Chapter 2
Flat DHT Routing Topologies

Abstract In this chapter, we survey existing classical DHTs grouped according to their topologies. We describe basic architectures for efficient overlay routing and corresponding classes of graphs. We start with introducing Content Addressable Network (CAN) as an example of Torus topology. It is followed by Chord, Kademlia and Accordion as DHTs using the Ring topology. Pastry, Tapestry and Bamboo DHTs are grouped under the PRR tree topology. Trie and balanced trees are represented by P-Grid, skip graphs. Finally, we present several DHTs that are using De Bruijn and Kautz graphs, Butterfly and O(1)-hop topologies. These topologies differ in how they collect local information in a DHT node about the global network. The next chapter will show how these architectures can be generalized using hierarchical approach.

2.1 Torus

2.1.1 Content Addressable Network

Content Addressable Network (CAN) [27] was one of four DHTs introduced in 2001. In CAN, each node is responsible for an area of key identifier space, called a zone. In addition, it maintains information on nodes responsible for adjacent zones. Request including insert, lookup or delete operations are routed hop-by-hop between zones until the target is reached. The main design goals of CAN where scalability (nodes maintain a limited state), fault tolerance (requests can be routed around failures) and P2P nature (distributed architecture with no single point of failure).

The CAN is organized as a d-dimensional torus with Cartesian coordinates. This is a purely logical construct where the coordinate space is partitioned between participating nodes. Figure 2.1 illustrates a two-dimensional coordinate space which

D. Korzun and A. Gurtov, *Structured Peer-to-Peer Systems: Fundamentals of Hierarchical Organization, Routing, Scaling, and Security*, DOI 10.1007/978-1-4614-5483-0_2,

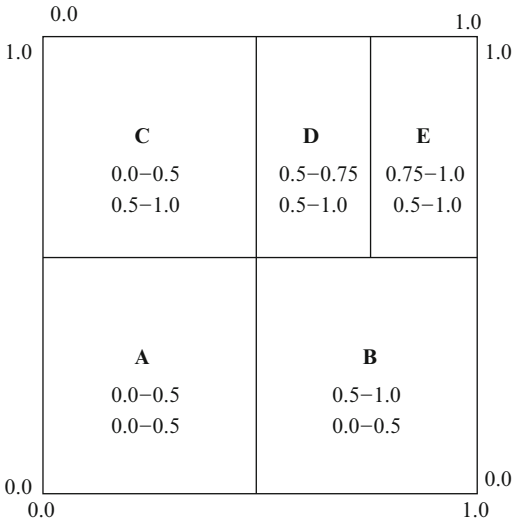

Fig. 2.1 The structure of 2D CAN space

is shared among several nodes. For simplicity, the space is drawn as a plane while in fact it wraps up at the edges, forming a torus. Many paths exist between source and destination and greedy routing is used to reach the destination.

Considering a d-dimensional CAN composed on n nodes, on the average each node would have $2d$ neighbors and the average path length will be $(d/4)(n^{1/d})$. Adding a new node does not affect the node state but increases the average path length as $O(n^{1/d})$.

To reduce the routing delay, CAN maintains several independent coordinate spaces called *realities*. If a node belongs to *r* realities, it has *r* independent neighbor sets and *r* coordinate points in separate zones. Multiple realities improve routing reliability, as a different reality can be used in case routing fails due to node departure within one reality. Furthermore, using different realities routing tables reduces the average routing path length, as distant locations can be often reached with one hop. Both increasing dimensions and number of realities reduces the average path length, but increasing the number of dimensions also increases per-node neighbor state.

Other techniques to improve the performance of CAN include using multiple hash functions to map a single key to several points of the coordinate space and hence increase the availability. Other enhancement is allocating zones taking the node physical location into account. In classic CAN, one logical hop can stretch over many IP hops so that two CAN neighbors can be located in different continents.

The CAN design was evaluated through simulations up to 260,000 nodes. With described enhancements, "knobs on full" CAN can route with less than twice the native IP latency between nodes.

