-hop-count* of node v for object k, denoted by $hop_k(v)$, is the number of nodes on the ring segment from k to v in the clockwise direction including v itself. For example, let $id(k_1) = 0.12$, $id(v_3) = 0.3$, and $id(v_6) = 0.8$ as in Figure 3.1. Then, $dist_{k_1}(v_3) = 0.18, hop_{k_1}(v_3) = 2$, $dist_{k_1}(v_6) = 0.68$, and $hop_{k_1}(v_6) = 5$. Recall that CHORD assigns an object k to a node v if $hop_k(v) = 1$. VALLEY-WALK however, determines the holders differently as discussed in Section 3.3 and 3.4.

3.2 VALLEY-WALK

The structure of VALLEY-WALK is minimal; each node keeps the *id-table*, a list of neighbor's *id* values. An entry of the *id*-table is $(v, id(v))$ where v is the neighbor node's address and $id(v)$ is its identity. To keep the *id*-table up to date, neighboring nodes periodically exchange their *id* values with each other.

VALLEY-WALK is the minimum key-distance searching algorithm where each node forwards a query for object k to the neighbor node with the smallest key-distance for k, except already visited nodes for the same query. That is, node v

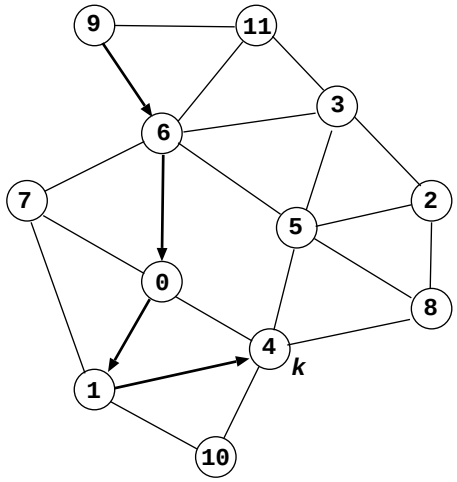


Figure 3.2: VALLEY-WALK

forwards a query for k to a non-visited node u if

$$dist_k(u) = \min_{w \in Nb(v)} dist_k(w)$$

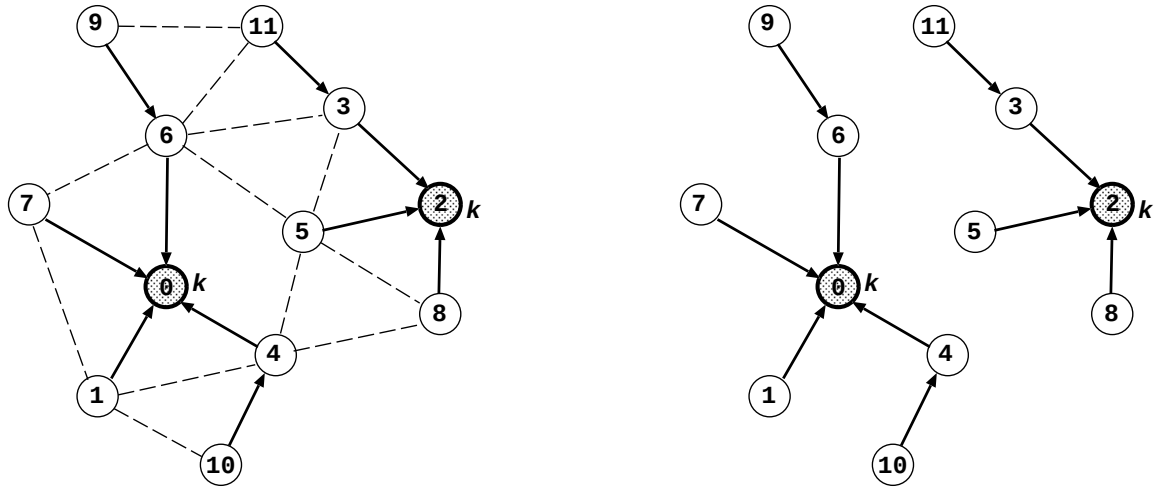
where $Nb(v)$ is the set of v's neighbor nodes. For node v to find an object k, v initiates a VALLEY-WALK and each node forwards the query according to the VALLEY-WALK algorithm until a holder receives the query. To avoid loops, we keep the last c visited nodes in the query packet or each node maintains a cache of past queries. If all neighbor nodes are already visited, the node chooses a neighbor uniformly at random as in a random walk.

For example, suppose we have a graph shown in Figure 3.2 where each node is assigned an id. For the simple presentation, instead of real values in $[0, 1)$ for node id's, we use integer values such that 0 represents the smallest id value assigned and 11 represents the largest id value assigned. Also assume that, for simplicity, the identity of the queried object is zero ($id(k) = 0$), thus, the key-distance of a node

Algorithm 1 VALLEY-WALK

```
1: {v: myself, u: v's neighbor}
2: {Nb(v): set of v's neighbor nodes }
3: {S: set of visited nodes}
4: {node v execute the following when v receives (k, S) }
5: if v is a holder for k then
6:    terminate
7: end if
8: if v is starting the query then
9:    S := {v}
10: else
11:   S := S ∪ {v}
12: end if
13: mindist := 1
14: next := ⊥
15: for all u : u ∈ Nb(v) ∧ u ∉ S do
16:   if dist_k(u) < mindist then
17:      next := u
18:      mindist := dist_k(u)
19:   end if
20: end for
21: if u = ⊥ then
22:   pick a random u ← Nb(v)
23: end if
24: send (k, S) to u
```

equals to the node identity ($dist_k(v) = id(v)$). Assume node 4 is a holder for object k and node 9 queries for k. According to the algorithm, node 9 chooses node 6 as the next node for it has the smallest key-distance among nodes $\{6, 11\}$ and node 6 in turn forwards to node 0 and node 0 forwards to 1. Since node 0 is already visited, node 1 forwards to node 4, which has the next smallest key-distance. In this way, the query arrives at the holder node 4 in 4 hops. Algorithm 1 describes the VALLEY-WALK algorithm in detail.



(a) Local minima graph **(b) Local minima trees**

Figure 3.3: Local minima

3.3 VALLEY-WALK$_{LM}$: Local Minima Based Key Distribution

Given such a structure and VALLEY-WALK algorithm described in the previous section, what is the best strategy to distribute objects such that with high probability the query finds a holder in a small number of hop counts? In this section and the following section, I discuss two strategies for storing objects; local minima based online object distribution (VALLEY-WALK$_{LM}$) and key-distance based offline object distribution (VALLEY-WALK$_{KD}$). VALLEY-WALK$_{LM}$ determines the holder based on the network topology and VALLEY-WALK$_{KD}$ determines only from the *id*-space regardless of the actual network topology. Therefore, VALLEY-WALK$_{LM}$ can distribute objects only after the network deployment (online) and VALLEY-WALK$_{KD}$ can distribute objects before the deployment (offline).

Given an object k, a node is called *Local Minima* if its key-distance to k is smaller than any of its neighbor nodes. In Figure 3.3(a), each node is labeled with

the key-distance for k. In this graph and given object k, there are two local minimum nodes 0 and 2 denoted by gray circles. The local minima based VALLEY-WALK (VALLEY-WALK$_{LM}$) stores copies of object k in each local minimum; in the example, node 0 and 2 become object holders. Note that the VALLEY-WALK starting from any node will stop at a local minimum. The arrow lines in Figure 3.3(a) represents the forwarding direction of queries by VALLEY-WALK algorithm. As seen in this example, with VALLEY-WALK$_{LM}$, any node can find a holder by performing a VALLEY-WALK and there can be no loop along the search. Figure 3.3(b) shows how VALLEY-WALK with local minima partitions the network into a set of trees with local minima as their roots.

Formally, given an undirected graph $G = (V, E)$ (Figure 3.3(a)) and an object k, we construct a directed graph $G_k = (V, E_k)$ (Figure 3.3(b)) as follows. For $(u, v) \in E$, there exists a directed edge $(u, v) \in E_k$ if and only if v has the smallest key-distance over u's neighbors including u itself, that is, for all $w \in Nb(u) \cup \{u\}$, $dist_k(v) < dist_k(w)$. Let LM_k denote the set of local minima for k. Then, G_k is a set of $|LM_k|$ disjoint trees with local minima as their roots and node v is the parent of node u if $(u, v) \in E_k$. Assuming that no two nodes have the same identities, G_k is acyclic because the key-distances along the directed edges are monotonic decreasing. Since every node is either a local minimum or has a parent node, VALLEY-WALK searching always guarantees of finding a holder.

3.3.1 The Number Of Local Minima

The limitation of VALLEY-WALK$_{LM}$ is that the replication number should equal to the number of local minima. Consider the probability that a node v with degree d_v becomes a local minimum for object k. For fixed k, $dist_k(u)$ is uniformly distributed on $[0, 1)$ and the probability that $dist_k(u) > x$ is

$$P[dist_k(u) > x] = 1 - x.$$

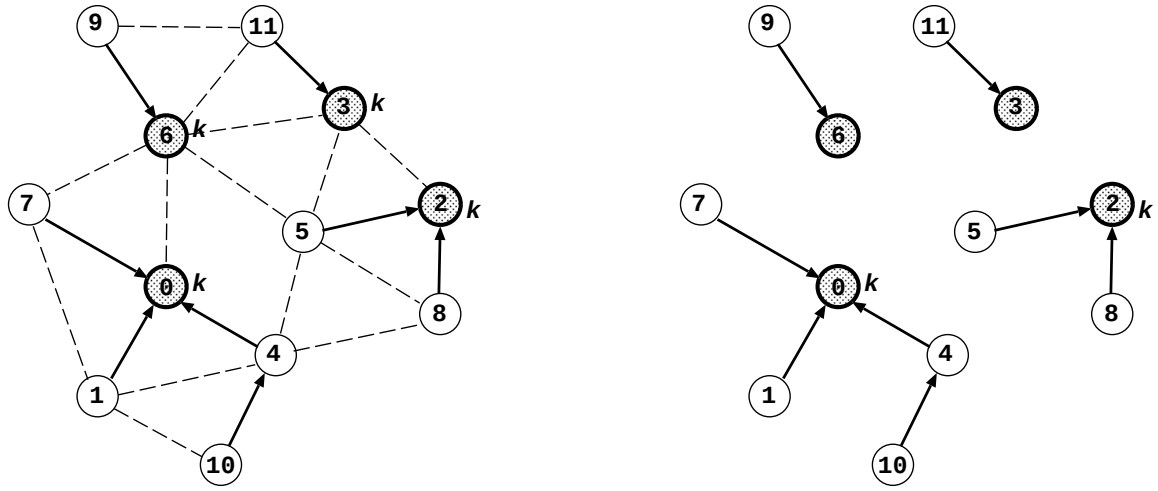
Since node id's are chosen independently, the probability that all v's neighbors have key-distances greater than x is

$$P[dist_k(u) > x \text{ for all } u \in N(v)] = (1 - x)^{d_v}.$$

Therefore, the probability that node v becomes a local minimum equals to the probability that all neighbor's key-distances are greater than v's key-distance, i.e.,

$$\begin{aligned} P[v \in LM_k] &= Pr[dist_k(u) > dist_k(v) \text{ for all } u \in N(v)] \\ &= \int_0^1 Pr[dist_k(u) > x \text{ for all } u \in N(v) | dist_k(v) = x]\ dx \\ &= \int_0^1 (1 - x)^{d_v}\ dx \\ &= \frac{1}{d_v + 1}. \end{aligned}$$

Therefore, if δ is the minimum degree of nodes, the expected number of local



(a) Local minima graph **(b) Local minima trees**

Figure 3.4: $r > |LM_k|$: Split

minima is bounded as

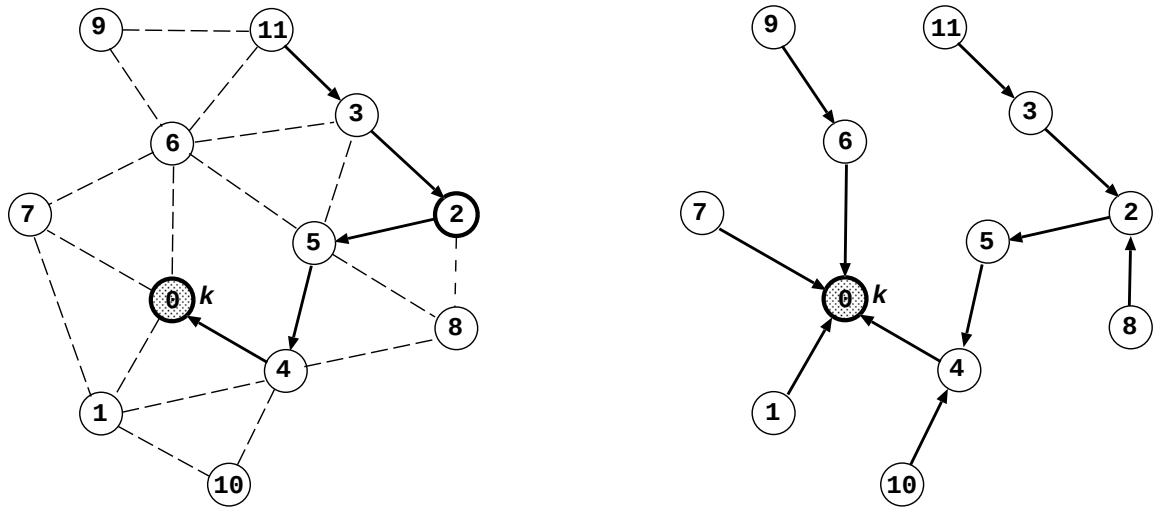
$$E[|LM_k|] \leq \frac{n}{\delta + 1}. \tag{3.1}$$

where n is the number of nodes in the network.

Since the number of local minima is a random variable whose expected value depends on the node degree and the network size, we do not have much control over the number of replications we can store in the local minima. In the following, I discuss how we can control the replication number in VALLEY-WALK$_{LM}$ method.

Suppose we want the replication number to be r. Then, the actual number of local minima $|LM_k|$ is either larger or smaller than r.

case 1: $r > |LM_k|$: When the number of objects to be stored is larger than the number of local minima, we can store objects at local minima and store the remaining objects in randomly chosen nodes. In local minima tree's point of view



(a) Local minima graph **(b) Local minima trees**

Figure 3.5: $r < |LM|$: Merge

(see Figure 3.3(b)), election of object-holders from non-local-minima is performing split operation where some subgraph G_{k_i} is being split into two subtrees. Suppose we choose node u for an object-holder and $u \in G_{k_i}$. Then, u is becoming a root node for the subtree below u and this is splitting the original subtree into two different trees. For example, in Figure 3.4, we have 4 replications while there are only two local minima. We randomly chose two more object-holders, node 3 and 6 and the original two trees are split into four new trees (Figure 3.4(b)). Note that this split operation costs nothing; the searching process stops when it reaches an object-holder.

case 2: $r < |LM_k|$: When the number of objects to be stores is less then the number of local minima, there exist local minima without objects. For example, in Figure 3.5(a), the replication number 1 is less than the number of local minima 2 (node 0 and 2) and a local minimum node 2 is not chosen as an object-holder. When the query starts at node 11 and reaches at the local minima node 2, since node 2

does not have the object, it continues with VALLEY-WALK until it reaches node the object holder node 0. As a consequence, continuous VALLEY-WALK results in a merge operation of local minima trees (see Figure 3.5(b)). I choose to call this process merge operation for ease of explanation but the actual merging path from the empty local minimum to an object-holder differs from query to query. For example, when the query starts from node 8 in Figure 3.5(a), the search path becomes $8, 2, 3, 5, 4, 0$, which makes different merged tree from when it starts from node 11. Unlike the case when $r > |LM_k|$, the mergin path can be large and the large different of $|LM_k| - r$ can significantly increase the searching cost. In the following section, I discuss how we can control the number of local minima to minimize the merging cost.

3.3.2 Tunable Local Minima

In this section, we discuss how to tune the number of local minima to minimize the searching overhead due to relatively smaller replication numbers than the number of local minima. As the merge operation becomes costly as $|LM_k| - r$ increases, we attempt to reduce $|LM_k|$ by reducing $E[|LM_k|]$. By Equation 3.1, we can suppress the expected number of local minima $E[|LM_k|]$ by increasing the minimum node degree δ. We can increase node degree by expanding the neighborhood to multiple hop neighbors such as 2-hop neighbors.

Suppose we have the replication number of r and the network size of n. Let X be a random variable representing the number of local minima $|LM_k|$. Then,

$X = \sum_{i=1}^{n} X_i$ where X_i is a Bernoulli random variable for ith node to become a local minimum and $Pr[X_i = 1] = \frac{1}{d_i+1}$ where d_i is the degree of ith node. We want to bound the following upper-tail probability $Pr[X > x]$. We introduce a Binomial random variable $Y \sim B(n, \frac{1}{\delta+1})$ where δ is the minimum degree in the network. Y represents an ideal case where each node has identical probability of becoming a local minimum and, since $\frac{1}{d_i+1} \leq \frac{1}{\delta+1}$, the probability becomes an upper bound of the actual case. Therefore,

$$\begin{aligned} Pr[X > x] &\leq Pr[Y > x] && (3.2) \\ &= \sum_{i=x+1}^{n} \binom{n}{i} \left(\frac{1}{\delta+1}\right)^{i} \left(1 - \frac{1}{\delta+1}\right)^{n-i} && (3.3) \end{aligned}$$

Now we can make the probability that $|LM_k| > r$ as small as we want by adjusting the minimum degree δ. Suppose we want $Pr[|LM_k| > r] < \epsilon$ for $\epsilon > 0$, then we can find the smallest $\hat{\delta}$ which satisfies $Pr[Y > r] < \epsilon$. For example, suppose there are 100 nodes ($n = 100$) and we want to distribute 10 object copies. To have $Pr[|LM_k| > 10] < 0.05$, $\delta = 15$ suffices since $Pr[|LM_k| > 10] < Pr[Y > 10] = 0.048$. Similarly, we can also increase $|LM_k|$ when $|LM_k| < r$ until $|LM_k| \sim r$ by decreasing the maximum node degree Δ.

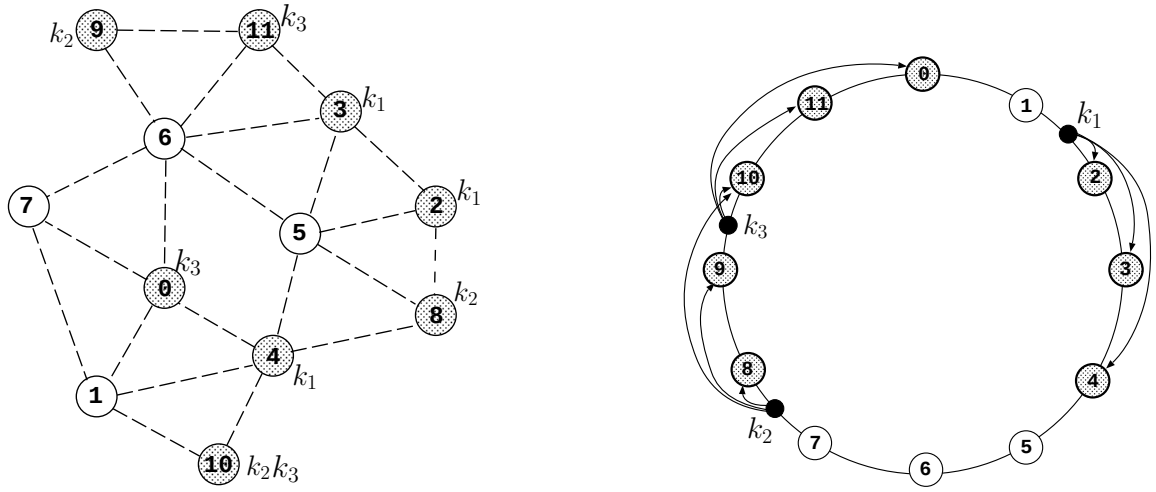
Tuning δ and Δ: To increase δ as we want, a node with degree less than δ expands its neighborhood by adding randomly chosen 2-hop neighbors (or 3-hop if needed) into their neighbor set until its degree becomes δ. Also one can decrease the maximum node degree Δ by shrinking the neighborhood so that each node has less than or equal to Δ neighbors. However, the shrink of neighborhood can cause an

apparent degrade of searching performance because the dropped one-hop neighbor might be an object-holder. Therefore, in practice, we do not decrease the degree below its natural one-hop degree. We only expand the neighborhood if needed. With adjusted neighborhood, the neighbor relationship becomes asymmetric; v can be a neighbor of u but u is not for v. The VALLEY-WALK$_{LM}$ construction is still valid with the asymmetry neighborhood relationship.

3.4 VALLEY-WALK$_{KD}$: Key Distance Based Key Distribution

VALLEY-WALK$_{LM}$ only works when we are able to store objects in local minima online. When it is not the case, for example, there is no secure way to find local minima or transfer objects to local minima, we should store objects in the nodes prior the deployment. We need a strategy of storing objects such that object-holders become a local minimum or near a local minimum with high probability. I propose a heuristic strategy which stores objects in the nodes which has as small key-distance to object as possible.

Given a replication number r and an object k, we assign k to the nodes with r smallest key-hop-counts, that is, $\{v|hop_k(v) \leq r\}$. Figure 3.6 shows an example of object assignments when $r = 3$. In Figure 3.6, nodes $2, 3, 4$ are assigned as object-holders of object k_1 since they are closest to the object ($hop_{k_1}(2) = 1, hop_{k_1}(3) = 2, hop_{k_1}(4) = 3$). Likewise, object k_2 is stored in nodes $8, 9, 10$ k_3 is in nodes $10, 11, 0$. Note that this object assignment is done before deployment and the network topology is not known at that time. The assignment requires to sort nodes by their



(a) Network topology **(b) Hash space**

Figure 3.6: Key distance based pre-distribution

node *id*s and to find the closest node for each object, thus the total processing time is $O((n+m)\log n)$. Figure 3.6(b) shows an example deployment with the pre-deployment object assignment; object holders are shown as gray nodes with their objects shown next to them.

Since we determine key-holders before deployment, the key-holders may or may not become local minima. Figure 3.7(a) shows a deployment of the network and the local minima tree for key k_2. In the figure, gray nodes are key-holders and thick circles are local minima. The arrow lines points the direction toward local minima which VALLEY-WALK may follow during the search. As shown in the figure, k_2 is stored in nodes $8, 9, 10$ and they are all local minima. There is also one more local minima node 0 but it does not have k_2. For k_1, however, while node 2 and 4 are local minima, node 3 is not. As shown in Figure 3.7(b), there exists only two local minima (node 2 and 4) but also node 3 become a root because it has the key.

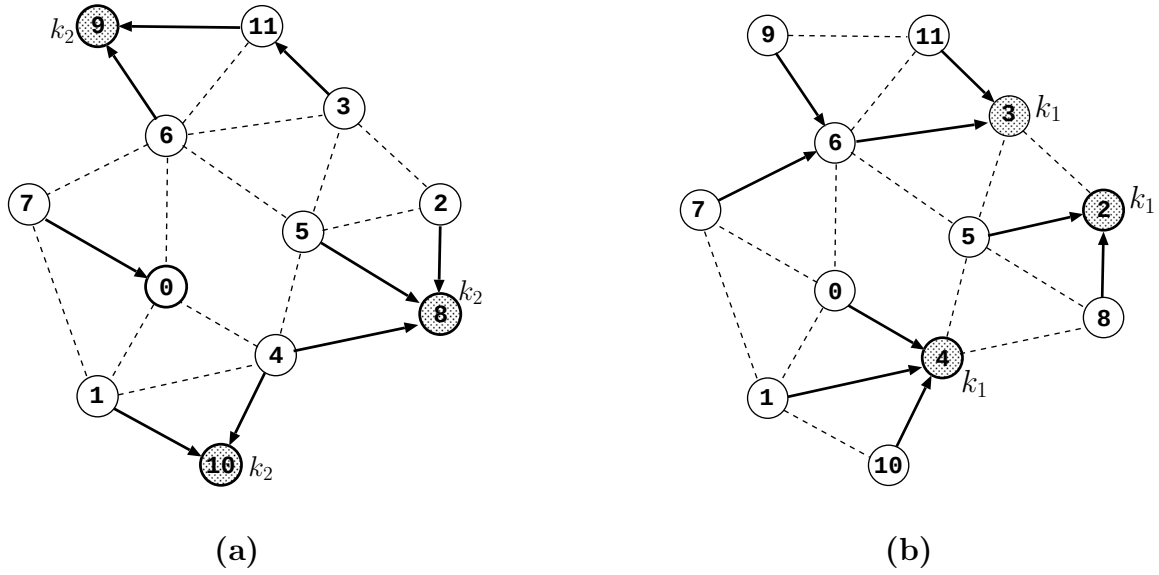


Figure 3.7: Key-holders and local minima by VALLEY-WALK$_{KD}$

3.5 Performance Analysis

In this section, we analyze the performance of VALLEY-WALK$_{LM}$ and VALLEY-WALK$_{KD}$ in terms of search path. I model the VALLEY-WALK as a series of independent random variables on $[0, 1)$ and analyze the tail probability, the probability that the search length is larger then a given value. The analytic results show that, most of the time (90%), the search succeeds in 3 hops or 6 hops by VALLEY-WALK$_{LM}$ or VALLEY-WALK$_{KD}$, respectively. Comparison with packet simulations show that our analytic bounds are tight.