2.2 Ring

The ring is a popular topology for organizing nodes in a DHT. In this section we describe three classic DHTs that use ring topology: Chord, Kademlia and Accordion. Chord represents a ring DHT which is linked in the clockwise direction. Although Kademlia is using a tree for routing, with binary identifiers its XOR metric is similar to the prefix metric and the ring topology. Therefore, Kademlia represents a bi-directional version of Chord.

2.2.1 Chord

Chord [32] is one of the first and most influential DHTs. It was developed by Ian Stoica with colleagues at MIT in 2001. In Chord, node a organized in a circle of up to 2^m nodes, with each node having an ID from that space. IDs of keys and nodes are of m bit length. The IDs are obtained using consistent hashing with SHA-1 algorithm and are uniformly distributed across the identifier space. This ensures even distribution of keys over nodes, and node place within the ring. The ID of a node is a hash of its IP address, and the ID of the key is a hash of its attribute, such as a file name.

Design of Chord aims at achieving following properties. *Load balancing* assumes even distribution of keys over nodes. *Scalability* requires that the lookup costs grow slower than the number of nodes, thus allowing large system construction. *Decentralization* is the main characteristics of P2P systems in general, and in Chord all nodes are equal without introducing single failure points. *Availability* is guaranteed by automatic maintenance of the node organization in response to node joins and leaves.

The ring of nodes is linked together in clockwise direction. The predecessor is the first node in counter-clockwise direction and the successor is a next node in clockwise direction. To ensure operation in the presence of node joins and departures, a node keeps information on r nodes succeeding it in the ring. A key k is assigned to successor of node with ID equal or greater than k.

```
join():
// create a new Chord ring.
n.create()
    predecessor = nil;
    successor = n;

// join a Chord ring containing node n
n.join(n')
    predecessor = nil;
    successor = n'.find_successor(n);
```

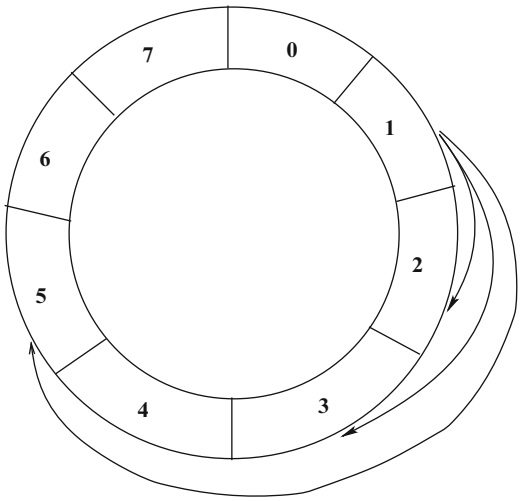

Fig. 2.2 The structure of Chord ring

In naive lookup routing, the ring can be traversed sequentially until a node responsible for a key is located. This is quite slow as about half of the ring would need to be traversed on the average. To accelerate the lookup progress, each node keeps a table of *fingers*, each pointing to a successor node of $(n+2^{i-1}) \mod 2^m$, where m is the number of fingers per node, n is the node ID, and i is the finger table index. Using fingers, the lookup time can be significantly reduced, as only $O(\log N)$ nodes need to be contacted to locate a key in a ring of N nodes. Using each finger halves the remaining distance to the destination node. Figure 2.2 illustrates the position of the fingers. Each entry in the finger table includes the node ID, its IP address and port number.

```
find_successor():
// ask node n to find the successor of id
n.find_successor(id)
    if (id in (n, successor])
        return successor;
    else
        n' = closest_preceding_node(id);
        return n'.find_successor(id);

// search the local table for the highest predecessor
  of id
n.closest_preceding_node(id)
    for i = m downto 1
        if (finger[i] in (n, id))
            return finger[i];
    return n;
```

The lookups can be implemented for recursive or iterative routing. With recursive approach, each new node in the lookup path queries the next hop. With iterative approach, the node that initiated the lookup receives answers from intermediate nodes in the path and itself performs the next hop lookup. The recursive approach reduces the number of required messages to perform a key lookup, while iterative approach is more robust in the presence of nodes joins and leaves. The simulator in the original Chord paper implemented the iterative approach.

```
find_successor():
// ask node n to find the successor of id
n.find_successor(id)
    if (id in (n, successor])
        return successor;
    else
        n' = closest_preceding_node(id);
        return n'.find_successor(id);

// search the local table for the highest predecessor
  of id
n.closest_preceding_node(id)
    for i = m downto 1
        if (finger[i] in (n, id))
            return finger[i];
    return n;
```

Nodes can join the ring and leave dynamically. When a new node joins, it takes over a part of keys that belong to its successor. When a node leaves the ring, all its keys are migrated to its successor. To ensure consistency, each node periodically runs a maintenance protocol to update its successor and finger table.

2.2.2 Kademlia

Kademlia [21] is a DHT introduced in 2002. It reduces the number of administrative messages needed to maintain routing tables, as nodes learn about the peers during lookup operations. The lookups are made in parallel and non-blocking way to minimize the delays. Its performance characteristics are formally proven. Kademlia is widely used for example by BitTorrent clients for distributed tracking of torrents.

Kademlia uses 160-bit IDs for nodes and keys. The keys/value pairs are stored on nodes with sufficiently close ID. XOR metric is used to determine a distance between nodes. Thanks to the metric, parallel queries to any node within a range nearby the target node can be sent and also learn new routes.

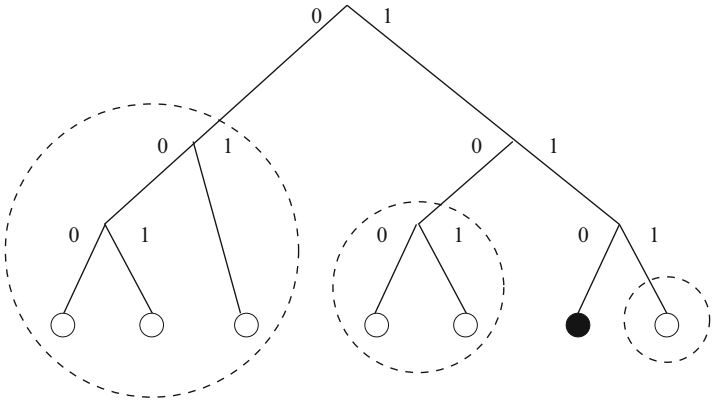

Fig. 2.3 The Kademlia tree for node 110

Kademlia does not separate the routing process into phases but uses the same mechanism until the target node is reached. On each step, the XOR distance to target node is roughly halved. XOR metric $x \oplus y$ is unidirectional that ensures convergence of requests towards the target.

Nodes in Kademlia store information about other nodes in a form of (Node ID, IP address, UDP port) of a distance in the range of 2^i and 2^{i+1}, where i is $0, \ldots, 160$. These lists form *k-buckets* sorted by the time of latest communication. Buckets for smaller i tend to be empty, while for larger i will be of size k (e.g., 20) which is a system parameter. The buckets are filled in whether a new message arrives from another node. Live nodes are never removed from the buckets, only nodes that fail to respond to pings are replaced by new nodes. That ensures that oldest live nodes remain as long as possible in the buckets, as those nodes are least probable to fail also in future.

Figure 2.3 illustrated a Kademlia routing tree for node with ID 110.

2.2.3 Accordion

Accordion [17] fills a gap between DHTs that provide $O(1)$ and $O(\log N)$ lookup times. Achieving $O(1)$ lookups is possible in small and relatively stable DHTs by maintaining a nearly complete state in each DHT node about other nodes ID and addresses. For larger and more dynamic networks, their maintenance traffic can grow prohibitively fast. Traditional DHTs such as Chord providing $O(\log N)$ lookup latency scale better, but require multiple hops to reach the destination that increases the response time. Accordion attempts to tune the DHT parameters to achieve best performance under a constraint of bandwidth budget defined by user. That frees users from the need to manually configure multiple DHT parameters and enables it to operate efficiently under varying network size, churn and lookup rates.