3.5.1 Model Of VALLEY-WALK$_{LM}$

For simplicity, I assume that we can store objects in and only in local minima: $r = |LM_k|$. In this case, all the local minima become object-holders and, therefore, VALLEY-WALK stops whenever it reaches a local minimum. A node is a local

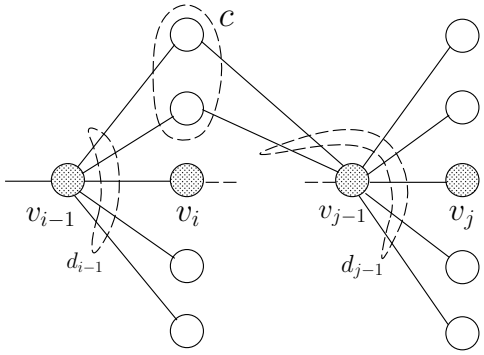
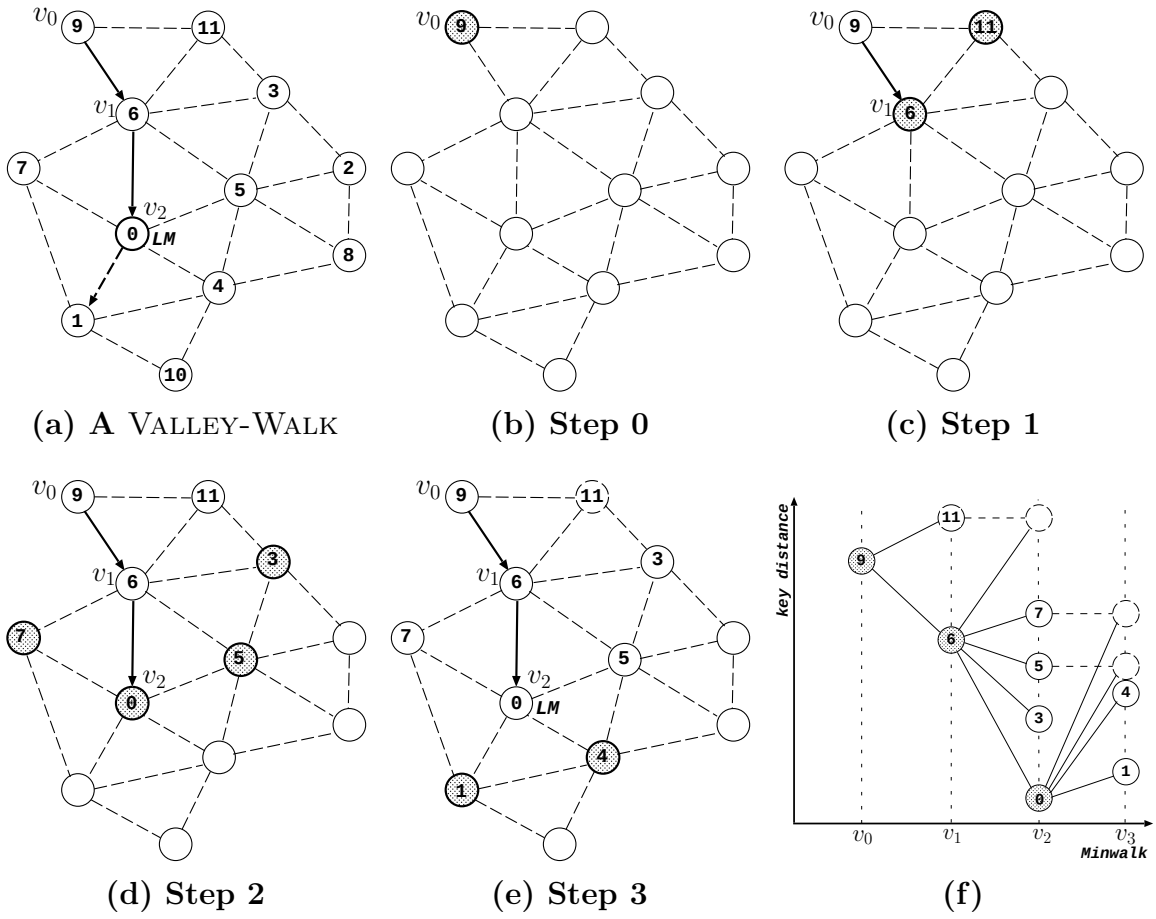


Figure 3.8: Dependency between consecutive forwardings in a VALLEY-WALK

minimum if its identity is smaller than the identity of the next node chosen by VALLEY-WALK algorithm. Figure 3.9(a) shows an example of a VALLEY-WALK on a graph of size 12 where the key-distance of each node is shown on the circle. Starting at node 9, the query finds a local minimum node 0 with a VALLEY-WALK-path $v_0 = 9, v_1 = 6, v_2 = 0$. Node 0 is a local minimum because its next node chosen by VALLEY-WALKis node $1 > 0$. In the following, I denote the minimum key-distance of a set of nodes V by $dist_k\{V\} = \min\{dist_k(v)|v \in V\}$.

I model the VALLEY-WALK as a random process (a series of random variables) as follows. Given an object k and a querying node v_0, let $v_0, v_1, v_2, \ldots$ be the series of nodes visited by VALLEY-WALK, called VALLEY-WALK-path. Let $X_i = dist_k(v_i)$ be a random variable for the identity of v_i on the VALLEY-WALK-path and, then $X_0, X_1, X_2, \ldots$ is a random process. The search length of VALLEY-WALK$_{LM}$ denoted by L, is the smallest integer i such that $X_i < X_{i+1}$.

In general, for $i < j$, X_i and X_j are not independent because they are minimum key-distances from two possibly non-distinct sets, i.e., $X_i = dist_k\{Nb(v_{i-1})\}$ and $X_j = dist_k\{Nb(v_{j-1})\}$, and possibly $Nb(v_{i-1}) \cap Nb(v_{j-1}) \neq \emptyset$. Figure 3.8(a) depicts



(a) A VALLEY-WALK **(b) Step 0** **(c) Step 1**

(d) Step 2 **(e) Step 3** **(f)**

Figure 3.9: Example of a VALLEY-WALK

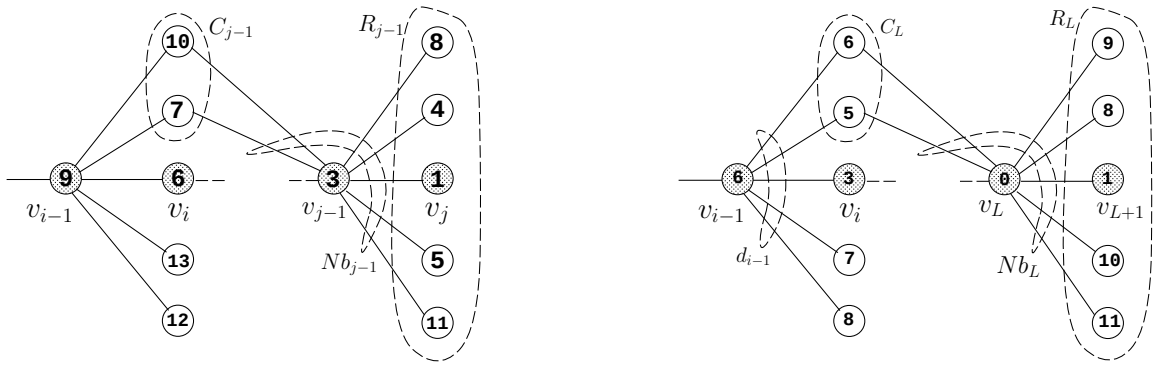
the dependency between X_i and X_j. In the figure, X_i is the minimum key-distance among d_{i-1} neighbors of node v_{i-1} and X_j is that of d_{j-1} neighbors of node $v_{j=1}$. Since node v_{i-1} and v_{j-1} can share c neighbors, there can be dependency between X_i and X_j in the form of $X_i < c$ and $X_j < c$ where $c = dist_k\{Nb(v_{i-1}) \cap Nb(v_{j-1})\}$. By the following new experiment, I create an *independent* random process whose distribution of search length is exactly the same as the original dependent random process.

We convert the original VALLEY-WALK to a modified version *incremental-*VALLEY-WALK, which has the same distribution of the search length with the orig-

inal. In the following description, we say "reveal key-distance of a node" to mean that we assign a node the key-distance as the same value when we assign them in the original VALLEY-WALK. For simple presentation, we say node v has key-distance i when v has ith smallest key-distance ($i = 0, 1, 2, \ldots$) among all the nodes in the network.

In the original VALLEY-WALK(e.g., Figure 3.9(a)), we choose key-distances of all the nodes at random before starting the VALLEY-WALK. In the incremental-VALLEY-WALK, however, at each step of the walk, we reveal key-distances only for the nodes that we need to know to determine next node (Figure 3.9(b)–(e)). For example, in Figure 3.9(b), we start from node v_0 by revealing its key-distance, say 9. To determine next node v_1, we reveal key-distances of node 9's neighbor nodes, say 6 and 11 as in Figure 3.9(c). Then, node 6 becomes v_1 since it has the smallest key-distance among node v_0's neighbors. We continue for finding v_2 because v_0 is not a local minimum for $dist_k(v_1) < dist_k(v_0)$. In the next step, we reveal key-distances of v_1's neighbor nodes which has not yet been revealed, i.e., node $7, 0, 5, 3$ in Figure 3.9(d) and chooses node 0 as v_2. In the next step shown in Figure 3.9(e), we find that v_2 is a local minimum because all revealed key-distances of v_2's neighbors are larger than v_2's, and we stop the search at v_2.

Figure 3.9(f) illustrates the incremental-VALLEY-WALK as a random process where, at node v_i ($i = 0, 1, 2, \ldots$), we reveal key-distances for v_i's neighbor nodes that is not revealed yet, and choose the smallest value as the key-distance of the $(i+1)$th node. The solid lines shows neighbor relationships between nodes and the horizontal dashed lines and dashed circles show that the node is a neighbor of both



(a) Case 1 **(b) Case 2**

Figure 3.10: Incremental VALLEY-WALK

previous and current node and, thus, we do not reveal key-distance of such a node since it already has been.

Let $X'_0, X'_1, X'_2 \ldots$ be the random process in the incremental-VALLEY-WALK. Then, X'_i's are independent random variables since X'_i is determined from disjoint sets. For example, in Figure 3.9(f), node 6 is the smallest of $\{6, 11\}$ and node 0 is the smallest of $\{0, 3, 5, 7\}$.

Lemma 3.5.1 *The distribution of search length L' in the incremental-*VALLEY-WALK *has the same distribution of search length L in* VALLEY-WALK.

Proof It is suffice to show that, if $X_0 > X_1 > \ldots > X_L < X_{L+1}$, then $X'_i = X_i$ for $i \leq L$ and $X'_L < X'_{L+1}$.

Note that $X'_0 = X_0$. Assume that $X'_i = X_i$ for $i < j \leq L$. Then, $X'_j < X'_i$ for $i < j$. Let Nb_{j-1} be v_{j-1}'s neighbors, $C_{j-1} \subset Nb_{j-1}$ be v_{j-1}'s neighbors whose key-distances are already revealed, and $R_{j-1} \subset Nb_{j-1}$ be v_{j-1}'s neighbors whose key-distances are not revealed yet (see Figure 3.10(a)). Note that all nodes in C_{j-1} have

larger key-distances than v_{j-1}. Since $X'_j < X'_{j-1}$, $v_j \notin C_{j-1}$. Therefore, minimum key-distance of C_{j-1} is larger than X'_j which is the minimum key-distance of R_{j-1}. Since $X'_j < dist_k\{C_{j-1}\}$ and $X'_j = dist_k\{R_{j-1}\}$, $X'_j = dist_k\{Nb_{j-1}\}$, thus $X'_j = X_j$. By induction, $X'_i = X_i$ for $i \leq L$

In Figure 3.10(b), since $X_L < X_{L+1}$, $dist_k\{Nb_L\} > X_L$, therefore, $dist_k\{R_L\} > X_L$. Since $X'_{L+1} = dist_k\{R_L\}$ and $X'_L = X_L$, $X'_L < X'_{L+1}$. ∎

3.5.2 Analysis Of VALLEY-WALK$_{LM}$-*iid*

In the following analysis, I focus on the incremental-VALLEY-WALK for it is equivalent to the original VALLEY-WALK with local minima object distribution in terms of the distribution of search length. We denote X_i for X'_i, the key-distance of ith node in the searching path of incremental VALLEY-WALK.

Let F_i and f_i be the *cdf* (cumulative probability function) and *pdf* (probability density function), respectively, for the random variable X_i. Note that X_i's follow different distributions because they have different size of node sets from which the next node is chosen, i.e, $F_i \neq F_j$ if $i \neq j$. For example, in Figure 3.9(f), X_1 is the smallest key-distance of two nodes while X_2 is that of 4 nodes.

To get an insight of my analysis on VALLEY-WALK$_{LM}$, we first analyze the simplified version of VALLEY-WALK$_{LM}$, called VALLEY-WALK$_{LM}$-*iid* where X_i's follow the same distribution, i.e., $F_i = F_j$ for all $i, j \geq 0$. The Theorem 3.5.2 shows that, if X_i's are *iid* (independent identical distribution), the distribution of the search length is independent of the actual distribution of X_i.

Theorem 3.5.2 (VALLEY-WALK$_{LM}$-*iid*) *If $X_0, X_1, \ldots$ follow identical independent distribution, then, the tail probability of the search length L is*

$$Pr(L \geq k) = \frac{1}{(k+1)!} \tag{3.4}$$

and

$$E[L] = e - 2 \tag{3.5}$$

where e is the base of natural logarithm.

Proof Please see Appendix A.1 ∎.

The result of Theorem 3.5.2 shows that, if incremental-VALLEY-WALK chooses the next node from the same number of neighbors at all steps, the tail probability becomes $1/(k+1)!$. Also the distribution of L is independent of X_i's distribution. For example, even if we choose the second smallest key-distances from the candidate nodes, the distribution of L remains the same.

3.5.3 Analysis Of VALLEY-WALK$_{LM}$

In VALLEY-WALK$_{LM}$, X_i's follow the different but independent distributions. Let d_i be the degree of node v_i and $\hat{d}_i$ be the number of revealed nodes by node v_i. Then, $\hat{d}_i < d_i$. We define the previous node of the first node v_0 as v_{-1} and let $d_{-1} = \hat{d}_{-1} = 1$. Then, X_i is the smallest value out of $\hat{d}_{i-1}$ random values on $[0, 1)$

for $i = 0, 1, 2, \cdots$, and therefore,

$$\begin{aligned} F_i(x) &= Pr(X_i \leq x) \\ &= 1 - (1-x)^{\hat{d}_{i-1}} \qquad (3.6) \\ f_i(x) &= F_i'(x) = \hat{d}_{i-1}(1-x)^{\hat{d}_{i-1}-1} \qquad (3.7) \end{aligned}$$

for $i \geq 0$.

The probability that a node v_i on the path becomes a local minimum is

$$\begin{aligned} Pr(v_i \in LM_k) &= Pr(dist_k(v_i) < dist_k(v_{i+1})) \qquad (3.8) \\ &= \int_0^1 Pr(x < dist_k(v_{i+1}) | dist_k(v_i) = x)\ dx \qquad (3.9) \\ &= \int_0^1 (1-x)^{\hat{d}_i} \qquad (3.10) \\ &= \frac{1}{\hat{d}_i + 1} \qquad (3.11) \end{aligned}$$

Therefore, the probability that a node in the path becomes a local minimum decreases as the number of candidate nodes for the next node increases, thus the search path increases. Let Δ be the maximum degree in the network. Then the modified network where all the nodes have degree Δ gives an upper bound of the expected search length and also the tail bound. We define $F(x) = 1 - (1-x)^{\Delta}$ and $f(x) = F'(x)$.

Theorem 3.5.3 (VALLEY-WALK$_{LM}$) *For the search length L of* VALLEY-WALK$_{LM}$,

$$Pr(L \geq k) \leq \frac{1}{k!} \sum_{i=0}^{k} \binom{k}{i} \frac{(-1)^i}{\Delta i + 1} \tag{3.12}$$

$$\leq \frac{1}{k!} \tag{3.13}$$

Proof Please see Appendix A.2 ∎.

Morselli et al. [12] showed a tail bound of $\frac{\Delta}{2^{k-1}}$. Our bound is tighter than Morselli's because

$$\frac{1}{k!} \leq \frac{1}{2^{k-1}} \tag{3.14}$$

$$\leq \frac{\Delta}{2^{k-1}} \tag{3.15}$$

for all $k \geq 0$.

Figure 3.11 shows that our analysis is significantly closer to the actual tail probability than Moriselli's. The big difference between the cross line and the circle line shows that my analysis is much tighter than Moriselli's. The starred line shows the results of packet simulations for VALLEY-WALK$_{LM}$ with $n = 50$. In the packet simulations, the network had the average degree of 6.5 and the maximum degree of 11. We used the same maximum degree $\Delta = 11$ in the analysis results. Although we see the difference between the packet simulation and my analysis, the error is caused by the large variance in the degree of the network. When the degree has a small variance, my analysis is very close to the simulation results. The average

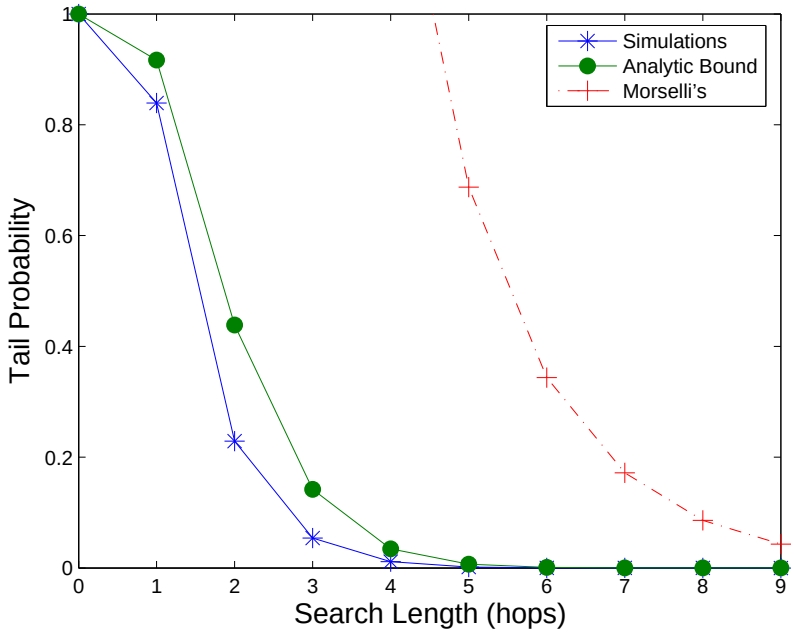


Figure 3.11: Comparison of my analysis, Morselli's, and ns2 simulation results for VALLEY-WALK$_{LM}$

search length and its analytic upper bound was 1.49 and 1.54, respectively.

3.5.4 Analysis Of VALLEY-WALK$_{KD}$

VALLEY-WALK$_{KD}$ strategically store objects in the nodes before deployment such that object-holders become local minima with high probability. Given a replication number r and an object k, the r closest nodes in terms of key-distance $dist_k(v)$ (distance from $id(k)$ to $id(v)$ in the clockwise direction) become object-holders. In other words, the nodes within r clockwise hops, i.e., $hop_k(v) \leq r$, become object-holders (see Figure 3.6).

Given an object k and n nodes $v_1, v_2, \ldots, v_n$, let random variables $D_1, D_2, \ldots, D_n$ be key-distances from the nodes , i.e., $D_i = dist_k(v_i)$. Then, D_i's follow the uniform

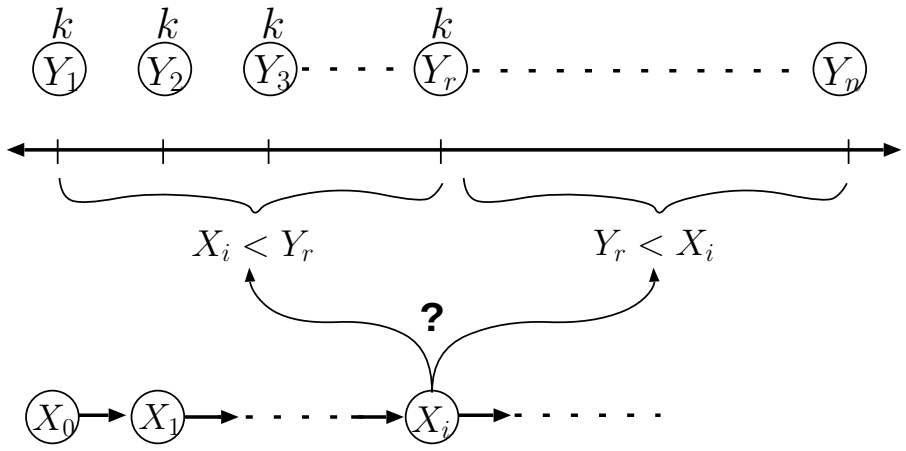


Figure 3.12: Analysis model of VALLEY-WALK$_{KD}$

distribution on $[0, 1)$. Let $Y_1, \ldots, Y_n$ be the same values as $D_1, \ldots, D_n$ in increasing sorted order. Then, with the replication number r, the set of object-holders is $\{Y_1, Y_2, \ldots, Y_r\}$. In Figure 3.12, Y_j is the random variables for jth smallest key-distance among n nodes and X_i is a random variable for the key-distance of the ith node in the search path. As shown in the figure, the ith node becomes an object-holder if its key-distance is less than the largest key-distance of the object-holder.i.e., $X_i < Y_r$. To analyze the search length, therefore, we have to know the probability distribution of both Y_j and X_i as well as their joint probability.

For later calculations, I introduce the following lemma for the definite integration.

Lemma 3.5.4 *For $i, k \geq 0$,*

$$\int_0^1 x^i (1-x)^k \, dx \;=\; \frac{1}{(i+k+1)\binom{i+k}{i}} \tag{3.16}$$

Proof Please see Appendix A.3 ∎.

Theorem 3.5.5 answers for the probability distribution of Y_k, the kth smallest value of n uniform random variables.

Theorem 3.5.5 (Distribution of Y_k) *Let $D_1, D_2, \ldots, D_n$ be a set of continuous random variables which independently follow the uniform distribution on $[0, 1)$. Denote the kth smallest value among them by a random variable Y_k. Then, for $y \in [0, 1)$ and $1 \leq k \leq n$, the cdf of Y_k is $F_{Y_k}(y) = Pr(Y_k \leq y) = 1 - Pr(Y_k > y)$ where*

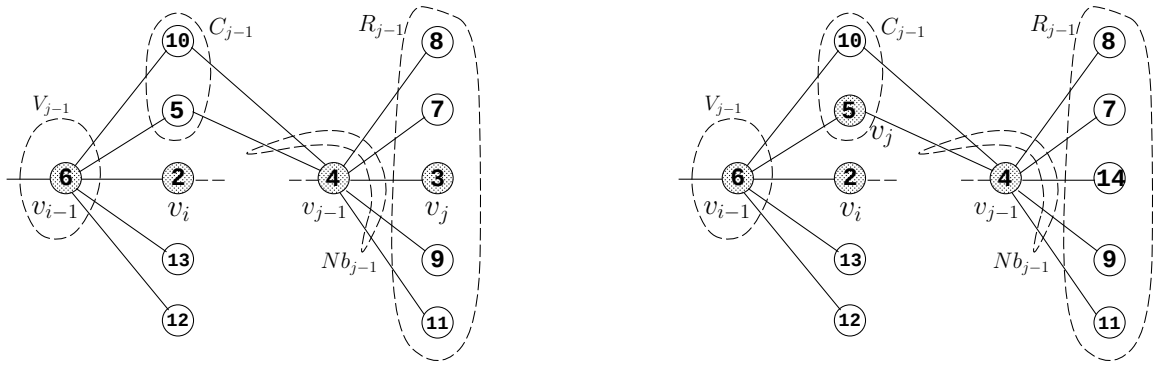
$$Pr(Y_k > y) = 1 - \sum_{i=0}^{k-1} \binom{n}{i} y^i (1-y)^{n-i} \tag{3.17}$$

with

$$E[Y_k] = \frac{k}{n+1}$$

Proof Please see Appendix A.4 ∎.

Suppose the search path by VALLEY-WALK$_{KD}$ is $\{v_0, v_1, v_2, \ldots\}$ where v_i is the ith node in the path. Let $X_0, X_1, X_2, \ldots$ be the random variables for the key-distances of the nodes along the path, i.e., $X_i = dist_k(v_i)$. Like VALLEY-WALK$_{LM}$, I can make a similar argument about the incremental-VALLEY-WALK such that the distribution of search length by incremental-VALLEY-WALK equals to that of VALLEY-WALK with VALLEY-WALK$_{KD}$. However, I choose not to exploit such conversion approach because replacing the number of revealed nodes at each step with the smallest number of revealed nodes (mostly 0) fails to provide a tight bound of the search length. Instead, I take a simplified assumption, independence between



(a) Case 1 **(b) Case 2**

Figure 3.13: VALLEY-WALK$_{KD}$ and incremental VALLEY-WALK$_{KD}$

X_i's, with sacrifice of upper bound. By independence between X_i's, I mean that, at each step i for determining v_i, we re-sample key-distances of all v_{i-1}'s neighbors. By simulations, I show that the independence assumption generates bound-like approximations to the tail probability of search length and the error is low.

Lemma 3.5.6 *The probability that the first node v_0 becomes an object-holder is*

$$Pr(v_0 \text{ becomes an object-holder}) = \frac{r}{n+1} \tag{3.18}$$

and the probability that ith node in the search path v_i ($i > 0$) becomes an object-holder is

$$Pr(v_i \text{ becomes an object-holder}) = d\sum_{i=0}^{r-1} \frac{\binom{n}{i}}{(n+d)\binom{n+d-1}{i}} \tag{3.19}$$

where d is the degree of v_{i-1}.

Proof Please see Appendix A.5 ∎.

Theorem 3.5.7 (simplified VALLEY-WALK$_{KD}$**)** *For $k > 0$, a tail probability of the search length L with the independence assumption is*

$$Pr(L \geq k) \leq (1 - p_0)(1 - p)^{k-1} \tag{3.20}$$

where $p_0 = r/(n+1)$ and, given the minimum node degree δ,

$$p = \delta \sum_{i=0}^{r-1} \frac{\binom{n}{i}}{(n+\delta)\binom{n+\delta-1}{i}}$$

Proof Let $p_i (i = 0, 1, 2, \ldots)$ be the probability that the ith node becomes an object-holder. By Lemma 3.5.6, $p_0 = r/(n+1)$ and

$$p_i = d \sum_{i=0}^{r-1} \frac{\binom{n}{i}}{(n+d)\binom{n+d-1}{i}}$$

for $i > 0$, where n is the number of nodes, r is the number of object copies, and d is the degree of the previous node. Let p be the upper bound of p_i's such that

$$p = \delta \sum_{i=0}^{r-1} \frac{\binom{n}{i}}{(n+\delta)\binom{n+\delta-1}{i}}$$

where δ is the smallest degree in the network. Then, $p \leq p_i$ for all $i > 0$. We can consider the distribution of the search length L as a geometric random variable with

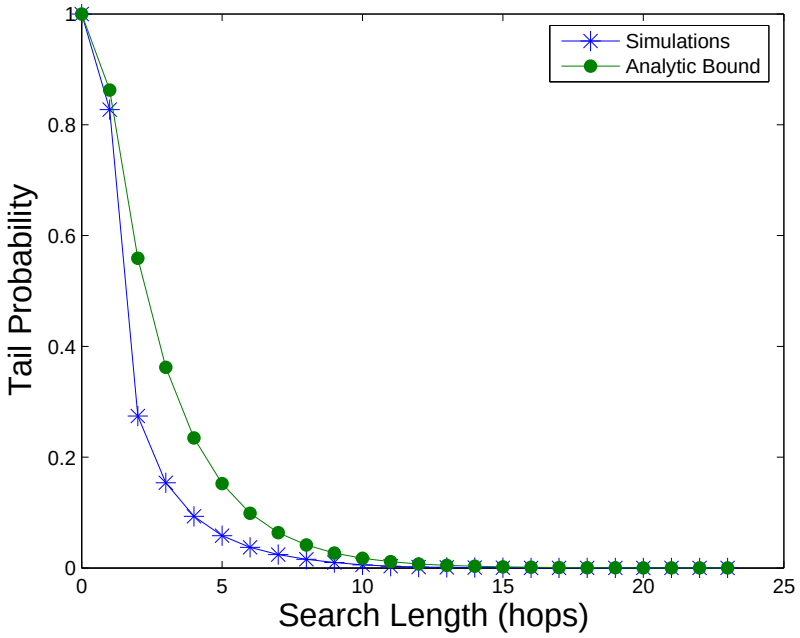


Figure 3.14: Comparison of my analysis and ns2 simulation for VALLEY-WALK$_{KD}$.

p_0 for the first trial and p for the rest of the trials. For $k > 0$, we have

$$Pr[L \geq k] \leq (1 - p_0)(1 - p)^{k-1}$$

∎

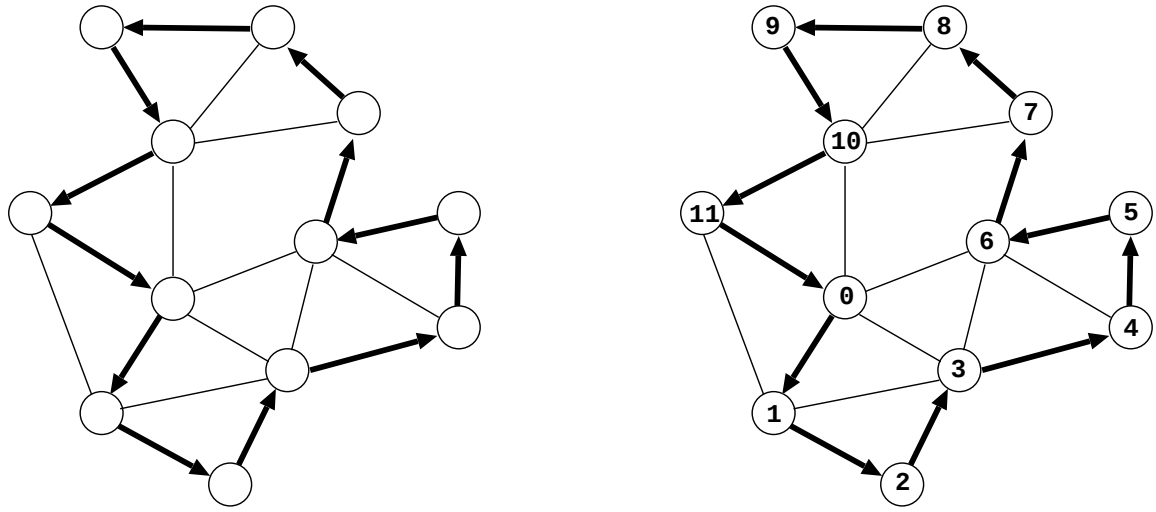
Figure 3.14 shows the tail probability $P(L > k)$ by both packet simulations and analytic bounds of simplified VALLEY-WALK$_{KD}$ in Theorem 3.5.7. In the packet simulations, the minimum degree was 4 and I used the same $\delta = 4$ for analytic bounds. The difference between two lines is due to high variance of node degree and two lines are very close when the variance of node degree is low.

Chapter 4

RIGS: A Topology-Dependent DHT With Ring Interval Graph

VALLEY-WALK$_{LM}$ and VALLEY-WALK$_{KD}$ bear limitations. With VALLEY-WALK$_{LM}$ a node becomes a local minimum if it has the lowest *id* among its neighbors. The number of local minima in the network depends on the degree of nodes and we can control the number of local minima by adjusting the node degrees. Let the *natural* local minima be the local minima generated without adjusting node degree. Simulation results show that VALLEY-WALK$_{LM}$ performs close to optimal when the number of *natural* local minima is close to the number of replications. To make the number of local minima close to the given replication number, nodes shrink or expand the neighborhood to change their degrees. On the other hand, VALLEY-WALK$_{KD}$ requires pre-distribution with processing time of $O((n+m)\log n)$. Also, VALLEY-WALK$_{KD}$ search can be arbitrarily large if the network has only a small number of replications, such as one or two. Those limitations of VALLEY-WALK$_{LM}$ and VALLEY-WALK$_{KD}$ comes from their loosely-structured characteristics.

In this chapter, I propose a structured searching scheme RIGS for multi-hop wireless networks which guarantees successful searches and performs close-to-optimal. RIGS is a fully structured searching mechanism. The structure exploits the locality of neighboring nodes and can be generated in a distributed fashion. All the nodes need to know only neighbor information and the network size.

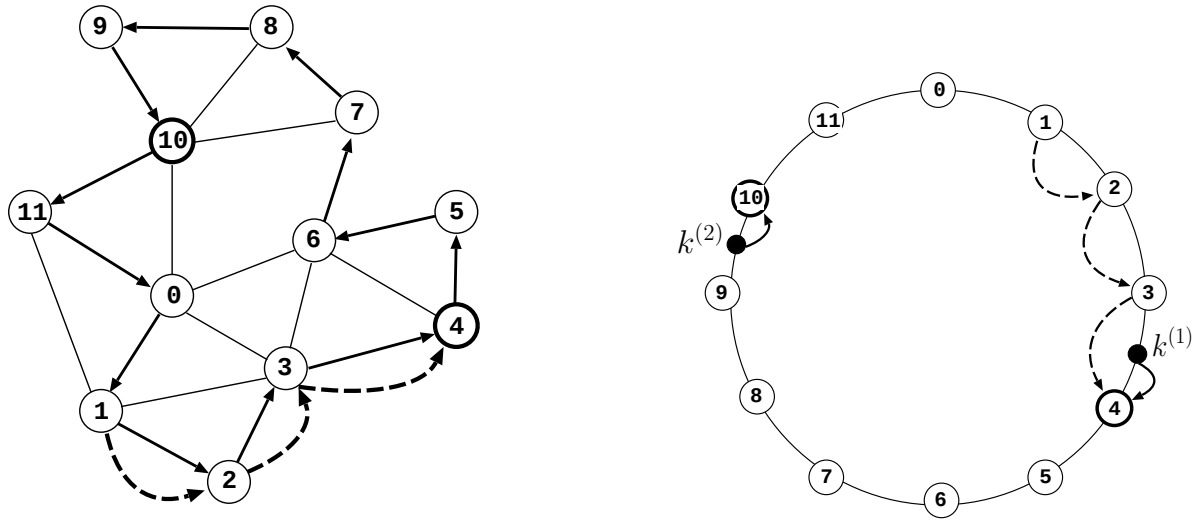


(a) Hamiltonian cycle **(b) Node numbering**

Figure 4.1: Hamiltonian cycle and node numbering along the cycle

4.1 Hamiltonian Search

To provide an insight of my approach, I begin with explaining the basic idea. Consider the following ring-based structure, as in Chord [14]. In Chord, every node are places on the ring-shaped id space and each node keeps a pointer to the next node on the ring. Therefore, given a data with an identity on the ring, every node can forward the query toward the destination by passing to the next node on the ring. We can build such a ring for a wireless network if the network has a Hamiltonian cycle such that we can visit every node exactly once by following the cycle in one direction. Unlike Chord, the next node on the ring is a one-hop neighbor in the network topology. Figure 4.1(a) show an instance of a graph with a Hamiltonian cycle as thick arrows. Given network size n, We can construct a consistent hashing space along the ring as follows; starting from a pivot node, we assign each node an *id* from $\{0, 1, 2, \ldots, n-1\}$ with an increasing order along the cycle. Figure 4.1(b)

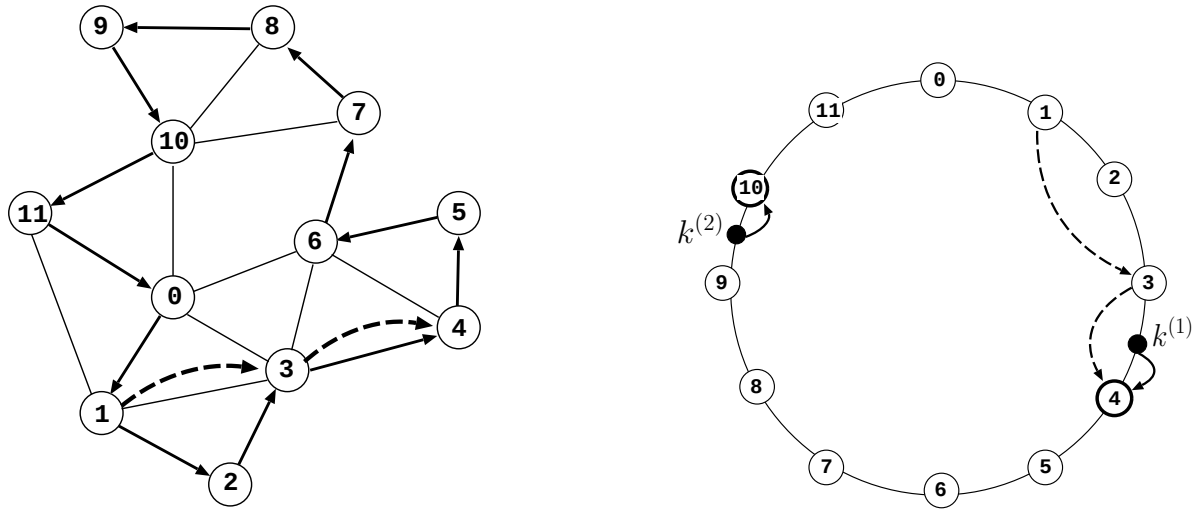


(a) Network topology **(b) Key hash space**

Figure 4.2: Search with the Hamiltonian ring

shows an example of node *id* assignments. Given a replication number r and an object k, we hash k and $r-1$ virtual objects on the same hash space $[0, 12)$ as follows. Given an object id k and a replication number r, we generate virtual object *id*s as $k^{(i)} = k + \frac{n}{r}(i-1)$ where $i = 1, \ldots, r$. Key-holders of k is the successive nodes of the virtual objects in a clockwise direction. For example, Figure 4.2(b) shows when r is 2 and k is 3.5. In the figure, the object replications are $\{k^{(1)}, k^{(2)}\} = \{3.5, 9.5\}$ and bold nodes 4 and 10 become object-holders because they are successive nodes of each object replications.

Searching is as simple as forwards the query packet to one of its neighbors on the ring whose *id* is the closest to the object. In the example, suppose node 1 wants to find a copy of object $k = 3.5$. Since $dist_{k^{(1)}}(2) = 1.5$ and $dist_{k^{(2)}}(0) = 2.5$, node 1 forwards the query to node 2, which in turn forwards to node 3 and finally the object holder 4 receives the query (Figure 4.2(a)).



(a) Node mapping **(b) Key hashing**

Figure 4.3: Hamiltonian cycle and node numbering along the cycle

Note that the search path on the ring is not necessarily the shortest path toward the destination. In the previous example, although node 2 is the closest to the destination 3.5 among 1's neighbors on the ring, node 3, 1's another neighbor on the graph, is closer to the destination than node 2. We can exploit this opportunity to expedite the search; forward to a neighbor who has closest id to the object even if it's not on the ring. For example, node 1 can improve search performance by forwarding to node 3 instead of 2, and new search path becomes $1, 3$, and 4 (Figure 4.3). In this way, the link redundancy of the graph provides many shortcuts toward destinations. This opportunity of shortcut is inherent to the wireless network where a node naturally can communicate with its neighbor nodes in vicinity without any extra effort for communication. In the structured schemes proposed in this work, I exploit this abundant local information to expedite the search.

Not every graph, however, has a Hamiltonian cycle and determining if one

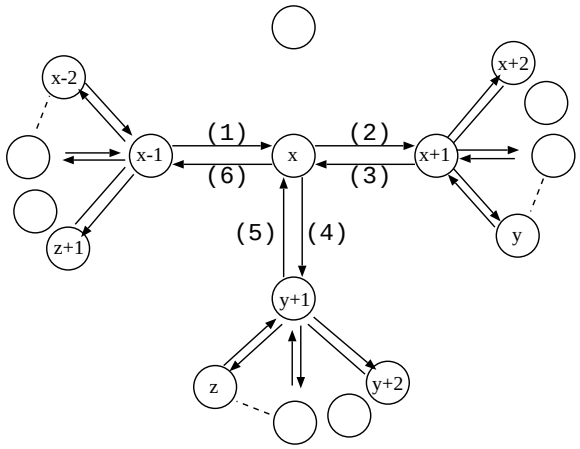
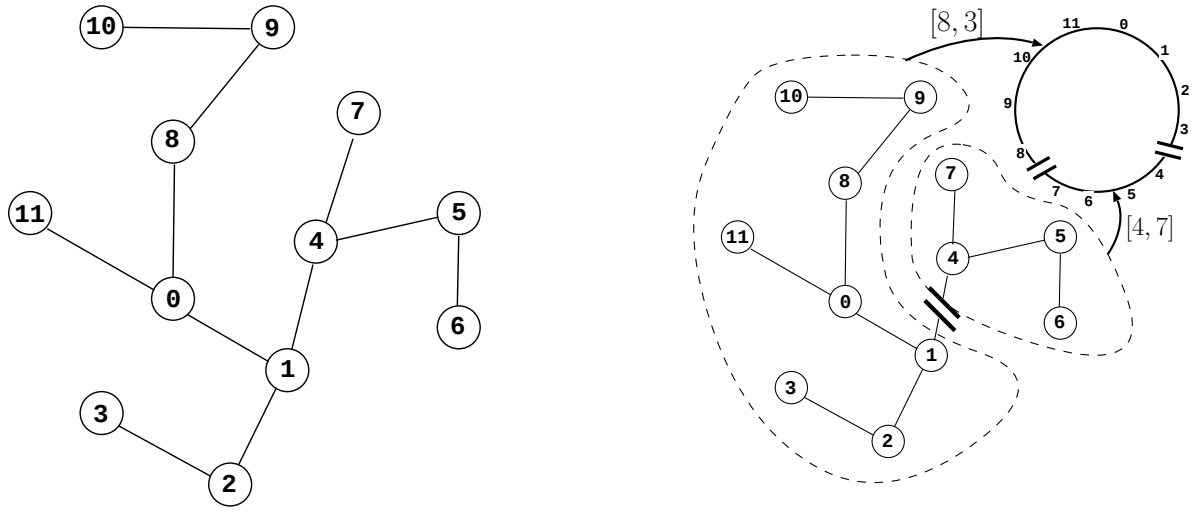


Figure 4.4: Construction of non-Hamiltonian search

exists in a graph is NP-complete [69]. Dirac showed that a graph is Hamiltonian if each node has degree $n/2$ or greater and Ore [70] showed that a graph is Hamiltonian if, for each non-adjacent vertex pair, the sum of their degrees is n or greater. Later, Bondy-Chvtal generalized Dirac and Ore's results; A graph is Hamiltonian if and only if its closure is Hamiltonian. A closure of a graph is a new graph constructed from G by edges (u, v) if the sum of their degree is n or greater.

Without having or knowing of a Hamiltonian cycle, it is inevitable to visit some nodes more than once. Suppose we have a non-Hamiltonian cycle with the following property. Since the path is cyclic, each node has the same number of incoming paths and outgoing paths (see Figure 4.4). Suppose any outgoing path comes back through the same node. Imagine we follow the path and assign node *id*s in an increasing order to each node we visit for the first time. For example, consider a node v in Figure 4.4 where node v has four neighbors v_1, v_2, v_3, and v_4. In this figure, a non-Hamiltonian cycle visits v_1, v (message (1)), and v_3 (message (2)) assigning node *id*s $x-1, x, x+1$, respectively. After coming back with message



(a) Ring Interval Graph **(b) Removal of an edge**

Figure 4.5: Ring Interval Graph

(3), we visit v_4 with $id(v_4) = y + 1$ (message (4)). Then the cycle goes back to node v_1. After the assignment, node v knows the followings. All the nodes behind node v_3, including v_3, have id value from $x + 1$ to y, and all the nodes behind node v_4, including v_4, have id value from $y+1$ to z. The rest of id values are behind node v_1. We develop this idea and build a graph structure with this property in the following section.

4.2 Ring Interval Graph

In this section, I define a tree-like graph structure *ring interval graph* (RIG) which enables a searching scheme faster than ring-based TD-DHT. Unlike ring, RIG provides more choices on choosing the closest neighbor by partitioning the hashing space into non-overlapping intervals and mapping intervals to neighbor nodes.

In the following descriptions, for ease of representation, I convert the hashing

space of $[0, 1)$ into a larger space $[0, n)$ so that we can use integer values for node identities. One can easily convert to the $[0, 1)$ hash space by dividing all identity values by n. When I say node i, it denotes either node name or node identity depending on the context.

4.2.1 Definition Of Ring Interval Graph

Given a set of n node identities $N = \{0, 1, \ldots, n-1\}$, let the *ring* be a cyclic chain of nodes starting from 0 and 1, 2, ..., $n-1$ and returning to 0, as in Figure 4.2-(b). A ***ring interval*** is a set of node identities in a connected segment of the ring. For example, given a ring of Figure 4.2-(b), $\{10, 11, 0, 1\}$ is a valid ring interval. We denote a ring interval by $[a, b]$ where $a, b \in \{0, 1, \ldots, n-1\}o$. If $a < b$, $[a, b] = \{a, a+1, \ldots, b-1, b\}$, if $a > b$, $[a, b] = \{a, a+1, \ldots, n-1, 0, 1, 2, \ldots, b-1, b\}$, and if $a = b$, $[a, b] = \{a\}$. Note that I use integer values for node identities only for simple representation but one can use any real numbers in $[0, n)$ for node identities and the following discussion is still valid.

Given a network graph $G = (V, E)$ where $|V| = n$, a ***Ring Interval Graph*** (RIG) is an acyclic undirected subgraph $G_{rig} = (V, E_{rig})$ with an identity assignment mapping $id : V \rightarrow \{0, \ldots, n-1\}$ such that any *one-cut* (a removal of one edge) partitions the graph into two distinct ring intervals. Figure 4.5-(a) shows an example of a ring interval graph with 12 nodes. As seen in Figure 4.5-(b), the removal of edge $(1, 4)$ partitions the ring into two ring intervals $[4, 7]$ and $[8, 3]$.