Accordion uses a Chord ring structure with consistent hashing, successor list, and the join protocol. The lookup find the node which ID follows the key. Accordion uses greedy routing approach, each node forwards the lookup request to the node with closest ID preceding the key. The process continues until the predecessor of the key hosting node is reached. The predecessor replies directly to the node that initiated the request.

Accordion explores multiple paths in parallel, sending as many lookups as the bandwidth budget allows. Parallel lookups are preferable compared to explicit probing, as those help to avoid timeouts due to dead nodes and also discover new nodes in a DHT. Each time a node forwards a lookup request to the next hop node, that node returns a list of its neighbors until the key ID. This way, the nodes can expand their routing tables to satisfy the main goal of Accordion, providing the minimum latency given the churn rate and the bandwidth budget.

2.3 PRR Trees

Plaxton, Rajaraman, and Richa proposed a PRR tree in 1997 [26]. PRR trees allow routing in a distributed publication system. They considered static network topology and did not provide a real-world implementation of the system. Later on, PRR trees were adopted by Tapestry and Pastry DHTs that also include management of dynamic membership of nodes in a tree.

Figure 2.4 shows neighbors of a node in PRR tree. Each node has $O(b * \log_b(N))$ neighbors in its routing table on several levels. A neighbor on ith level shares i digits with node ID. Hence, L0 node shares no common digits and L3 node shares three prefix digits. A node with ID closest to the key is the owner of that key.

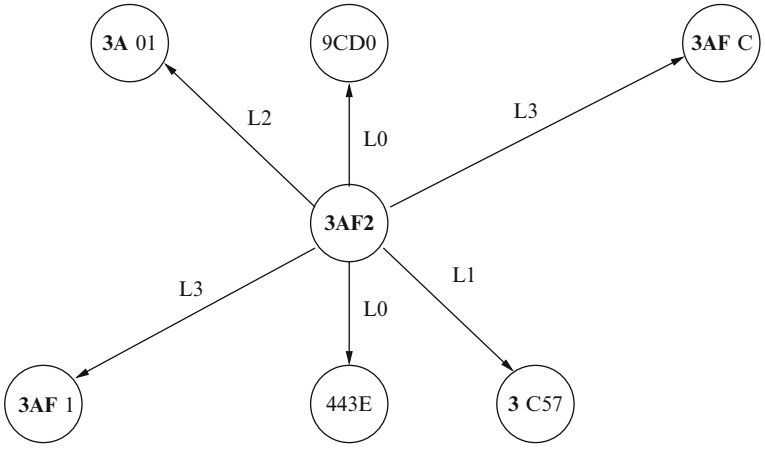

Fig. 2.4 Routing table of a PRR tree

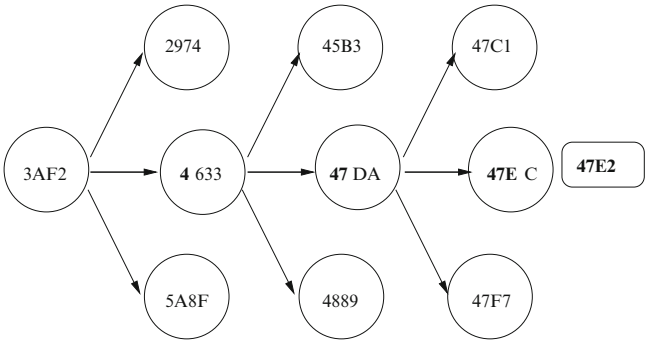

Fig. 2.5 Routing process in a PRR tree

Figure 2.5 shows routing process initiated from node with ID $3AF2$ to a key with ID $47E2$. The routing goal is to find a node with the longest matching prefix of digits. The query proceeds to node 4633 first since it shares the first digit with destination. The second hop is node with ID $47DA$ with two digits and finally node $47EC$ which is numerically closest to the key ID and hence responsible for it.