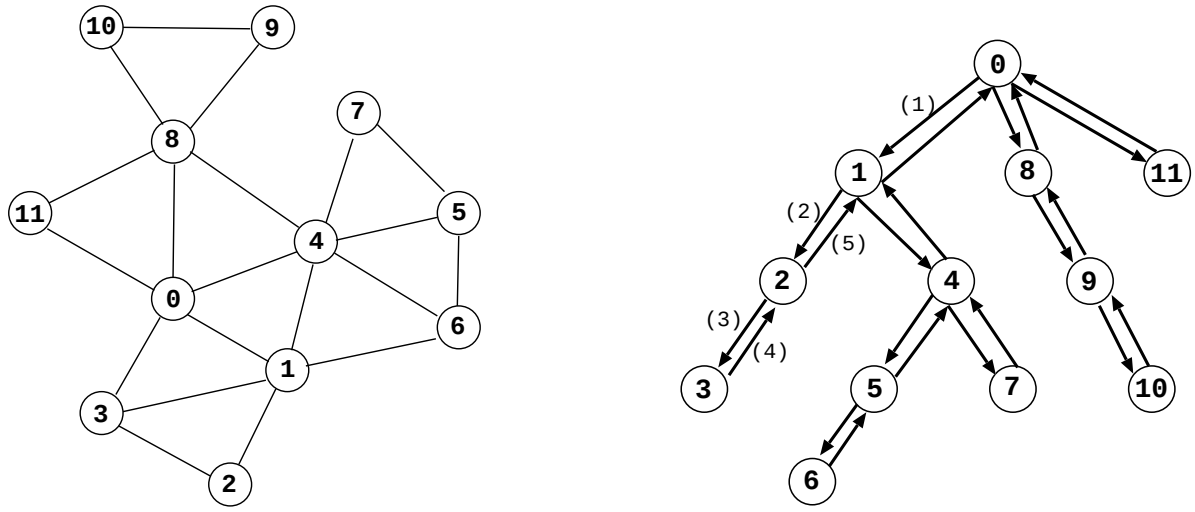
Suppose a node v_0 has d neighbor nodes $v_1, v_2, \ldots, v_d$ and denote the ring

interval formed by nodes behind a neighbor v_i by I_i. For example, in Figure 4.5-(a), from the node 1's point of view, the ring interval behind node 4 is $[4, 7]$. In this way, node v_0 can partition the ring space $\{0, 1, \ldots, n-1\}$ into $d+1$ ring intervals $\{I_0, I_1, \ldots, I_d\}$ where $I_0 = \{v_0\}$ and I_i is associated with v_i. Such a RIG structure provides a interval-based routing mechanism such that, given a query for a target identity i, each node knows the next node toward the destination by choosing a neighbor node v associated with a ring interval I containing i.

4.2.2 Construction Of Ring Interval Graph

Given a spanning tree of a graph, one can easily construct a RIG by a depth first search (DFS) traversal and assigning a value in the increasing order to each node at the first visit. Figure 4.6 shows a graph and a spanning tree with DFS trace. During a DFS traversal, we visit each node multiple times. For example, in Figure 4.6-(b), node 1 is visited three times, first from node 0 and second and third from 2 and 4, respectively. We keep a counter, initially set to 0, and assign the counter's value to a node when we visit the node for the first time, and the counter is increased at each assignment. Figure 4.6-(b) shows an example construction of RIG on a given spanning tree. Starting node 0, we traverse along the edges $(1), (2), \ldots,$ and assign integer values in the visiting order. The resulting spanning tree with assigned id values become a RIG.

Construction over different spanning trees creates different RIGs. As shown by simulation results, TD-DHT schemes based on different RIG provides different



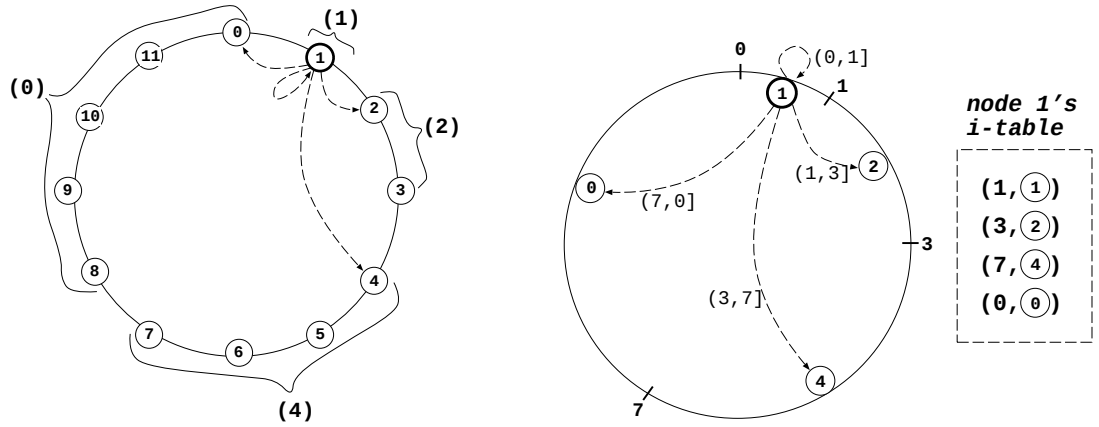
(a) Network Graph **(b) DFS tree**

Figure 4.6: Construction of RIG through DFS on the network graph

performances. In this work, I study two kinds of constructions, depth first spanning tree and breadth first spanning tree. RIG construction with depth first spanning tree is easy to implement in a distributed setting as in wireless networks, and we can combine construction of DFS spanning tree with the construction of RIG. Breadth first searching tree is used for broadcasting or multi-casting in wireless networks. Given a BFS tree, one can easily construct RIG by DFS traversal on the tree.

4.3 Hashing And Search

In this section, I describe the construction of RIG-based TD-DHT structure, called Rigs. We assume that the construction of Rigs can be based on either DFS spanning tree or BFS spanning tree. We call DFS-based scheme Rigs-Dfs and BFS-based scheme Rigs-Bfs.



(a) Discrete partition **(b) Continuous partition**

Figure 4.7: Construction of RIG for continuous intervals

4.3.1 Hashing With Rigs

Recall the ring of $[0, n)$ where n nodes are evenly located. We want to hash objects on the same ring space and assign an object-holder for each object to the successive node of the object. With a ring interval graph, one can find an object-holder by forwarding to a neighbor who is mapped to an interval containing the object-holder's id. In the following, I describe the construction of RIG-based TD-DHT.

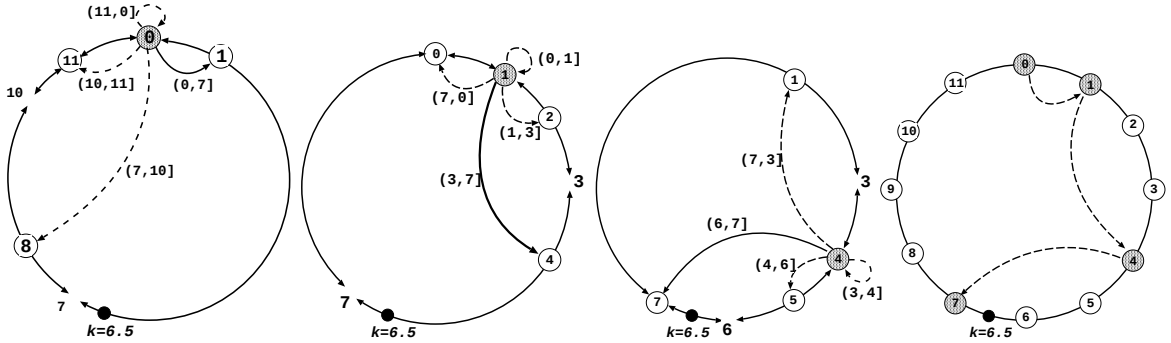
To handle objects properly, we have to convert the ring interval into the continuous real interval. Suppose we have a node with $d + 1$ intervals $I_0 < I_1 < \ldots < I_d$ and I_i is associated with node v_i. Let b_i be the largest of the interval I_i, then we partition the hash space $[0, n)$ into $d + 1$ continuous ring intervals as $(b_0, b_1], (b_1, b_2], \ldots, (b_{d-1}, b_d], (b_d, b_0]$ and associate with nodes $v_1, v_2, \ldots, v_d, v_0$, respectively. For example, consider node 1 in Figure 4.5, then ring intervals $[2, 3], [4, 7], [8, 0], [1, 1]$ are associated with nodes $2, 4, 0, 1$, respectively. Then, the new partition becomes $(3, 7], (7, 0], (0, 1], (1, 3]$ with associated nodes $4, 0, 1, 2$ (see Figure 4.7).

Algorithm 2 Construction of RIG and distributed hash table

```
1: {v: myself, w: parent in DFS, u: v's neighbor}
2: {V: set of visited nodes}
3: {add(x, v): add (x, v) into interval table}
4: {Execute the following when v receives (x, w) }
5: if v is the starting node then
6:    id(v) := x
7:    V := {v}
8: else
9:    add(x, w)
10:   id(v) := x + 1
11:   V := V ∪ {v}
12: end if
13: add(id(v), v)
14: y := id(v)
15: for all u ∈ Nb(v) ∧ u ∉ V do
16:   send (id(v), v) to u
17:   wait until u sends back (y, u)
18:   add(y, u)
19: end for
20: reply with (y, v) to w
```

Let node v maintain an ***interval table***, or i-table, which is a sorted list of value-node pairs, i.e., i-table $= ((b_1, v_1), (b_2, v_2), \ldots, (b_d, v_d), (b_0, v_0))$. With the above example, node 1's i-table becomes $((7, (4)), (0, (0)), (1, (1)), (3, (2))$.

Algorithm 2 illustrates how we can construct RIG and generate i-tables. Note that, if $Nb(v)$ is on the network graph G, it generates a DFS-tree based RIG and, if $Nb(v)$ is on the given BFS spanning tree, it generates BFS-based RIG. Each node v runs this algorithm when v receives a RIG packet from a node w, say node 1 receives from node 0, or when v is the starting node, say node 0 in the example. Upon receiving a value x, v increases x by one (except starting node) and set the new value as its id value (line 4–11). The set of visiting node V now contains v. Also v adds (x, w) into its interval table as well as $(id(v), v)$ (line 12). Then, v sends



(a) $v = 0$ (b) $v = 1$ (c) $v = 4$ (d) global

Figure 4.8: Range-based forwarding by RIG

a RIG packet $(id(v), v)$ to each unvisited neighbor u and wait for reply value y and add (y, u) into the interval table before moving to next neighbor node (line 14–18). v also replies to the previous node w with the last value from its children nodes or its own id value if there is no child (line 13 and 19).

4.3.2 Shorted Interval Forwarding

Given an object k, each node forwards to a neighbor node associated with a ring interval containing k. In Figure 4.5-(a), Suppose node 0 queries for object 6.5. By the RIG-construction algorithm, the interval table of node 0 has $\{(0,(0)),(7,(1)),(10,(8)),\ (11,(11))\}$. Figure 4.8(a) depicts node 0's i-table. According to node 0's i-table, object 6.5 belongs to the interval $(0, 7]$, so the query is forwarded to node 1. In turn, 6.5 belongs to the interval $(3, 7]$ in node 1's i-table and, thus, node 1 forwards the query to node 4 (see Figure 4.8(b)). Similarly, node 4 in turn forwards to node 7 (Figure 4.8(c)) which is the object-holder of the object. Figure 4.8 shows a global view of the search path from node 0 to node 7.

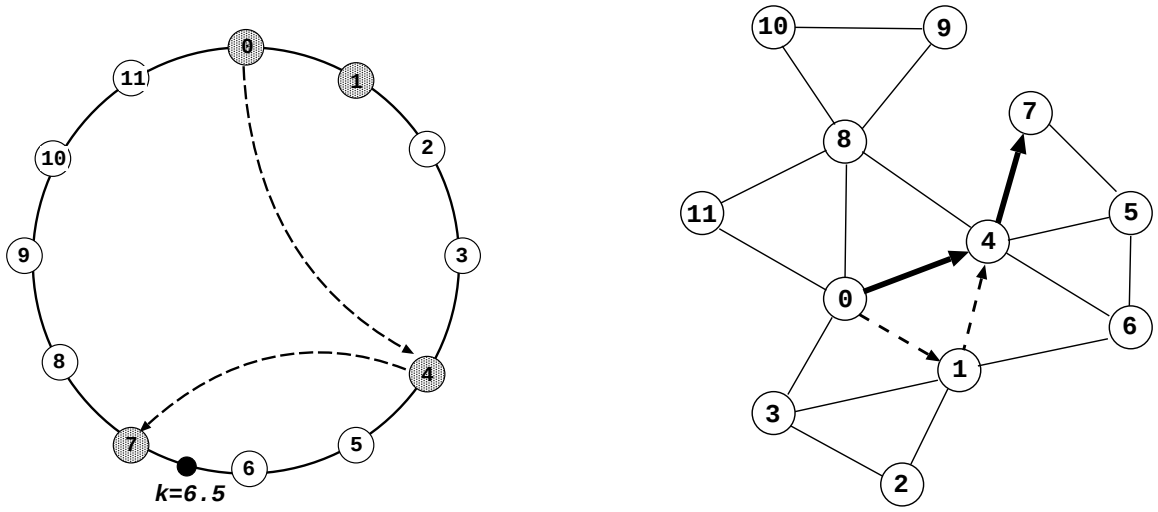


Figure 4.9: RIG searching with a shortcut from node 0 to node 4

As in ring-based schemes, RIG-based TD-DHT also can use shortcuts to improve the search performance by choosing neighbors not in the RIG but provides better route to the destination. In the previous example, the query from node 0 travels over the path $0, 1, 4, 7$ and node 0 chose node 1 because, from node 0's point of view, the query must visit node 1 to reach any node in the interval $(0, 7)$. However, in Figure 4.6(a), node 0 has other neighbors 3 and 4 that are not on the RIG (Figure 4.5(a)). If node 0 knows that node 4 has the interval $(6, 7]$ in its i-table, node 0 can directly send the query to node 4 instead of 1 and achieve a shorter search path of length 2 instead of 3 (Figure 4.9).

Suppose neighbor nodes exchange their i-table, especially the interval partitions. For example, node 4 sends neighbors only end points of its intervals: $\{3, 4, 6, 7\}$. Then, a neighbor node 0 knows that node 4 knows where to forward for intervals $\{(3, 4], (4, 6], (6, 7], (7, 3]\}$. When forwarding for object k, node v first finds

an interval I from its i-table such that $k \in I$. Before forwarding to the node mapped to I, node v checks the set of neighbor's intervals and find the shortest interval I' such that $k \in I'$ and $I' \subset I$ but $I' \neq I$. We call this the shortest interval forwarding. For example, for object 6.5, node 0 chooses the interval $I = (0, 7]$ from its i-table. But a neighbor node 4 has an interval $I' = (6, 7]$ which contains the object and also a proper subset of I. Therefore, node 0 finds that forwarding to node 4 instead of 1 is a shortcut toward the destination.

Theorem 4.3.1 states that the shortest interval forwarding always chooses the closest neighbor to the destination among neighbors.

Theorem 4.3.1 (Monotonic property of shortest interval forwarding) *Given an object k, let v and v' be two distinct nodes. Let $k \in I \in itable(v)$ and $k \in I' \in itable(v')$.*

If $I' \subset I$ and $I \neq I'$, then node v' is closer to the destination than node v.

Proof For node v, let node u be the leading node to the interval I. If $v = u$, v is the destination and we are done. Assume $v \neq u$. Then, edge (v, u) partitions the network into I and $R - I$ where $R = [0, n)$. Suppose $v' \in R - I$. Then, v should be in the path from v' to k which means that $v \in I'$, thus $v \in I$. But $v \notin I$ because of the definition of I, thus contradiction. Therefore, $v' \in I$.

Now we claim that v' should be in the path from v to k. Suppose not. Let P be the path from v to k and P' from v' to k. Let $P = (v, \ldots, w, \ldots, k)$ and $P' = (v', u', \ldots, w, \ldots k)$ where $w \in P \cap P'$. The removal of edge (v', u') partitions the network into two and one of them is I'. Since $k \in I'$, $w \in I'$, and w and v

Algorithm 3 Shortest Interval Forwarding

```
1: {Nb(v): v and its neighbor nodes }
2: {Node v runs this when receiving query for k}
3: min := MAX
4: for all u ∈ Nb(v) do
5:    for all  I' ∈ itable(u)  do
6:       if  k ∈ I' and |I'| < min  then
7:          next := u
8:          min := |I'|
9:       end if
10:   end for
11: end for
12: forward query to next.
```

is connected through path P, $v \in I'$, thus $v \in I$. But $v \notin I$, a contradiction. Therefore, v' should be in the path from v to k. ∎

Algorithm 4 describes the shortest interval forwarding algorithm. Unlike descriptions above, node v simply checks all i-tables of neighbor nodes and forward the query to the node with the shortest interval with k. Note that, RIG structure guarantees that every node can find an interval that contains any object k from neighbor's intervals.

4.3.3 Replication

Given a replication number r and an object k, we generate additional $r-1$ virtual object ids that evenly spread in the id space. If the id space is $[0, n)$, the r virtual object ids are $\{k + \frac{i}{r}n \mid i = 0, 1, 2, \ldots, r-1\}$. Since each node can calculate object ids of replicated objects, the searching algorithm intensionally chooses one of them which is the closest among replications as follows. For each virtual object, the node run Algorithm 4 to get the shortest interval neighbor. Choose the neighbor

Algorithm 4 Shortest Interval Forwarding with Replications

```
1: {r: replication number}
2: {Nb(v): v and its neighbor nodes }
3: {Node v runs this when receiving query for k}
4: min := MAX
5: totalmin := MAX
6: for all virtual object k' do
7:    for all u ∈ Nb(v) do
8:       for all I' ∈ itable(u) do
9:          if k ∈ I' and |I'| < min then
10:            next := u
11:            min := |I'|
12:         end if
13:      end for
14:   end for
15:   if min < totalmin then
16:      totalnext := next
17:      totalmin := min
18:   end if
19: end for
20: forward query to totalnext.
```

which has the shortest interval among chosen neighbor for each virtual object.

Chapter 5

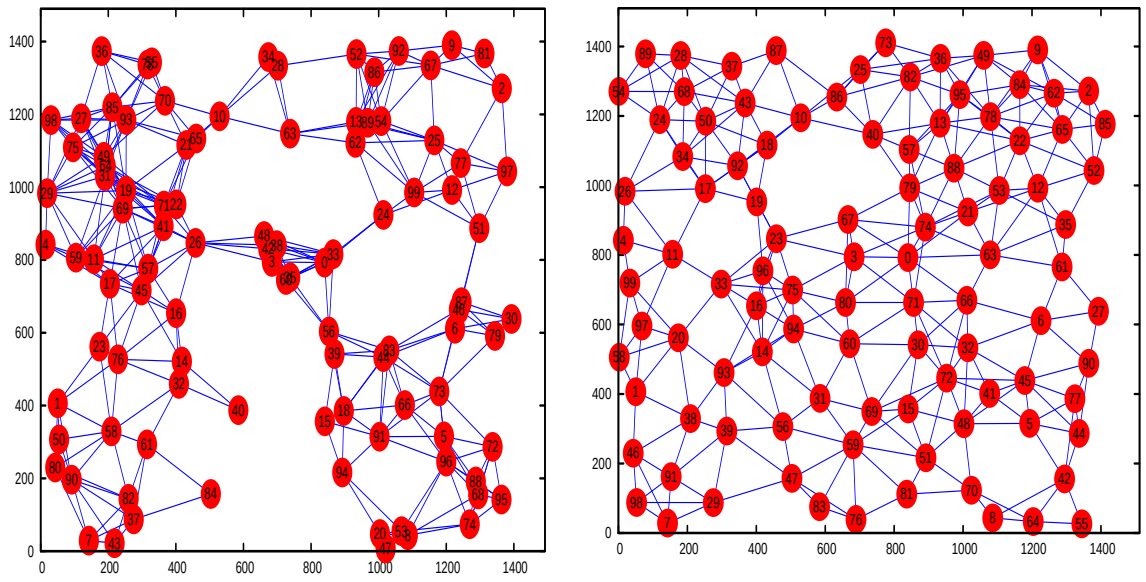
Simulation

By packet-level simulations, I evaluate the performance of the proposed algorithms VALLEY-WALK$_{LM}$, VALLEY-WALK$_{KD}$, and RIGS and compare with that of existing methods, such as Chord, LMS, and Sozer's scheme. To evaluate P2P lookup mechanisms, I simulate the user authentication scenarios in wireless mesh networks. Network size, query rate, and replication factor are varied between simulations. The simulation results show that proposed schemes not only outperform other schemes but also achieve close-to-optimal performances.

5.1 Methodology

I used a packet-level network simulator ns2 [71] for the simulation. Various network topologies are generated as follows. Define the unit node density as the case when 50 nodes are deployed in $1000 \times 1000m^2$ square area. Given the number of node and node density, the topology area is recalculated accordingly. For example, given 50 nodes with density 2.0, the topology area is reduced to one half. The transmission range of each node is fixed to $250m$.

I used two topologies; random geometric graph and mesh network graph. Random geometric graph is generated by placing nodes uniformly at random and adding an edge between two nodes if they are within the transmission range. Mesh net-



(a) Random geometric graph **(b) Mesh network graph**

Figure 5.1: Topologies

work graph, however, adds one constraint on random geometric graph; any two nodes cannot be closer than a given minimum node distance (e.g., 100 m). This topology simulates the real mesh network which enforces enough separation between mesh nodes to minimize interference and control the load balance. Figure 5.1 shows examples of the topologies with network size of 100 and unit density (d=1).

Given a network topology, during a period of 100 seconds, unlimited number of users arrive at randomly chosen nodes which, then, perform authentication procedures. The user arrival time follows the Poisson distribution with a mean of given user arrival rate (user/second) from 1 to 80. For each user, a user key is chosen uniformly at random from $[0, 1)$ and the mesh node then discovers a key-holder using the given P2P lookup algorithm. Once the query packet arrives at a key-holder, the authentication protocol starts. I assume that underlying routing protocol can

Direction	Content	Size (B)
$U \rightarrow S$	Acc-Request/EAP-Response-ID	127
$U \leftarrow S$	Acc-Challenge/EAP-Request/PSK-1	99
$U \rightarrow S$	Acc-Request/EAP-Response/PSK-2	181
$U \leftarrow S$	Acc-Challenge/EAP-Request/PSK-3	101
$U \rightarrow S$	Acc-Request/EAP-Response/PSK-4	135
$U \leftarrow S$	Acc-Accept/EAP-Success	46

Table 5.1: Message Exchange for EAP-PSK

effectively deliver the protocol packets between the serving node and the key-holder node. DSDV (Destination-Sequenced Distance Vector) is used in this experimental study. Note that proposed algorithms are independent of underlying routing protocols because each forwarding is mostly to one hop neighbor. Even after the 100 second period, the simulation waits for the started authentications to finish up to 300 seconds. An additional 30 seconds are spent for initialization such as neighbor discovery or routing stabilization.

For the authentication protocol, EAP-PSK (Pre-Shared Key Extensible Authentication Protocol, RFC 4764 [72]) is chosen because the protocol is a standardized practical shared-key authentication method. EAP [73] and AAA [74] (Radius [75] or Diameter [76]) based authentication framework is popular in wireless network because the conventional setting of three entities (user, authenticator, and back-end server) fits to the wireless network services [77] [78]. Table 5.1 shows the message exchange and message size of EAP-PSK used for the simulations.

Table 5.2 summarizes the simulation parameters used and their values.

Parameter	Values
number of nodes	$50 \sim 200$
network density	$1.0 \sim 3.0$
user arrival rate	$1 \sim 80$
replication number	$1 \sim 20$
random seed	$11 \sim 30$

Table 5.2: Simulation Parameters

5.1.1 Algorithms

I implemented and simulated the following algorithms and compared their performances. Without specific explanations, neighbors or neighborhood means a set of nodes reachable by direct wireless communication, which are often called one-hop neighbors.

RANDOM

With random walk (RandomWalk) method, each node forwards the query to one of unvisited neighbor nodes uniformly at random. We can prevent loops by either recording (or bloom filtering) visited nodes in that specific walk in the query packet or having nodes keep recent forwards in their caches. In this simulations, I record visited nodes in the query packet and nodes exclude those nodes for forwarding unless there are no unvisited nodes left. To maximize service availability, no expiration (or time to live) of RandomWalk packets is set, i.e, the query is forwarded until it reaches the destination. When replicate, I place keys at nodes uniformly at random.

OPTIMAL

The optimal solution of the key discovery problem can be instantiated by the following scheme; for any key, distributed uniformly at random, the serving node *magically knows* who is the closest key-holder and starts the authentication to it. The hop-stretch of this scheme is one, i.e., it always finds the key-holder through the shortest path. We call this method OPTIMAL because no other algorithm can perform better. However, OPTIMAL method is impractical in the real world. The OPTIMAL query requires global knowledge of the topology and the locations of each key. For a decentralized protocol, each node should flood the network for each authentication request. Although the method is impractical, we experiment with OPTIMAL method to evaluate proposed algorithms and examine how their performances are close to optimal.

LMS

The structure of the hash space in LMS is similar with VALLEY-WALK; unit length ring to which all nodes and keys are hashed. The key distance of LMS is not directional; the shortest distance between the key and the node regardless of the direction. To expedite the search, LMS suggests the use of 2-hop neighborhood for forwarding rather than one hop. In this simulation, however, I use one-hop neighbor because the use of 2-hop neighborhood degrades the performance instead of improving in multi-hop wireless networks. This is because of the fact that the increase of hop distance has more negative impacts on the performance than wired

networks.

For each authentication request, the serving node initiates a random walk with given initial time-to-live (TTL). If it fails to find a key-holder after random walk, it continues with a deterministic walk forwards the request to a neighbor with smallest key distance. When it reaches a local minimum without a key, the local minimum node reports the serving node of the failure. Then, the serving node starts another random walk with a doubled TTL. I use TTL of three, as suggested by the authors of LMS [12].

Chord

This is the original Chord scheme. I assume that every node somehow knows its finger table from the beginning of the simulations. Please see Section 2.3.1 for more detail.

Sozer's

This scheme is an implementation of Sozer's wireless file sharing scheme described in Section 2.6.

Valley-Walk$_{LM}$

This scheme is explained in Section 3.3. After network deployment and independently choosing nodes' *id*'s, we can decide which node becomes a local minimum. Each query is forwarded by Valley-Walk until it reaches a local minimum (see

Algorithm 1). When the expected number of local minima ($\frac{n}{d+1}$) is different from the replication number r, we probabilistically adjust the number of local minima by increasing (or decreasing) the minimum (or maximum) node degree to decrease (or increase) the replication number (see Section 3.3.2).

VALLEY-WALK$_{KD}$

This scheme is explained in Section 3.4. When a user arrives and has key k, the r mesh nodes closest to the key become key-holders.

RIGS

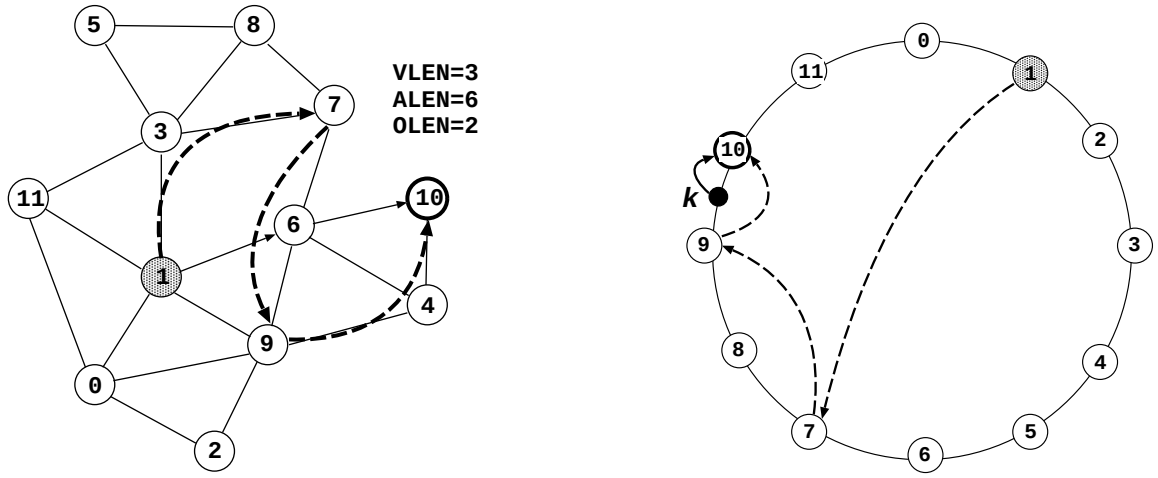
These schemes are explained in Chapter 4. Only BFS based RIGS algorithm is evaluated.

5.1.2 Metric

In this section, I define various metrics used for performance evaluation of key discovery schemes.

Search Performance

The search performance is a direct measure of how much a key discovery algorithm is effective on finding keys. It is easy to express the search performance by the cost of the search or the overhead of the search. We first define four different metrics that somehow reflect the cost of the search: virtual search length, actual



(a) Chord in Graph **(b) Chord on ring**

Figure 5.2: Chord and its stretches

search length, shortest search length, and optimal search length.

Virtual search length (VLEN) is the number of application-level forwarding over the high-level DHT structure. Figure 5.2 shows an example search by Chord where node 1 queries for k and finds node 10 as a key-holder. By the Chord algorithm, the query follows the route denoted by dashed arrows in Figure 5.2(a) and its VLEN is three, the hop count in the conceptual Chord Ring in Figure 5.2(b). Most structured DHT schemes claim the worst-case VLEN of $O(\log N)$ but in multi-hop wireless, VLEN alone fails to correctly express the search performance.

The *actual length* (ALEN) is the number of hops the query packet actually traveled on the network topology. In Figure 5.2, the query packet traveled along the path of $(1, 3, 7, 6, 9, 4, 10)$, thus ALEN is six.

The *shortest search length* (SLEN) is the shortest hop count on the physical topology from the requesting node to the destination. In Figure 5.2, although the

query packet traveled 6 hops, the shortest distance between node 1 and node 10 is only two through path $(1, 6, 10)$. If there exists more than one replicated keys, SLEN reflects how well the algorithm finds a close key copies, or "local targets".

The *optimal search length* (OLEN) is the search length of ideal algorithm which finds a target which is the closest among the replications and the query packet travels along the shortest path from the requesting node to the destination. For example, suppose there is another replication at node 11 in Figure 5.2. Then, the optimal search length should be 1, but the algorithm missed the closest replication and finds one hop farther replication at node 10 costing 6 hops. This metric is meaningful for evaluating simulated schemes to see how much they perform close to the optimal solution. With the OPTIMAL algorithm in the simulations, the metric of VLEN, ALEN, SLEN, and OLEN are all equal.

Now we define derived metrics from aforementioned metrics. The *search-hop-stretch* is the ratio of the ALEN to OLEN ($\frac{\text{ALEN}}{\text{OLEN}}$), which represents the overall performance of the algorithm compared to the optimal solution. In the example, assuming we have another replication at node 11, search-hop-stretch is 6 (6/1).

The *detour-stretch* is the ratio of the ALEN to SLEN ($\frac{\text{ALEN}}{\text{SLEN}}$), which reflects how much the actual searching path detoured from the shortest path which should be taken by an ideal algorithm. This metric shows how much the search path of the algorithm follows a straight line to the destination. Note that this metric do not reflect how well the algorithm finds a target close enough to the requester. In the example, detour-stretch is 3 (6/2).

The *locality-stretch* is the ratio of the SLEN to OLEN ($\frac{\text{SLEN}}{\text{OLEN}}$), which reflects

how close the algorithm chooses the replication compared to the optimal solution. Locality-stretch close to one means that the algorithm successfully finds the closest replication. In the example, locality-stretch is 2 (2/1).

The *virtual-hop-stretch* is the ratio of ALEN to VLEN ($\frac{\text{ALEN}}{\text{VLEN}}$), which reflects how each virtual link of the searching scheme *hides* the underlying hop counts in the real world. In our example, virtual-hop-stretch is 2 (6/3), meaning that on average, each virtual hop costs two hops on the actual topology. Note that most proposed schemes have the virtual-hop-stretch of one. Only VALLEY-WALK$_{LM}$ can have more than one virtual-hop-stretch if it expands the neighborhood to two hop neighbors.

Since the average search performance cannot show the distribution of the performance, I include the *tail probability* of the search length, the probability of having search lengths (ALEN) at least given search lengths. For example, if the search tail probability is 0.2 for the search length of 4, it means that only 20% of all queries take more than 3 hops.

Authentication Performance

The authentication performance evaluates the feasibility of the given key discovery algorithms when applied to the distributed authentication in wireless mesh networks. The authentication performance is affected by searching performance and also by the balance of the authentication traffic.

We measure the authentication performance by authentication success ratio and authentication latency. An authentication is successful if the last message of the

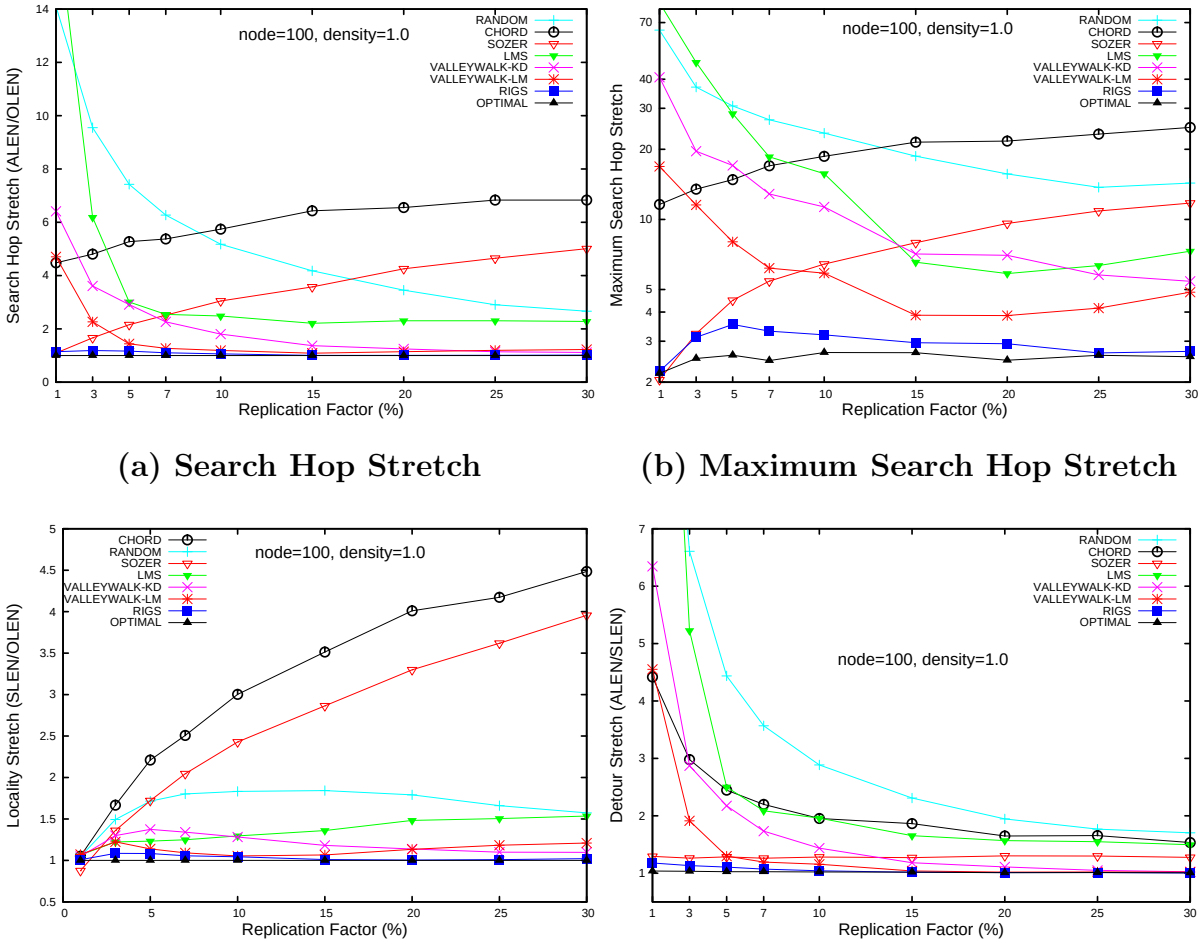
authentication protocol is delivered. The *authentication success ratio* is the ratio of the number of successful authentications to the number of initiated authentication requests. An authentication fails if an authentication message is lost due to interference or packet drops. In this simulations, failed authentication do not retry.

The *authentication latency* is the time between the first message of the authentication protocol sent and the last message received. The transmission delay due to contention for channel access and queuing delay due to congestion can cause longer authentication latency. The high network density and high node degree can increase contention related delay and the unbalanced selection of key holders can generate traffic congestion.

The *search delay* is the time between the start of the query and the discovery of a key-holder. Long search delay can cause a long authentication latency.

5.2 Results For Searching Performance

In this section, I evaluate the search performance of the proposed schemes by running several sets of simulations varying parameters such as replication factor, network size, and network density. I use the metrics introduced in Section 5.1.2. To measure only searching performance, I fix the average user arrival rate as 1 per second to minimize the impact of interference and traffic congestion on the key discovery process. For simulations with increasing arrival rates, please refer to Section 5.3.



(a) Search Hop Stretch **(b) Maximum Search Hop Stretch**

(c) Locality Stretch **(d) Detour Stretch**

Figure 5.3: Search performance by replication factors - (1)

5.2.1 By Replications

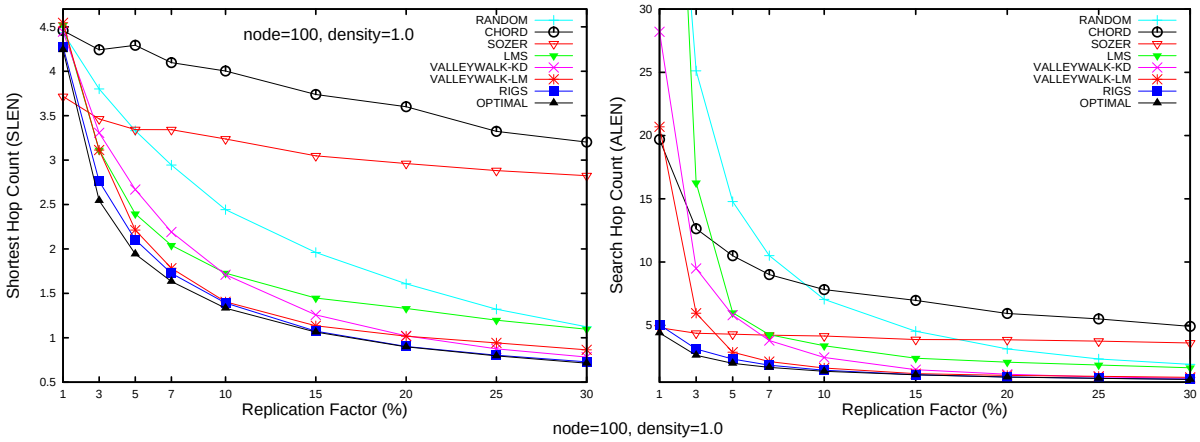
The replication factor is the ratio of replications to the network size. In this section, I vary the replication factor as $\{1\%, 3\%, 5\%, 7\%, 10\%, 15\%, 20\%, 25\%, 30\%\}$ with network size of 100 and density of 1.0.

Figure 5.3 shows the overall searching performances of each schemes. Figure 5.3-(a) shows the search-hop-stretch metric. This figure contrasts CHORD and SOZER's against other schemes; although the replications factor increases, the

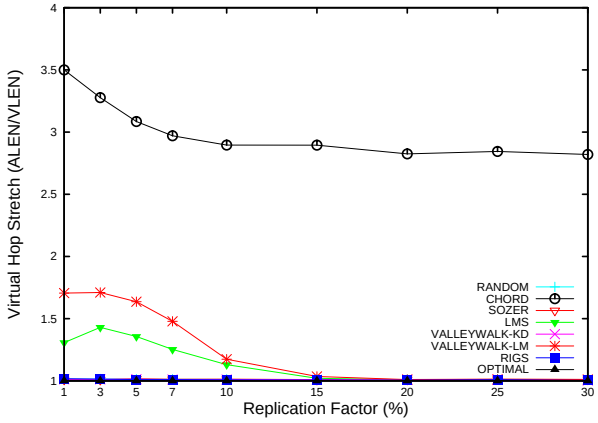
search-hop-stretch, the searching overhead compared with that of Optimal, of Chord and Sozer's stays high. That is, replication does not help with these schemes. Chord's routing entries fail to reflect the abundant replications around. The main problem of Sozer's is the biased hashing and most objects are stored in a small number of nodes, and this accounts for the limitation of replication in the scheme. Despite improving with more replications to some extent, RandomWalk and Lms also fail to keep reflecting the replications.

Figure 5.3-(a) also shows that Valley-Walk-based schemes do not perform well with a small number of replications. The performance of Valley-Walk is limited with small number of replications because Valley-Walk$_{LM}$ does not expand its neighborhood more than 2-hops and there become many empty local minima with scarce replication. The figure also confirms that Rigs scheme performs close to Optimal.

Figure 5.3-(b) shows the maximum search-hop-stretch with its y-axis in log scale for clear representation of differences between proposed schemes. In this figure, proposed schemes perform well in the order of Rigs, Valley-Walk$_{LM}$, and Valley-Walk$_{KD}$. Throughout the simulations, this trend stays the same except that the performance of Valley-Walk$_{LM}$ looks limited with large number of replications because, after the replication number exceeds its natural number of local minima, the change of hitting a key-holder becomes that of random selection. In this simulations, the number of natural local minima, the number of local minima when no neighborhood adjustment is performed, is around 12. Note that, in the figure, Rigs always keeps the variance of the search-hop-stretch low.



(a) Shortest Hop Count **(b) Search Hop Count**



(c) Virtual Hop Stretch

Figure 5.4: Search performance by replication factors - (2)

Figure 5.3-(c) shows how close key holders (i.e., local) we can find compared to that of OPTIMAL. After 12, VALLEY-WALK$_{LM}$ begins to find distant key-holders. Detour stretch (Figure 5.3-(d)) shows how straight the query approaches to the destination. As shown in the figure, RIGS always finds the target along the shortest path.

Figure 5.4-(a) and Figure 5.4-(b) shows SLEN and ALEN, respectively, which shows the similar trends as in the previous figures. Virtual-hop-stretch of Fig-

ure 5.4-(c) shows how many hops in the real world each logical forwarding costs. The figure shows that proposed schemes only forwards to 1-hop neighbors, except VALLEY-WALK$_{LM}$ with low replication factors, which may have logical neighbors that is 2-hop away. Note that with CHORD, every forwarding to neighbor costs more than 3 hops.

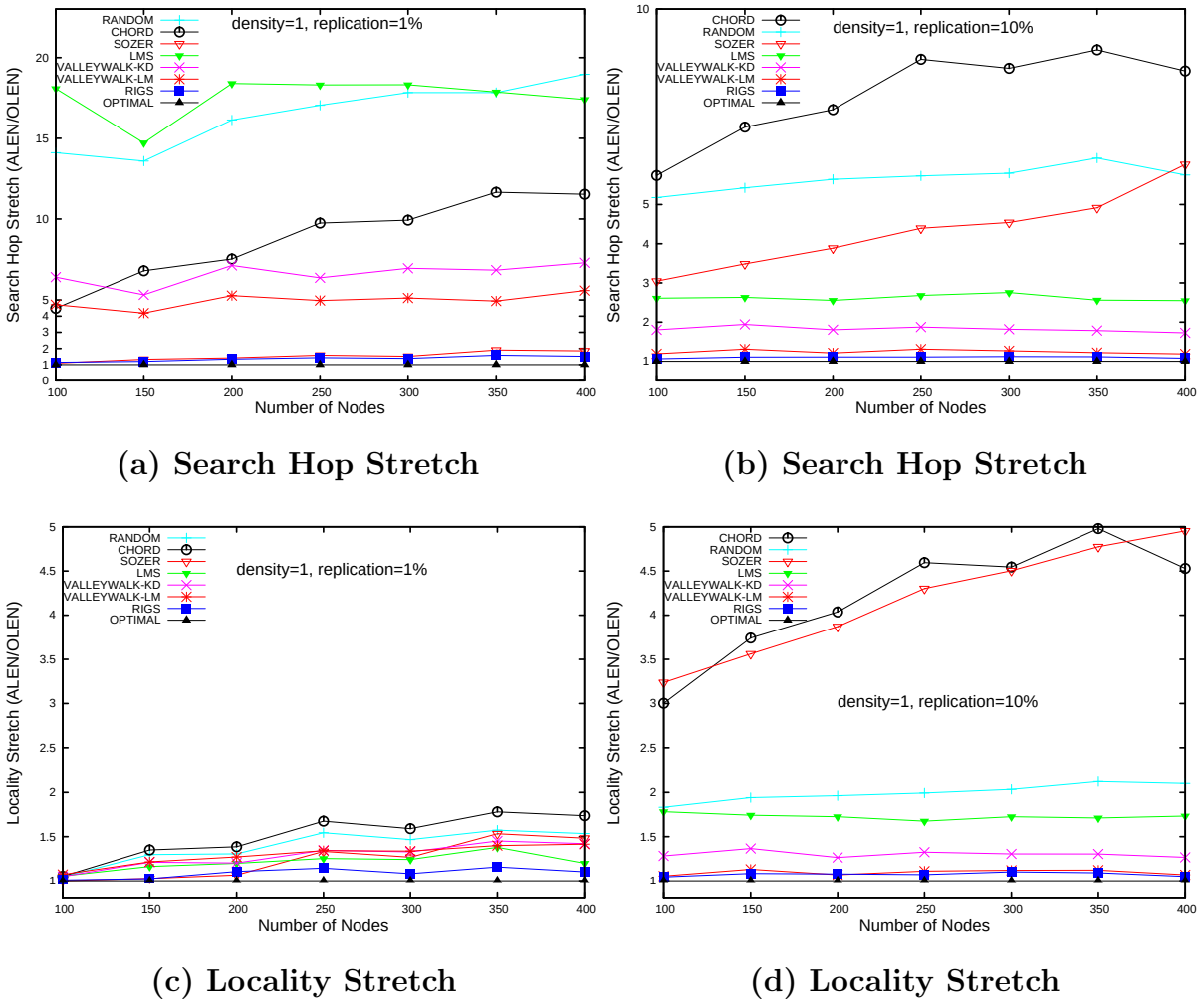
From the simulations, it is clear that CHORD schemes can be outperformed by the simple RANDOMWALK as the number of replications increase. While RANDOMWALK naturally benefits the higher fraction of key-holders, the logical links in CHORD ignore that effect and blindly forwards to the virtual neighbors distant more than 3 hops, missing nearby targets.

5.2.2 By Network Size

In this section, I vary the network size from 100 to 400 increased by 50, with fixed node density of 1, and the replication factors of 1% or 10%.

Figure 5.5 shows that proposed schemes scale well with increasing number of nodes. The locality-stretch in Figure 5.5-(c) shows that, even if the replication is as low as 1%, as the number of nodes increases, schemes find different destinations. For example, when network size is 400, there are four replications and RIGS chooses the close enough ones while others fail to choose the best.