2.3.1 Pastry

Pastry [29] is one of early DHTs introduced in 2001. Like other DHTs, it distributes keys among participating nodes with IDs, routing a request to the responsible node in $O(\log N)$ hops in a network of N nodes. The destination node is a live node with ID numerically closest to the key ID.

A special feature of Pastry is proximity-aware routing. Each Pastry node keeps information on k nodes closest to it in the identifier space. Such nodes are likely to be spread geographically which could produce very long hops in terms of network latency. Pastry attempts to choose the next hop node which is closest by the routing metric, such as RTT delay.

The basic topology of Pastry is similar to Chord. Node IDs of 128 bits are arranged into a ring. The node ID can be generated as a hash of node's public key or its IP address, which ensures random positioning of the node within the ring.

The Pastry node maintains a routing table, a leaf set and a neighborhood set. Instead of Chord's fingers, a PRR tree is used to route requests to nodes within the ring.

The Pastry authors have evaluated it in an emulated network of 100,000 nodes, confirming its scalability and self-organization capabilities, and ability to take advantage of geographically close nodes.

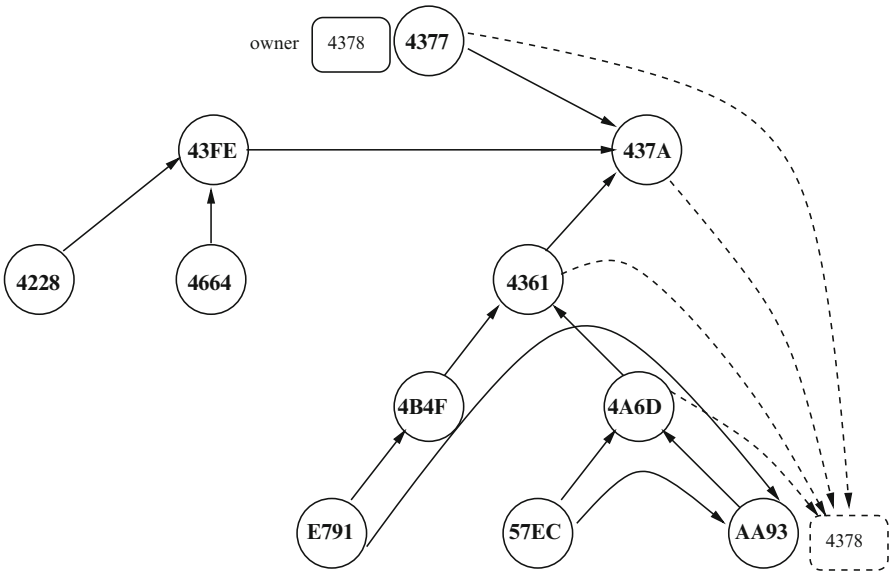

Fig. 2.6 Tapestry PRR topology combined with location mappings

2.3.2 Tapestry

Tapestry [36] is one of first DHTs introduced in 2001. It represents a structured peer-to-peer overlay network that guarantees key location in the system when node do not fail badly. Like Pastry, it is based on PRR trees. Tapestry authors use an alternative term to DHT, Decentralized Object Location and Routing (DOLR). A special feature of Tapestry is ability to exploit locality in accessing object replicas.

Figure 2.6 shows Tapestry topology based on PRR trees. Lookup requests are forwarded to the closest copy of an object using location mappings created when an object was published. In the figure, node with ID 4377 is responsible for the key 4378. The key was published by a request to node ID *AA*93 which routed it towards the owner node. At each hop until the destination along the PRR tree, a link to the local copy of the key at node *AA*93 is created. Therefore, a lookup request from a nearby node 57*EC* gets immediately redirected to local copy from a node 4*A*6*D*, and likewise by node 4361 when request comes from node *E*791.