Figure 5.6 also shows the same trend as Figure 5.6. Although CHORD performs as good as proposed schemes with a low replication factor, its performance degrades as the network size grows while proposed scheme stays the same. Figure 5.7 and



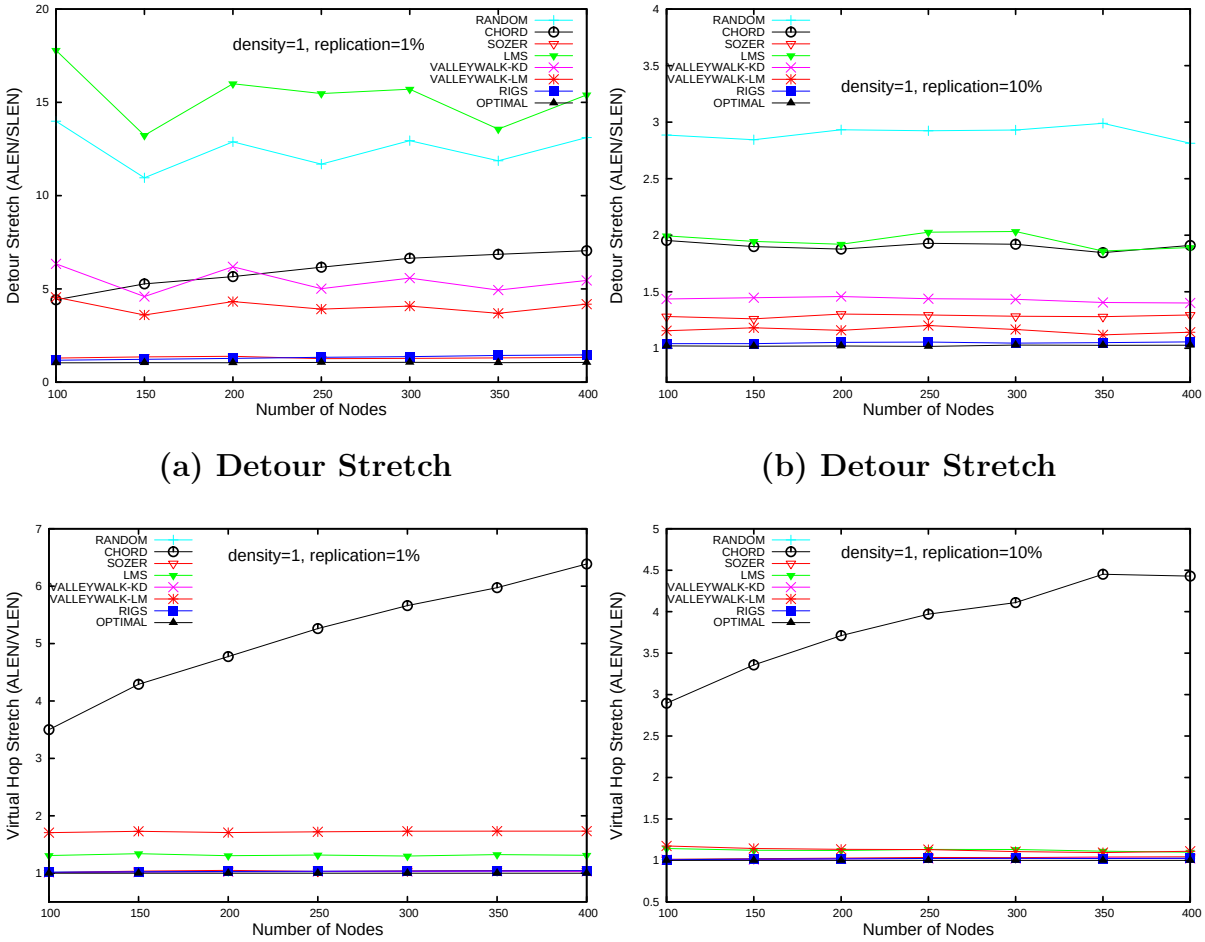
(a) Search Hop Stretch **(b) Search Hop Stretch**

(c) Locality Stretch **(d) Locality Stretch**

Figure 5.5: Search performance by network size -(1)

5.8-(a),(b) simply reflects the trend of Figure 5.5. Figure 5.8-(c),(d) shows that RIGS maintains even the maximum search hop count as low as OPTIMAL.

In this section, I showed that proposed schemes are scalable for the network size and RIGS based schemes perform close to OPTIMAL even for a large network size, with low variances.



(a) Detour Stretch **(b) Detour Stretch**

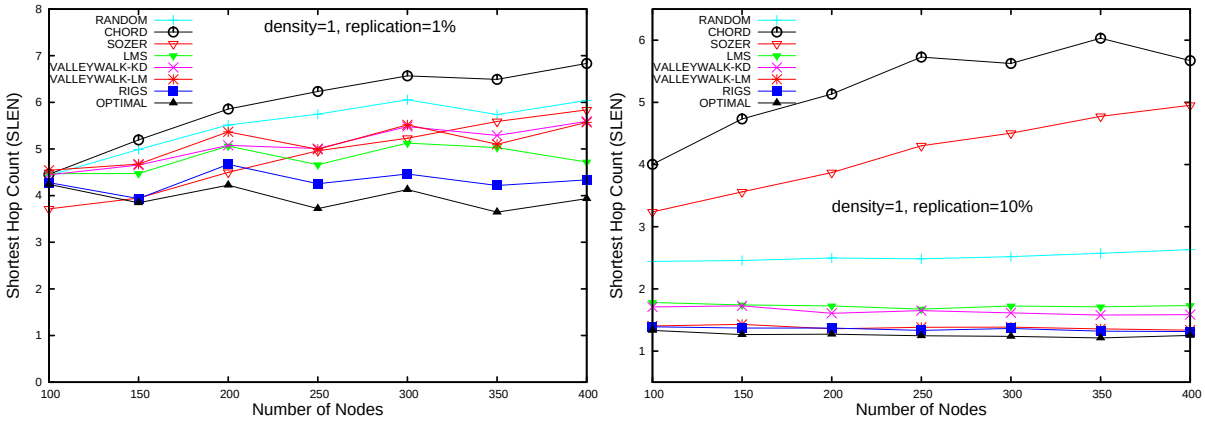
(c) Virtual Hop Stretch **(d) Virtual Hop Stretch**

Figure 5.6: Search performance by network size -(2)

5.2.3 By Node Density

In this simulations, I vary the network density from 1 to 3 by 0.5 with fixed network size of 100 and replication factors of 1% or 10%. Density of 1 represents a node density when there are 50 nodes inside 1000 by 1000 square area with 250 transmission range. Other densities are proportional from the density of one.

Figure 5.9 through Figure 5.12 shows various metric on the searching performance by varying densities. As the network becomes dense, the number of degree



(c) Shortest Hop Count **(d) Shortest Hop Count**

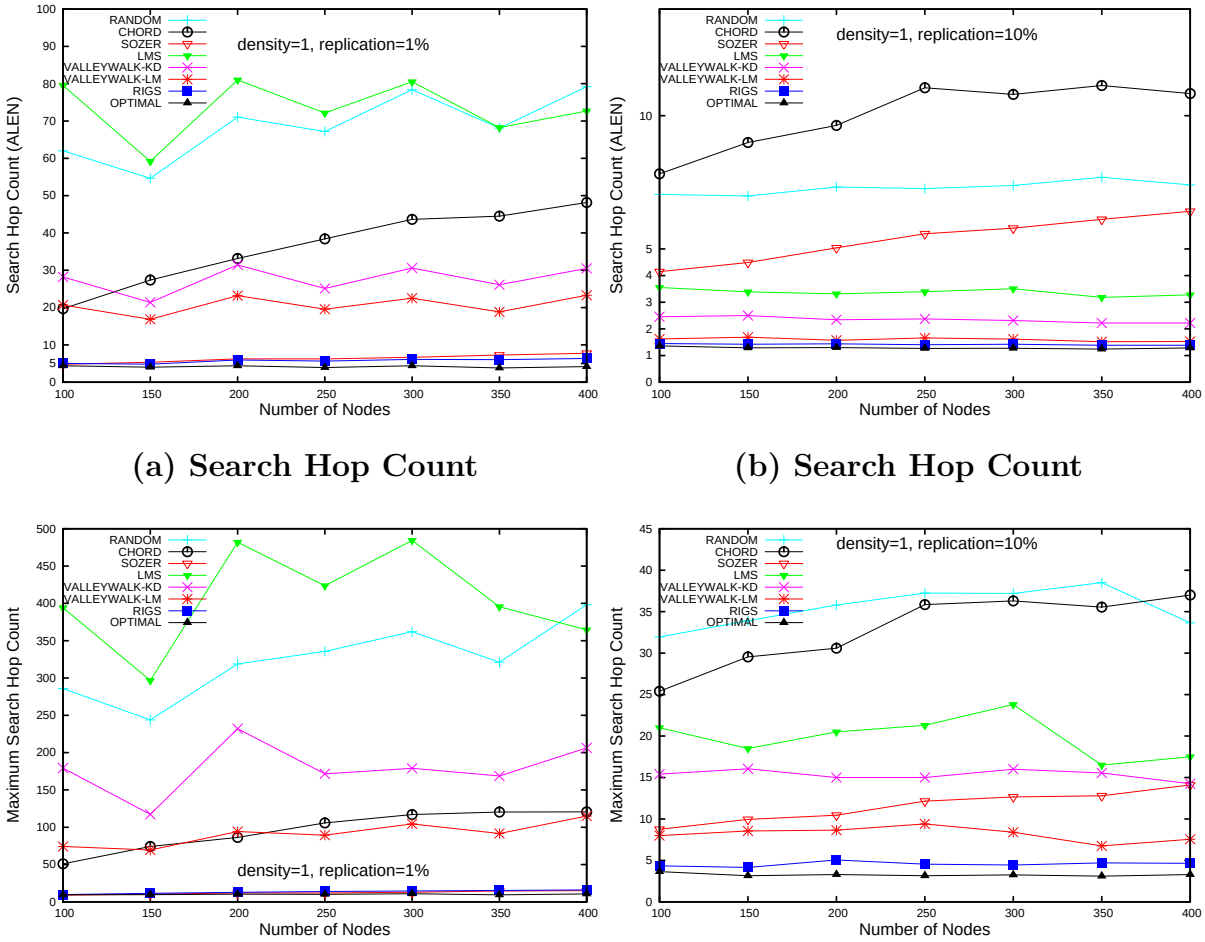
Figure 5.7: Search performance by network size - (3)

increases, which mostly improve the searching performance because the network diameter decreases and more information about other nodes is available to each node. RandomWalk, however, cannot benefit from this abundant information because of the blindness of its searching method. What is worse for RandomWalk with dense network is that even if the key-holder is one of its neighbor, the probability of hitting that neighbor is becomes low with a large number of degrees. The search-hop-stretch in Figure 5.9-(a),(b) and the detour-hop-stretch in Figure 5.10-(a),(b) clearly shows this trend.

Since the range of "local" becomes larger as the number of degree increases, the locality of each scheme, whether is close to Optimal.

5.2.4 Tail Probability

The tail probability of a given search length is the probability that the search length is at least the given length. This metric is useful to engineer the algorithm



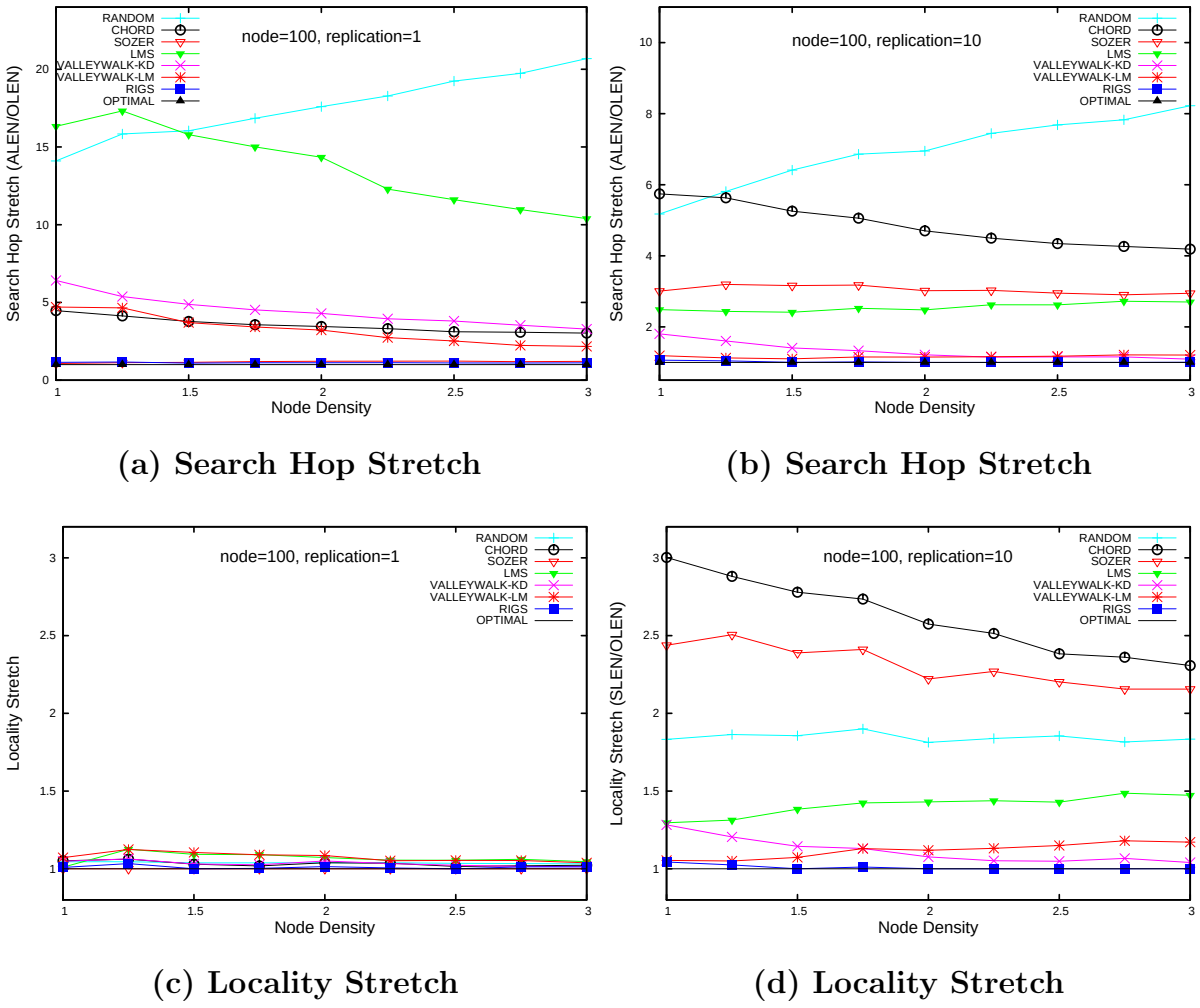
(a) Search Hop Count **(b) Search Hop Count**

(c) Maximum Search Hop Count **(d) Maximum Search Hop Count**

Figure 5.8: Search performance by network size - (4)

in terms of worst-case performance as well as the variance of the performance. In this simulations, I vary the density 1, or 2, and the replication factor 1%, 5%, and 10%, with fixed network size of 100.

Figure 5.13-(a) shows that with a small number of replications most queries are found in 10 hops for RIGS while VALLEY-WALK$_{LM}$ and VALLEY-WALK$_{KD}$ can have search length more than 40 hops with a probability of more than 10%. As there are more replications, the search hop count decreases down to 6 for RIGS.



(a) Search Hop Stretch **(b) Search Hop Stretch**

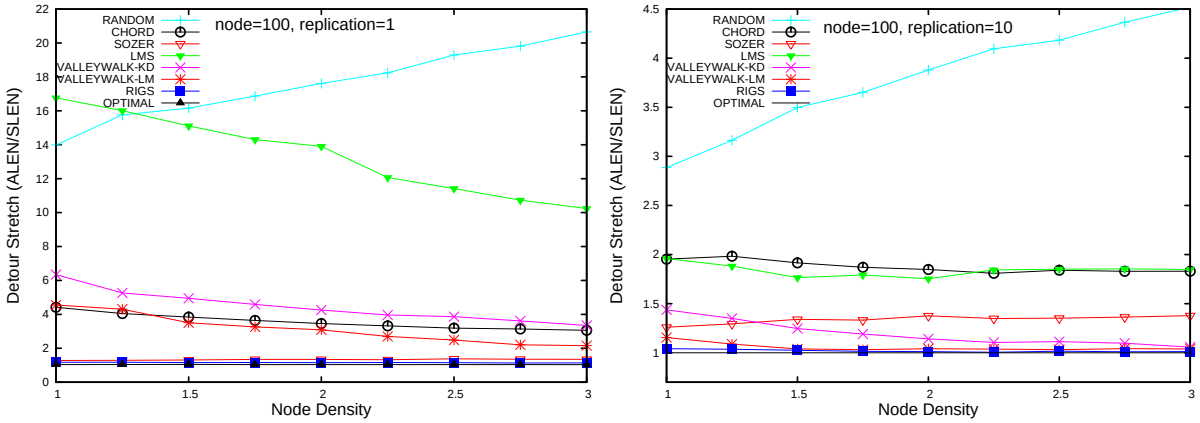
(c) Locality Stretch **(d) Locality Stretch**

Figure 5.9: Search performance by node density - (1)

The increased network density improves the tail probabilities such that RIGS can find most of queries only in 3 hops (see Figure 5.14-(b)).

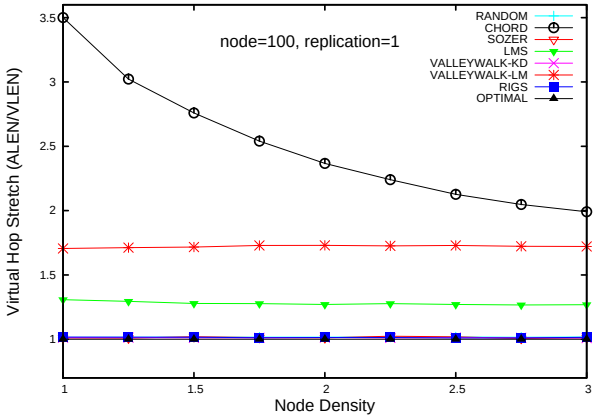
5.2.5 Random Graph

In this section, I run the same set of simulation as in Section 5.2.1 for random geometric graphs with 100 nodes and 2.0 density. I used the density of 2.0 because a density below 2.0 makes the network disconnected.

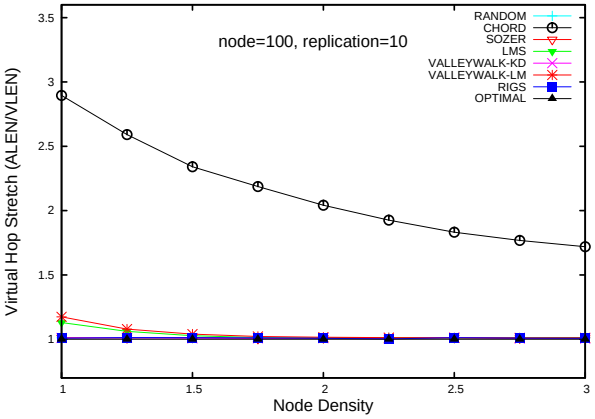


(c) Detour Stretch

(d) Detour Stretch



(a) Virtual Hop Stretch



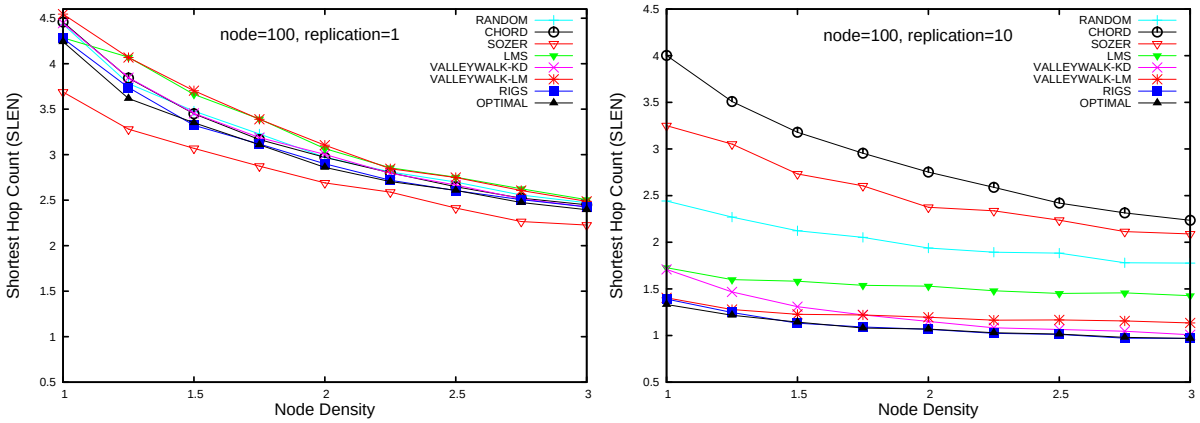
(b) Virtual Hop Stretch

Figure 5.10: Search performance by node density - (2)

Figure 5.15 and 5.16 shows the similar trends as with mesh network topology. But the performance degrade of VALLEY-WALK$_{LM}$ compared with OPTIMAL is more apparent.

5.3 Results For Authentication Performance

In this section, I test for the feasibility of key-discovery algorithms when used for user authentication problem in wireless mesh networks. I use the metrics de-



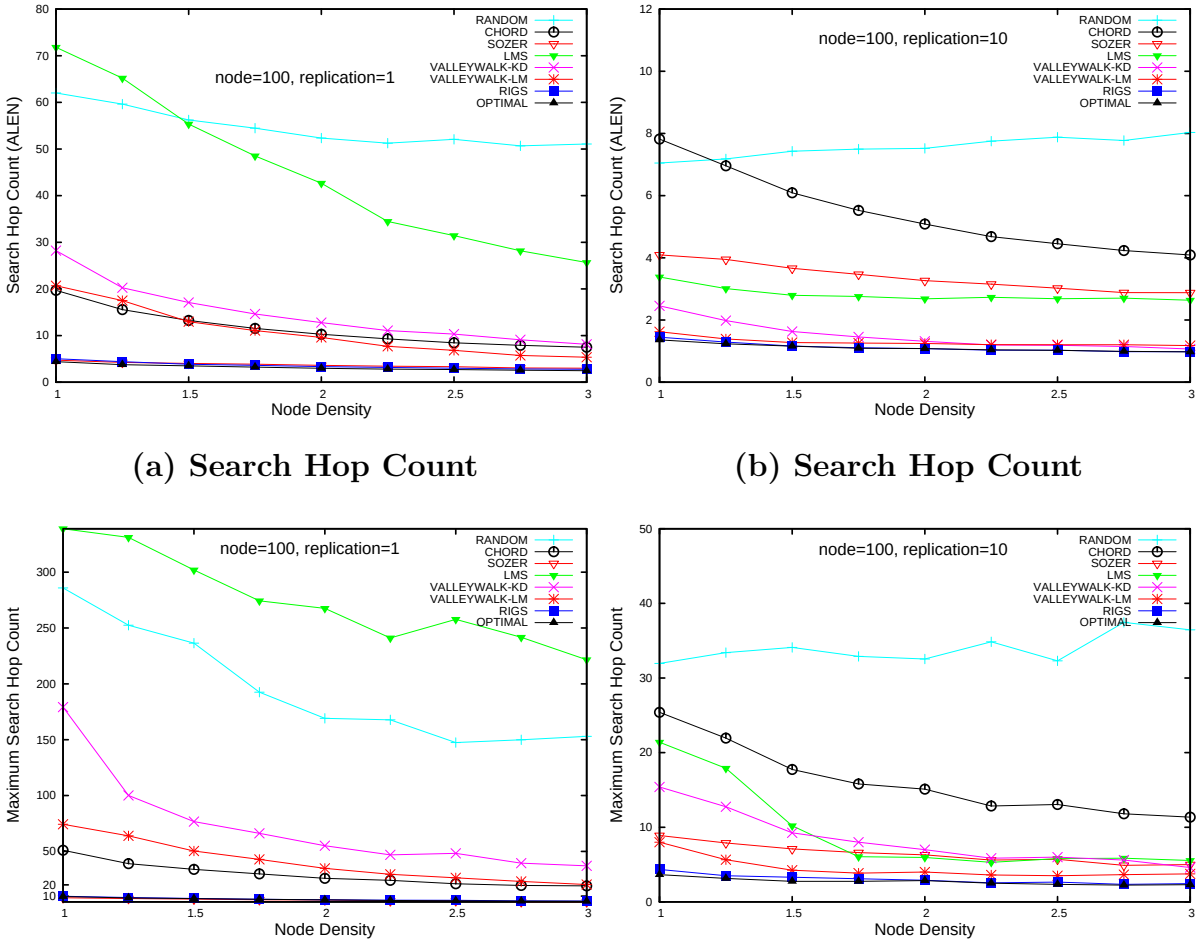
(a) Shortest Hop Count **(b) Shortest Hop Count**

Figure 5.11: Search performance by node density - (3)

scribed in Section 5.1.2.

In this set of simulations, I vary the aggregate user arrival rate from 20 users per second to 100 users per second with fixed networks size of 100, node density of 1, and replication factors of 5% or 10%.

Figure 5.17(a)–(b) shows the authentication success ratio with replications factor of 5% and 10%. With a small number of replications, say 5%, even optimal solution cannot achieve high success ratio, say more than 90%, if the user arrival rate is more than 60. However, RIGS is achieving close to optimal performance all the time with the difference less than 10%. But when we have 10% replications, RIGS is achieving more than 90% success ratio and very close to the optimal solution. VALLEY-WALK$_{LM}$ also performs well with almost 90% success ratio even with the highest user arrival rate. Note that with both replication ratio 5% and 10%, VALLEY-WALK$_{LM}$ performs similar which also matches the result of other simulations, e.g., Figure 5.3-(a).

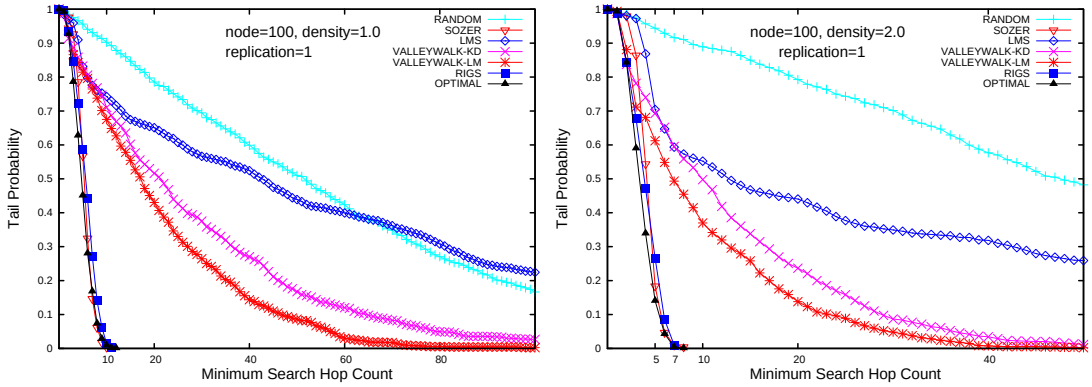


(a) Search Hop Count **(b) Search Hop Count**

(c) Maximum Search Hop Count **(d) Maximum Search Hop Count**

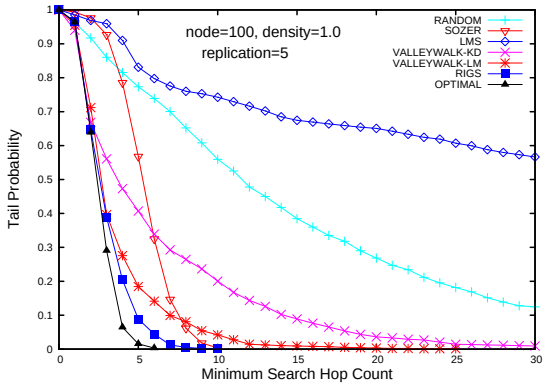
Figure 5.12: Search performance by node density - (4)

Figure 5.17-(c) and (d) shows the authentication rate (or authentication throughput) of each scheme. The authentication rate is defined as the average number of successful authentications per second. The diagonal line shows the maximum achievable authentication rate, which equals to the user arrival rate. The closer to the maximum achievable rate, the better the scheme performs. Note that, with 10% replications, optimal solutions as well as proposed solutions (except VALLEY-WALK$_{KD}$) performs close to the maximum achievable rates. The differ-

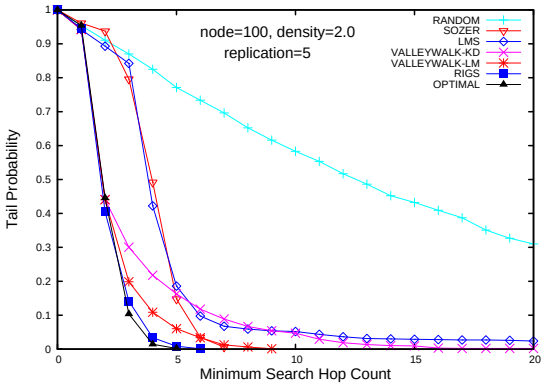


(a) Tail probability n=100, d=1.0, r=1

(b) Tail probability n=100, d=2.0, r=1



(c) Tail probability n=100, d=1.0, r=5

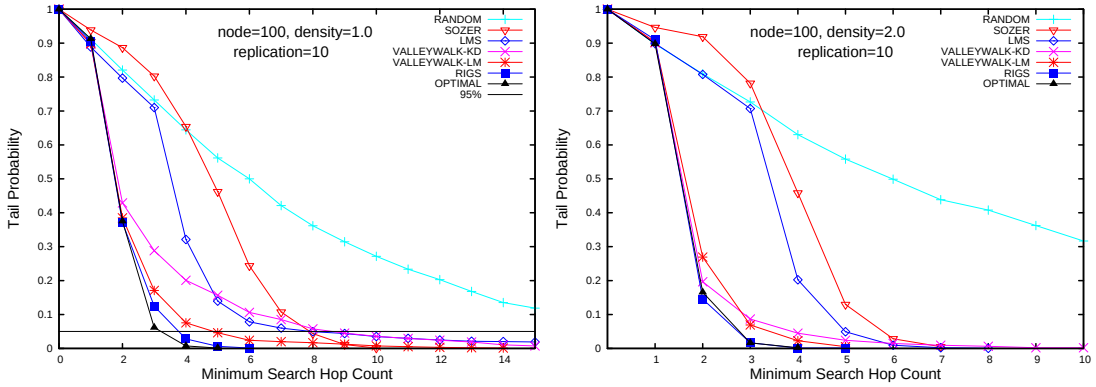


(d) Tail probability n=100, d=2.0, r=5

Figure 5.13: Tail probability-(1)

ence between OPTIMAL and the maximum achievable rate is due to interference and packet drops, which is inevitable with limited bandwidth of wireless links.

Now we look at the delay of the authentication process and the key discovery. In Figure 5.18-(a) and (b), with 5% replications, RIGS and VALLEY-WALK$_{LM}$ can finish the authentication protocol in one second for up to 60 user per seconds, but with 10% replications, they finish below 0.5 seconds even for the highest user arrival rate. We observe a dropping delay of CHORD with high user arrival rates and this is



(a) Tail probability n=100, d=1.0, r=10

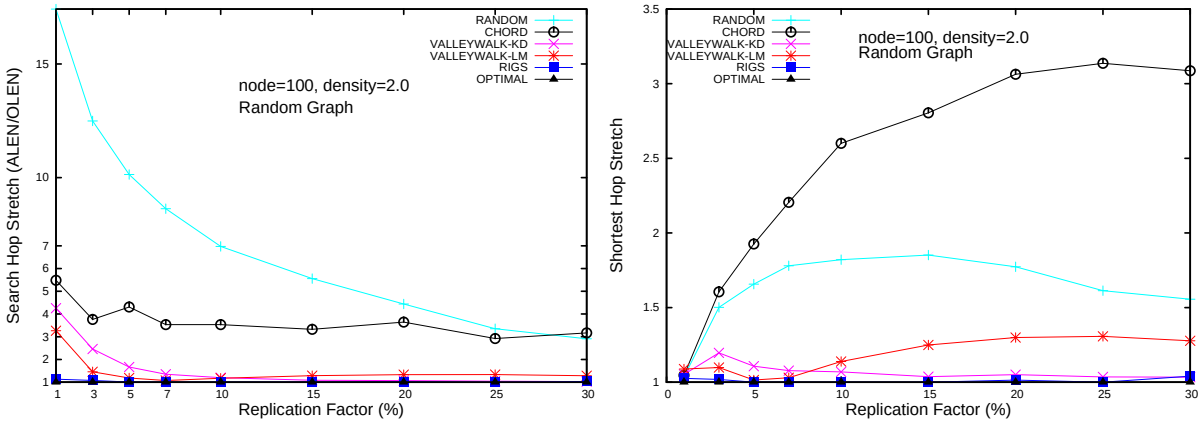
(b) Tail probability n=100, d=2.0, r=10

Figure 5.14: Tail probability-(2)

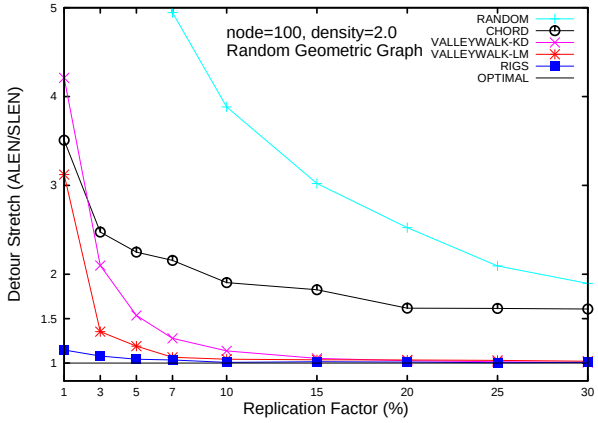
because the CHORD is failing to finish authentications more than 80% of the total requests and so the authentication with long delay cannot even finish the protocol. Figure 5.18-(c),(d) confirms this explanation showing that the search delay, which is the first message in the series of protocol messages, continuously increases unlike the authentication delay. Comparing with RANDOMWALK, although CHORD and RANDOMWALK achieves similar delay for the key discovery (figure (c)), CHORD fails with more authentication requests because it chooses farther destinations than RANDOMWALK(see Figure 5.3-(c)). Note that, with 10% replications, even with the highest user arrival rates, RIGS and VALLEY-WALK$_{LM}$ can finish the authentication in less than 0.5 seconds and finds the destination in less than 0.2 seconds.

5.4 Summary

Simulation results show that RIGS performs very close to optimal and VALLEY-WALK achieves the reasonable performance. With simple heuristic key store and oppor-



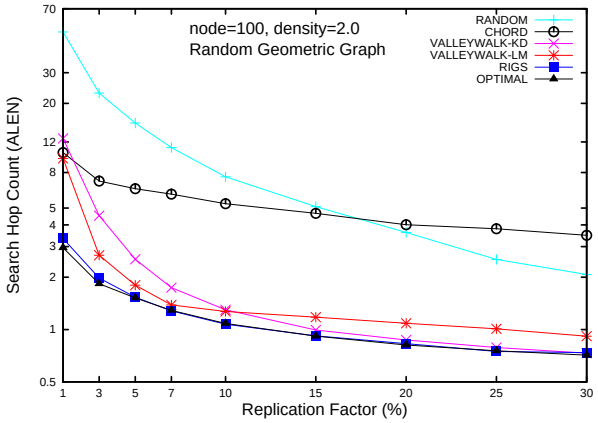
(a) Search Hop Stretch **(b) Locality Stretch**



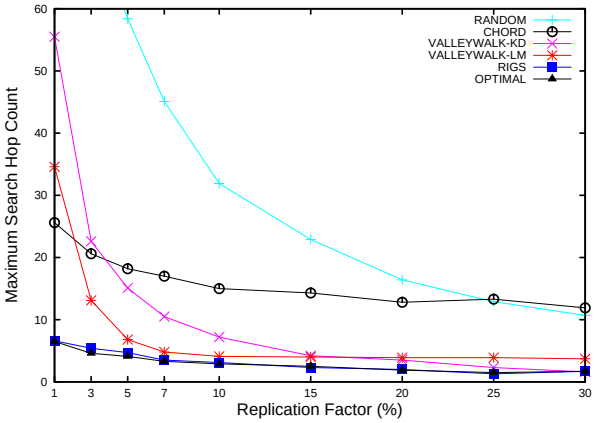
(c) Detour Stretch

Figure 5.15: Random geometric graph : by replication factors - (1)

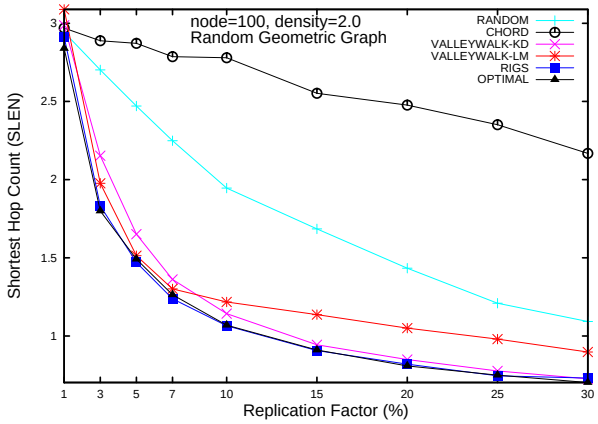
tunistic walk allows VALLEY-WALK for drastic improvement from random walk.



(a) Search Hop Count

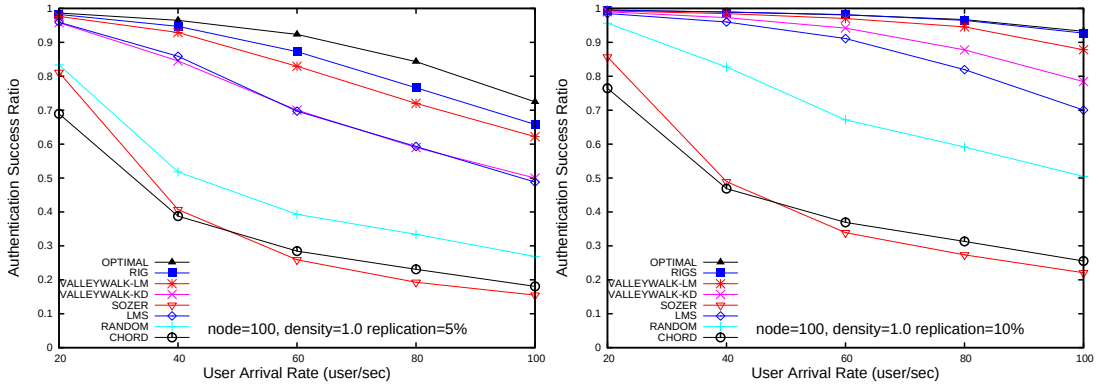


(b) Maximum Search Hop Count



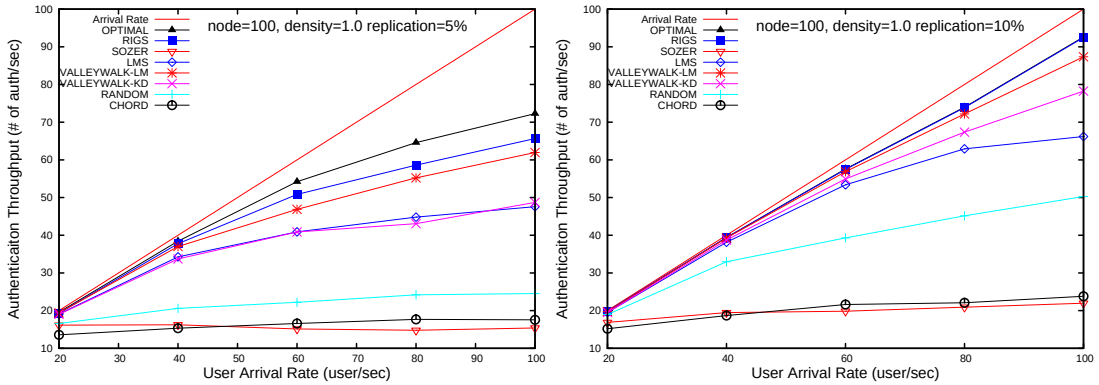
(c) Shortest Hop Count

Figure 5.16: Random geometric graph : by replication factors - (2)



(a) Success Ratio when r=5%

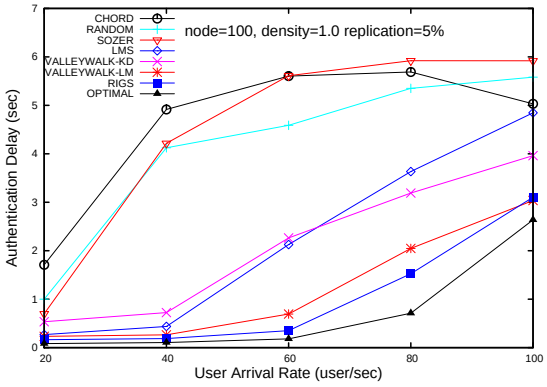
(b) Success Ratio when r=10%



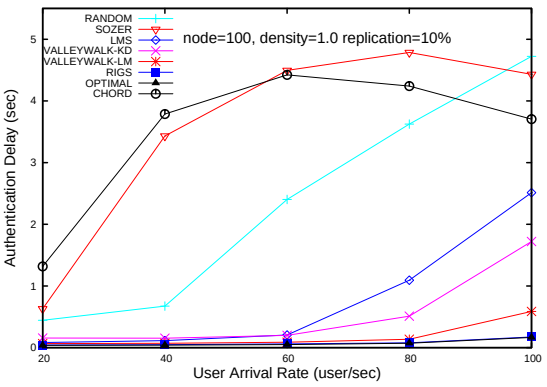
(c) Authentication Rate when r=5%

(d) Authentication Rate when r=10%

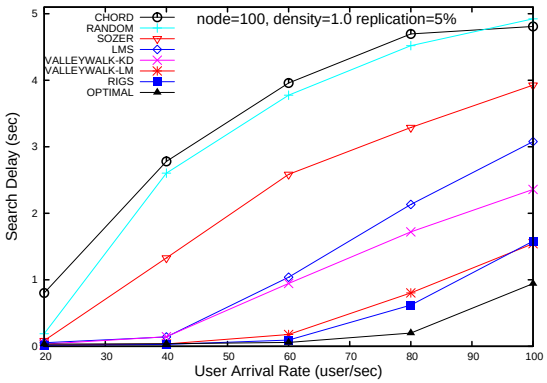
Figure 5.17: Authentication performance



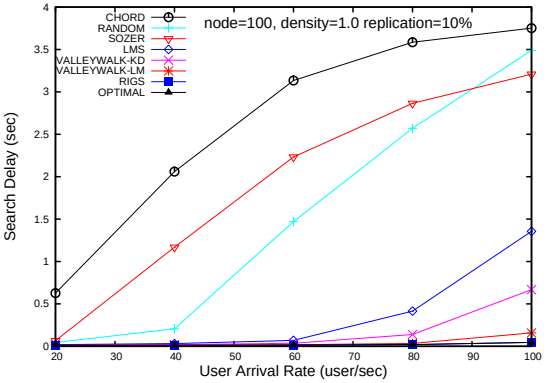
(a) Authentication delay when r=5%



(b) Authentication delay when r=10%



(c) Search Delay when r=5%



(d) Search Delay when r=10%

Figure 5.18: Authentication delay

Chapter 6

Conclusion And Future Work

6.1 Conclusion

In this dissertation work, I identified the key-discovery problem in wireless mesh networks and explored the possibilities for providing reliable, efficient, and scalable *object lookup service* in multi-hop wireless networks. Despite the promising characteristics of the existing DHT schemes in P2P networks, I identified that existing DHT schemes cannot scale in multi-hop wireless environments because of their independence of underlying topology. Therefore, I proposed a novel concept of topology-dependent DHT and designed a loosely-structured solution VALLEY-WALK, and a tightly-structured TD-DHT solution RIGS.

VALLEY-WALK is minimally structured and the node states are independent of each other. Therefore, this scheme is low-cost and robust against network dynamics. VALLEY-WALK, however, performs well only with reasonable fraction of replications because of its blindness for the global state. The differences between VALLEY-WALK$_{LM}$ and VALLEY-WALK$_{KD}$ comes from how to distribute objects: pre-distribution or post-distribution, respectively. VALLEY-WALK$_{LM}$ is specially designed for the case when the objects should be stored before the deployment in which the deploying entity is unaware of the actual topology of the network. VALLEY-WALK$_{KD}$ can achieve better performance because it can optimize the ob-

ject locations to maximize the search performance, knowing the network topology. The limitation of VALLEY-WALK$_{KD}$ is the performance degrade if the number of replications differ from the number of natural local minima. I also provide in depth analysis of VALLEY-WALK which provide much more tight bound on the performance of LMS than that of [12].

On the contrary, RIGS features a global structure which is constructed in a distributed manner by each node. The construction requires only a global parameter, the network size. RIGS utilizes a novel lookup structure RIG to store and lookup objects of interest. RIGS can have different characteristics depending on the spanning tree upon which RIGS builds the RIG structure. I experimented with two easy-to-build spanning trees, BFS and DFS. Due to the balanced property of BFS, RIGS-BFS performed better than RIGS-DFS all the time. Moreover, RIGS-BFS is achieving close to optimal performance in the most scenarios.

Experiments with authentication problems in wireless mesh networks shows that the proposed lookup services can successfully deliver a reliable and scalable key-discovery service as much as the optimal solution can provide.

6.2 Future Work

I plan to continue with the exploration of TD-DHT scheme because I believe it has many potential applications in multi-hop wireless networks.

The network dynamics become more and more important and I want to investigate how proposed schemes can handle such situations, especially when the nodes

arrive and leave. VALLEY-WALK schemes need to address following issues. First, how to detect such dynamics and update node states. This will be easily done by periodic updates or request-of-leave if it is a planned leave. The change of neighborhood can change the location of local minima, so the object will be relocated accordingly to maximize the probability of successful search.

RIGS has more challenging tasks to maintain its global structure in face of network dynamics. I plan to start with the incremental node join where each node joining the network first make a contact to a neighbor node and become a child node of the existing node by choosing its node *id* properly. We can keep the RIG property while adding one more node at any place of the existing RIG graph. In the perspective of consistent hashing, the uniform property of node *id* converts to the uniform property of node locations. Leave of a node is more challenging. A removal of node makes a graph partitioned and the isolated group of nodes will actively find through which link, that was not in the RIG before, they can attach to the other part of the graph with minimal state updates and key relocations.

Probabilistic analysis on the performance of RIGS is of interest to provide analytic guarantee of the scheme. Especially the tail probability, as given for VALLEY-WALK in this dissertation, will be closely investigated.

As a long term plan, I want to implement TD-DHT in a real environment and explore with various applications. For example, a sensor network requires various lookup services such as data query and data repository discovery. Also content-based routing will be one of its applications including sensor networks. In wireless mesh networks, not only the key-discovery problem, but also other applications are

of useful, such as user profile lookup system or even the P2P-like information-sharing system in a metropolitan area.

Appendix A

Proofs For Theorems In Chapter 3

A.1 Proof Of Theorem 3.5.2

Theorem 3.5.2 (VALLEY-WALK$_{LM}$-*iid*). *If $X_0, X_1, \ldots$ follow identical independent distribution, then, the tail probability of the search length L is*

$$Pr(L \geq k) = \frac{1}{(k+1)!} \tag{A.1}$$

and

$$E[L] = e - 2 \tag{A.2}$$

where e is the base of natural logarithm.

Proof Note that

$$Pr(L \geq k) = Pr(X_0 > X_1 > \ldots > X_k).$$

Suppose we sample $k+1$ numbers and denote the ith largest value by Y_i where $i = 0, 1, \ldots, k$. Since X_i can equally take any value of Y_j, i.e., $Pr[X_i = Y_j] = Pr[X_i = Y_k]$ where $j \neq k$, any one-to-one matching between $\{X_0, X_1, \ldots, X_k\}$ and

$\{Y_0, Y_1, \ldots, Y_k\}$ are equally possible. Since there are $(k+1)!$ possible matchings, the probability that X_i's have decreasing order is

$$Pr(X_0 > X_1 > \ldots > X_k) = \frac{1}{(k+1)!}.$$

Therefore, the probability that search length equals to k is

$$\begin{aligned} Pr(L = k) &= Pr(L \geq k) - Pr(L \geq k+1) \\ &= \frac{1}{(k+1)!} - \frac{1}{(k+2)!} \\ &= \frac{k+1}{(k+2)!} \end{aligned}$$

and the expected number of length is

$$\begin{aligned} E[L] &= \sum_{k=1}^{\infty} Pr(L \geq k) && \text{(A.3)} \\ &= \sum_{k=1}^{\infty} \frac{1}{(k+1)!} && \text{(A.4)} \\ &= \sum_{k=0}^{\infty} \frac{1}{k!} - 2 && \text{(A.5)} \\ &= e - 2, && \text{(A.6)} \end{aligned}$$

where we get Equation A.5 by the Taylor expansion of e. ∎

A.2 Proof Of Theorem 3.5.3

Theorem 3.5.3 (VALLEY-WALK$_{LM}$). *For the search length L of* VALLEY-WALK$_{LM}$,

$$\begin{aligned} Pr(L \geq k) &\leq \frac{1}{k!} \sum_{i=0}^{k} \binom{k}{i} \frac{(-1)^i}{\Delta i + 1} && \text{(A.7)} \\ &\leq \frac{1}{k!} && \text{(A.8)} \end{aligned}$$

Proof Let $\hat{f}(x_0, x_1, \ldots, x_k)$ be the joint pdf of $X_0, \ldots, X_k$, and, because of the independence,

$$\hat{f}(x_0, x_1, \ldots, x_k) = f_0(x_0) f_1(x_1) \cdots f_k(x_k).$$

Therefore, we have that

$$\begin{aligned} Pr(L \geq k) &= Pr(X_0 \geq X_1 \geq \ldots \geq X_k) && \text{(A.9)} \\ &= \int_0^1 \int_0^{x_0} \ldots \int_0^{x_{k-1}} \hat{f}(x_0, x_1, \ldots, x_k)\, dx_k \ldots dx_1 dx_0 && \text{(A.10)} \\ &= \int_0^1 \int_0^{x_0} \ldots \int_0^{x_{k-1}} f_0(x_0) f_1(x_1) \cdots f_k(x_k)\, dx_k \ldots dx_1 dx_0 && \text{(A.11)} \end{aligned}$$

Since we get an upper-bound if we increase the number of candidate nodes from $\hat{d}_i$ to Δ, so do we if $f_i(x)$'s are replaced by $f(x)$ for $i > 0$, thus

$$\begin{aligned} Pr(L \geq k) &\leq \int_0^1 \int_0^{x_0} \ldots \int_0^{x_{k-1}} f_0(x_0) f(x_1) \cdots f(x_k)\, dx_k \ldots dx_1 dx_0 && \text{(A.12)} \\ &= \int_0^1 f_0(x_0) \int_0^{x_0} f(x_1) \cdots \int_0^{x_{k-1}} f(x_k)\, dx_k \ldots dx_1 dx_0 && \text{(A.13)} \end{aligned}$$

$$
\begin{aligned}
&= \int_0^1 f_0(x_0) \int_0^{x_0} f(x_1) \ldots \int_0^{x_{k-2}} f(x_{k-1}) F(x_{k-1})\, dx_{k-1} \ldots dx_0 \\
&= \int_0^1 f_0(x_0) \int_0^{x_0} f(x_1) \ldots \int_0^{x_{k-3}} f(x_{k-2}) \frac{(F(x_{k-2}))^2}{2}\, dx_{k-2} \ldots dx_1 dx_0 \\
&\vdots \\
&= \int_0^1 f_0(x_0) \frac{(F(x_0))^k}{k!}\, dx_0 \qquad \text{(A.14)} \\
&= \int_0^1 \frac{(1-(1-x_0)^{\Delta})^k}{k!}\, dx_0 \qquad \text{(A.15)}
\end{aligned}
$$

We get Equation A.15 from 3.7 with $d_{-1} = 1$ and because $F(x) = 1 - (1-x)^{\Delta}$. If we use Binomial expansion,

$$
(1-(1-x_0)^{\Delta})^k = \sum_{i=0}^{k} \binom{k}{i} (-1)^i (1-x)^{\Delta i}, \qquad \text{(A.16)}
$$

therefore, we have

$$
\begin{aligned}
Pr(L \geq k) &\leq \frac{1}{k!} \sum_{i=0}^{k} \binom{k}{i} (-1)^i \int_0^1 (1-x)^{\Delta i} dx \qquad \text{(A.17)} \\
&= \frac{1}{k!} \sum_{i=0}^{k} \binom{k}{i} \frac{(-1)^i}{\Delta i + 1} \qquad \text{(A.18)}
\end{aligned}
$$

Now we prove that

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

Given $f(x)$, let $Df(x)$ denote the *finite difference* [79] such that $Df(x) = f(x+1) - f(x)$. Then,

$$
D^2 f(x) = Df(x+1) - Df(x) \qquad \text{(A.19)}
$$

$$= f(x+2) - 2f(x+1) + f(x) \tag{A.20}$$

$$D^3 f(x) = f(x+3) - 3f(x+2) + 3f(x+1) - f(x) \tag{A.21}$$

$$\vdots \tag{A.22}$$

$$D^k f(x) = \sum_{i=0}^{k} \binom{k}{i} (-1)^{k-i} f(x+i). \tag{A.23}$$

Let $f(x) = 1/x$, then

$$Df(x) = \frac{1}{x+1} - \frac{1}{x} = (-1)\frac{1}{x(x+1)} \tag{A.24}$$

$$D^2 f(x) = (-1)\left\{\frac{1}{(x+1)(x+2)} - \frac{1}{x(x+1)}\right\} \tag{A.25}$$

$$= (-1)^2 \frac{2}{x(x+1)(x+2)} \tag{A.26}$$

$$D^3 f(x) = (-1)^3 \frac{3!}{x(x+1)(x+2)(x+3)} \tag{A.27}$$

$$\vdots \tag{A.28}$$

$$D^k f(x) = (-1)^k \frac{k!}{x(x+1)(x+2)\cdots(x+k)} \tag{A.29}$$

Therefore, by Equations A.23 and A.29, if we rule out $(-1)^k$,

$$\sum_{i=0}^{k} \binom{k}{i} \frac{(-1)^i}{x+i} = \frac{k!}{x(x+1)(x+2)\cdots(x+k)} \tag{A.30}$$

$$\sum_{i=0}^{k} \binom{k}{i} \frac{(-1)^i}{\Delta x + \Delta i} = \frac{k!}{\Delta x(x+1)(x+2)\cdots(x+k)}. \tag{A.31}$$

Letting $x = \frac{1}{\Delta}$ and dividing by $\frac{1}{k!}$, we have

$$\frac{1}{k!} \sum_{i=0}^{k} \binom{k}{i} \frac{(-1)^i}{\Delta i + 1} = \frac{1}{(\frac{1}{\Delta}+1)(\frac{1}{\Delta}+2)\cdots(\frac{1}{\Delta}+k)} \tag{A.32}$$

$$\leq \frac{1}{k!} \tag{A.33}$$

∎

A.3 Proof Of Lemma 3.5.4

Lemma 3.5.4. *For $i, k \geq 0$,*

$$\int_0^1 x^i(1-x)^k \, dx = \frac{1}{(i+k+1)\binom{i+k}{i}} \tag{A.34}$$

Proof Let $A(i,k) = x^i(1-x)^k$ then

$$A(i,k)|_{x=0}^{x=1} = 0 \quad \text{if } i > 0 \text{ and } k > 0 \tag{A.35}$$

and

$$\int_0^1 A(0,k) \, dx = \int_0^1 (1-x)^k = \frac{1}{k+1}. \tag{A.36}$$

By partial integration, we get

$$\int_0^1 A(i,k) \, dx = \int_0^1 x^i(1-x)^k \tag{A.37}$$

$$= -\frac{1}{k+1}x^i(1-x)^{k+1}|_{x=0}^{x=1} + \frac{i}{k+1}\int_0^1 x^{i-1}(1-x)^{k+1} \, dx \tag{A.38}$$

$$= \frac{i}{k+1}\int_0^1 A(i-1,k+1)\ dx \tag{A.39}$$

We get Equation A.39 because of Equation A.35. Therefore, for $i > 0, k > 0$,

$$\begin{aligned}
\int_0^1 A(i,k)\ dx &= \frac{i}{k+1}\int_0^1 A(i-1,k+1)\ dx && \text{(A.40)}\\
&= \frac{i(i-1)}{(k+1)(k+2)}\int_0^1 A(i-2,k+2)\ dx && \text{(A.41)}\\
&\ \ \vdots && \text{(A.42)}\\
&= \frac{i!}{(k+1)(k+2)\cdots(k+i)}\int_0^1 A(0,k+i)\ dx && \text{(A.43)}\\
&= \frac{1}{\binom{i+k}{i}}\int_0^1 A(0,i+k)\ dx && \text{(A.44)}\\
&= \frac{1}{\binom{i+k}{i}(i+k+1)}. && \text{(A.45)}
\end{aligned}$$

We get Equation A.44 by the definition of combinations and we use Equation A.39 to get A.45. ∎

A.4 Proof Of Theorem 3.5.5

Theorem 3.5.5 (Distribution of Y_k). *Let $D_1, D_2, \ldots, D_n$ be a set of continuous random variables which independently follow the uniform distribution on $[0,1)$. Denote the kth smallest value among them by a random variable Y_k. Then, for $y \in [0,1)$ and $1 \le k \le n$, the cdf of Y_k is $F_{Y_k}(y) = Pr(Y_k \le y) = 1 - Pr(Y_k > y)$ where*

$$Pr(Y_k > y) = 1 - \sum_{i=0}^{k-1}\binom{n}{i}y^i(1-y)^{n-i} \tag{A.46}$$

with

$$E[Y_k] \;=\; \frac{k}{n+1}.$$

Proof Note that $Y_1 < Y_2 < \ldots < Y_{k-1} < Y_k < Y_{k+1} < \ldots < Y_n$. Since Y_j is uniformly distributed on $[0, 1)$,

$$\begin{aligned} Pr(y < Y_j) &= y, \\ Pr(y < Y_j) &= 1 - y, \end{aligned}$$

and $Pr(y < Y_1) = (1-y)^n$ because y should be less than all the Y_j's. The probability that y lies between Y_k and Y_{k+1} equals to the probability that k of Y_j's are less than y and the rest $(n-k)$ of Y_j's are greater than y, thus we get

$$Pr(Y_k < y < Y_{k+1}) \;=\; \binom{n}{k} y^k (1-y)^{n-k}.$$

$Y_k > y$ if $(y < Y_1)$ or $(Y_1 < y < Y_2)$ or $\ldots$or $(Y_{k-1} < y < Y_k)$, where all cases are disjoint events. Therefore, the probability that the kth smallest value is larger than y is

$$\begin{aligned} Pr(Y_k > y) &= Pr(y < Y_1) + Pr(Y_1 < y < Y_2) + \ldots + Pr(Y_{k-1} < y < Y_k)) \\ &= \sum_{i=0}^{k-1} \binom{n}{i} y^i (1-y)^{n-i} \end{aligned}$$

and the expected value of Y_k is

$$\begin{aligned} E[Y_k] &= \int_0^1 Pr(Y_k > y)\, dy && \text{(A.47)} \\ &= \int_0^1 \sum_{i=0}^{k-1} \binom{n}{i} y^i (1-y)^{n-i}\, dy && \text{(A.48)} \\ &= \sum_{i=0}^{k-1} \binom{n}{i} \int_0^1 y^i (1-y)^{n-i}\, dy && \text{(A.49)} \\ &= \sum_{i=0}^{k-1} \binom{n}{i} \frac{1}{(n+1)\binom{n}{i}} && \text{(A.50)} \\ &= \frac{k}{n+1} && \text{(A.51)} \end{aligned}$$

We derive Equation A.50 by Lemma 3.5.4. ∎

A.5 Proof Of Lemma 3.5.6

Lemma 3.5.6. *The probability that the first node v_0 becomes an object-holder is*

$$Pr(v_0 \text{ becomes an object-holder}) = \frac{r}{n+1} \qquad \text{(A.52)}$$

and the probability that ith node in the search path v_i ($i > 0$) becomes an object-holder is

$$Pr(v_i \text{ becomes an object-holder}) = d \sum_{i=0}^{r-1} \frac{\binom{n}{i}}{(n+d)\binom{n+d-1}{i}} \qquad \text{(A.53)}$$

where d is the degree of v_{i-1}.

Proof Denote the random variables for $dist_k(v_0)$ and $dist_k(v_i)$ by X_0 and X, respectively. Since X_0 follows the uniform distribution on $[0, 1)$, X_0 has *cdf* $F_0(x) = x$ and *pdf* $f_0(x) = 1$. If v_{i-1} has degree d, X has *cdf* $F_X(x) = 1 - (1 - x)^d$ and *pdf* $f_X(x) = d(1 - x)^{d-1}$. Let $f_{Y_r}(y)$ be *pdf* of the random variable Y_r. Let $f_{X_0,Y_r}(x, y)$ be the joint *pdf* of X_0 and Y_r and $f_{X,Y_r}(x, y)$ be the joint *pdf* of X and Y_r. Since Y_r is independent with X_0 and X, we have

$$f_{X_0,Y_r}(x, y) = f_{X_0}(x) \cdot f_{Y_r}(y) \tag{A.54}$$

$$f_{X,Y_r}(x, y) = f_X(x) \cdot f_{Y_r}(y) \tag{A.55}$$

Then, the probability that the first node becomes an object-holder is

$$\begin{aligned}
Pr(X_0 \leq Y_r) &= \int_0^1 \int_x^1 f_{X_0,Y_r}(x, y)dydx && \text{(A.56)}\\
&= \int_0^1 f_{X_0}(x) \int_x^1 f_{Y_r}(y)dydx && \text{(A.57)}\\
&= \int_0^1 f_{X_0}(x)Pr(Y_r > x)dx && \text{(A.58)}\\
&= \int_0^1 f_{X_0}(x) \sum_{i=0}^{r-1} \binom{n}{i} x^i(1 - x)^{n-i}dx && \text{(A.59)}\\
&= \int_0^1 \sum_{i=0}^{r-1} \binom{n}{i} x^i(1 - x)^{n-i}dx && \text{(A.60)}\\
&= \sum_{i=0}^{r-1} \binom{n}{i} \int_0^1 x^i(1 - x)^{n-i}dx && \text{(A.61)}\\
&= \sum_{i=0}^{r-1} \frac{\binom{n}{i}}{(n + 1)\binom{n}{i}} && \text{(A.62)}\\
&= \frac{r}{n + 1} && \text{(A.63)}
\end{aligned}$$

and the probability that a node $v_i (i > 0)$ becomes an object-holder is

$$
\begin{aligned}
Pr(X \leq Y_r) &= \int_0^1 \int_x^1 f_{X,Y_r}(x,y) dy dx && \text{(A.64)} \\
&= \int_0^1 f_X(x) \int_x^1 f_{Y_r}(y) dy dx && \text{(A.65)} \\
&= \int_0^1 f_X(x) Pr(Y_r > x) dx && \text{(A.66)} \\
&= \int_0^1 f_X(x) \sum_{i=0}^{r-1} \binom{n}{i} x^i (1-x)^{n-i} dx && \text{(A.67)} \\
&= \int_0^1 d(1-x)^{d-1} \sum_{i=0}^{r-1} \binom{n}{i} x^i (1-x)^{n-i} dx && \text{(A.68)} \\
&= d \sum_{i=0}^{r-1} \binom{n}{i} \int_0^1 x^i (1-x)^{n+d-i-1} dx && \text{(A.69)} \\
&= d \sum_{i=0}^{r-1} \frac{\binom{n}{i}}{(n+d)\binom{n+d-1}{i}} && \text{(A.70)}
\end{aligned}
$$

Equation A.59 and A.67 are by Equation A.46 and Equation A.62 and A.70 are by Equation A.34. ∎

Bibliography

[1] [Online]. Available: http://www.emnwifi.net

[2] [Online]. Available: http://www.waztempe.com

[3] [Online]. Available: http://www.belairnetworks.com

[4] [Online]. Available: http://research.microsoft.com/mesh

[5] [Online]. Available: http://www.motorola.com

[6] [Online]. Available: http://www.meshdynamics.com

[7] [Online]. Available: http://www.tropos.com

[8] [Online]. Available: http://www.strixsys.com

[9] C. E. Perkins and E. M. Belding-Royer, "Ad hoc on-demand distance vector (AODV) routing," in *IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999.

[10] V. Park and M. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *IEEE Infocom*, Apr. 1997.

[11] S.-J. Lee and M. Gerla, "Dynamic load-aware routing in ad hoc networks," in *Proceedings of IEEE ICC 2001, Helsinki, Finland*, 2001, pp. 3206–3210.

[12] R. Morselli, B. Bhattacharjee, A. Srinivasan, and M. A. Marsh, "Efficient lookup on unstructured topologies," in *PODC '05: Proceedings of the twenty-fourth annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*. New York, NY, USA: ACM Press, 2005, pp. 77–86.

[13] P. Ganesan, Q. Sun, and H. Garcia-Molina, "Yappers: A peer-to-peer lookup service over arbitrary topology," in *Proceedings of IEEE INFOCOM, San Francisco, California, USA (2003)*, 2003.

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

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *SIGCOMM 2001*, vol. 31, no. 4. ACM Press, October 2001, pp. 161–172.

[16] 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)*, Nov. 2001, pp. 329–350.

[17] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and," Berkeley, CA, USA, Tech. Rep., 2001.

[18] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," in *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM Press, 2001, pp. 202–215.

[19] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in *Proceedings of ACM ASPLOS*. ACM, November 2000.

[20] A. I. T. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," in *Symposium on Operating Systems Principles*, 2001, pp. 188–201.

[21] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: making backup cheap and easy," in *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*. New York, NY, USA: ACM Press, 2002, pp. 285–298.

[22] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications (JSAC)*, vol. 20, no. 8, pp. 1489–1499, 2002.

[23] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: a decentralized peer-to-peer web cache," in *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*. New York, NY, USA: ACM Press, 2002, pp. 213–222.

[24] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: a bandwidth-intensive content streaming system," in *3rd International Workshop on Peer-to-Peer Systems (IPTPS 03)*, Berkeley, CA, February 2003.

[25] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of NOSSDAV*, June 2001.

[26] M. Castro, P. Druschel, Y. Hu, and A. Rowstron, "Topologyaware routing in structured peer-to-peer overlay networks," 2002.

[27] J. Li, C. Blake, D. S. D. Couto, H. I. Lee, and R. Morris, "Capacity of ad hoc wireless networks," in *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM Press, 2001, pp. 61–69.

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

[29] A. Gamal, J. Mammen, B. Prabhakar, and D. Shah, "Throughput-delay trade-off in wireless networks," 2004. [Online]. Available: citeseer.ist.psu.edu/elgamal04throughputdelay.html

[30] Z. Cheng and W. B. Heinzelman, "Flooding strategy for target discovery in wireless networks," *Wirel. Netw.*, vol. 11, no. 5, pp. 607–618, 2005.

[31] N. Chang and M. Liu, "Revisiting the ttl-based controlled flooding search: optimality and randomization," in *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM Press, 2004, pp. 85–99.

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

[33] M. Castro, P. Druschel, Y. Hu, and A. Rowstron, "Exploiting network proximity in distributed hash tables," 2002.

[34] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 2002, pp. 1190–1199 vol.3.

[35] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao, "Distributed object location in a dynamic network," in *SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM Press, 2002, pp. 41–52.

[36] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM Press, 1997, pp. 311–320.

[37] 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 '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, 2003, pp. 381–394.

[38] 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.

[39] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *In Proceedings of the 16th annual ACM International Conference on supercomputing*, 2002.

[40] 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.

[41] G. H. L. Fletcher, H. A. Sheth, and K. Brner, "Unstructured peer-to-peer networks : Topological properties and search performance," vol. 3601. Springer, Berlin, 2005, pp. 14–27.

[42] M. Penrose, *Random Geometric Graphs (Oxford Studies in Probability)*. Oxford University Press, USA, July 2003.

[43] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," in *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, 2002, pp. 177–190.

[44] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 1984, pp. 47–57.

[45] I. Kamel and C. Faloutsos, "Parallel R-trees," in *SIGMOD92*, 1992, pp. 195–204.

[46] N. Koudas, C. Faloutsos, and I. Kamel, "Declustering spatial databases on a multi-computer architecture," in *Proceedings of the 5th International Conference on Extending Databases Technology (EDBT)*, 1996.

[47] B. Schnitzer and S. T. Leutenegger, "Master-Client R-Trees: A new parallel R-tree architecture," in *SSDBM99*, 1999, pp. 68–77.

[48] A. Mondal, Yilifu, and M. Kitsuregawa, "P2PR-tree: An R-tree-based spatial index for peer-to-peer environments," in *Proceedings of the 1st international workshop on P2P Computing and Databases*, 2004.

[49] B. Nam and A. Sussman, "Spatial indexing of distributed multidimensional datasets," in *CCGRID05*, May 2005.

[50] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram, "Querying peer-to-peer networks using P-trees," in *Proceedings of the 7th International Workshop on the Web and Databases (WebDB)*, 2004.

[51] [Online]. Available: http://www.napster.com

[52] [Online]. Available: http://www.morpheus.com

[53] [Online]. Available: http://www.gnutella..com

[54] B. Yang and H. Garcia-Molina, "Efficient search in peer-to-peer networks," in *PODC '05: Proceedings of the twenty-fourth annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing.* New York, NY, USA: ACM Press, 2005, pp. 77–86.

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

[56] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing.* New York, NY, USA: ACM Press, 1997, pp. 654–663.

[57] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Communications of the ACM*, vol. 33, no. 6, pp. 668–676, June 1990.

[58] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher, "Adaptive replication in peer-to-peer systems," in *The 24th International Conference on Distributed Computing Systems*, March 2004.

[59] M. Cai, A. Chervenak, and M. Frank, "A peer-to-peer replica location service based on a distributed hash table," in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing.* Washington, DC, USA: IEEE Computer Society, 2004, p. 56.

[60] M. Christodoulidou and P. Fatourou, "Simple and efficient replication in chord." Proceedings of the IASTED Parallel and Distributed Computing and Systems (PDCS'06), Dallas, Texas, USA, 2006.

[61] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks." *Nature*, vol. 393, no. 6684, pp. 440–442, June 1998.

[62] 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.

[63] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Mixing times for random walks on geometric random graphs," *SIAM ANALCO (Workshop on Analytic Algorithmics & Combinatorics), Vancouver*, 2005.

[64] 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.

[65] A. Duran and C.-C. Shen, "Mobile ad hoc p2p file sharing," in *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, vol. 1, 2004, pp. 114–119 Vol.1.

[66] 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.

[67] C. Lindemann and O. P. Waldhorst, "A distributed search service for peer-to-peer file sharing in mobile applications," in *P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing*. Washington, DC, USA: IEEE Computer Society, 2002, p. 73.

[68] 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)*.

[69] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[70] O. Ore, "Note on hamiltonian circuits," *American Mathematical Monthly*, p. 55, 1960.

[71] [Online]. Available: http://www.isi.edu/nsnam/ns

[72] F. Bersani and H. Tschofenig, "The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method," RFC4764, January 2007.

[73] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and E. H. Levkowetz, "Extensible Authentication Protocol (EAP)," RFC3748, June 2004.

[74] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence, "Generic AAA Architecture," RFC2903, August 2000.

[75] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method," RFC2865, June 2000.

[76] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko, "Diameter Base Protocol," RFC3588, September 2003.

[77] IEEE, "Ieee standard for information technology telecommunications and information exchange between systems local and metropolitan area networks specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 6: Medium access control (mac) security enhancements," *IEEE Standards 802.11i-2004*, July 2004.

[78] ——, "Standards for local and metropolitan area networks: Standard for port based network access control," *IEEE Draft P802.1X/D11*, March 2001.

[79] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1994.