2.3.3 Bamboo

Bamboo [28] is a ring-based DHT that specifically focuses on efficient operation in the presence of high churn. Its authors evaluated the impact of churn on several DHTs and different strategies to cope with churn, including reactive vs periodic failure recovers, timeout calculation during lookups and choice of neighbors.

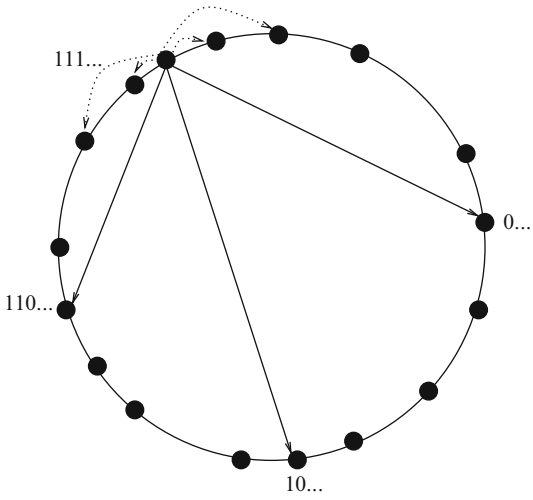

Fig. 2.7 Neighbor selection in Bamboo

In particular, the churn handling capabilities of Pastry and Chord were investigated. It was found that Pastry recovers poorly even under medium churn, the main reason being that Pastry uses reactive recovery (i.e., repairing failures as soon as they are detected) and thereby is subject to the problem of positive feedback cycles, where network link congestion causes repair packets to be sent, which in turn causes more congestion, mistaken conclusion of whether other neighbors are down, and eventually congestion collapse. On the other hand, the main problem with Chord (which uses periodic recovery) is that under churn, lookup latency increases substantially, which results from inaccurate timeout threshold calculations.

Based on these observations, Bamboo applies three techniques to address churn handling: periodic recovery, timeout calculation algorithms, and proximity neighbor selection (PNS) algorithms. PNS helps to reduce lookup latency by picking up neighbors close in terms of network latency at the expense of additional bandwidth use. Consequently, lower lookup latency leads to better timeout calculation.

The routing algorithm of Bamboo is borrowed from Pastry. It is illustrated in Fig. 2.7. Bamboo maintains the same geometry despite the churn. Two sets of pointers are present in a node, a leaf set to k neighbors before and after the node, as well as pointers to nodes with matching prefixes in PRR tree. Lookups are routed in $O(\log(N))$ hops even in the presence of a large number of broken links.

To cope with node failures, Bamboo uses periodic recovery like Chord. Periodically, a complete routing table is shared with one of a randomly chosen node from the leaf set. A node is considered dead and removed from the routing table after 15 timeouts, although routing to it stops after 5 timeouts. The general conclusion of Bamboo authors is that periodic recovery is well suited for high churn scenarios, while reactive recovery is appropriate when churn is low.

2.4 Tree

2.4.1 Trie

A trie, or prefix tree, is a data structure which name comes from the *retrieval*. In trie, the node position within the tree defines its key value, unlike in the binary search tree. All descendants of a node share a common prefix with a node. Figure 2.8 illustrates the structure of a trie containing a few English words. The character keys are most common for tries, although any ordered lists are suitable.

A trie has a nice property that all basic operations take nearly same time, because lookup, insertion or deletion is largely the same code. That enables tries to be more efficient than binary search trees and hash tables in most important properties. Most importantly, longest-prefix matching or finding a node with a longest prefix with a key is very efficient with a trie.

An example of use of distributed tries in a DHT [9] is described in Sect. 3.5. P-Grid [1, 5] is also using a distributed tree and is described in Sect. 4.6. Another example is ZIGZAG that provides P2P media streaming.

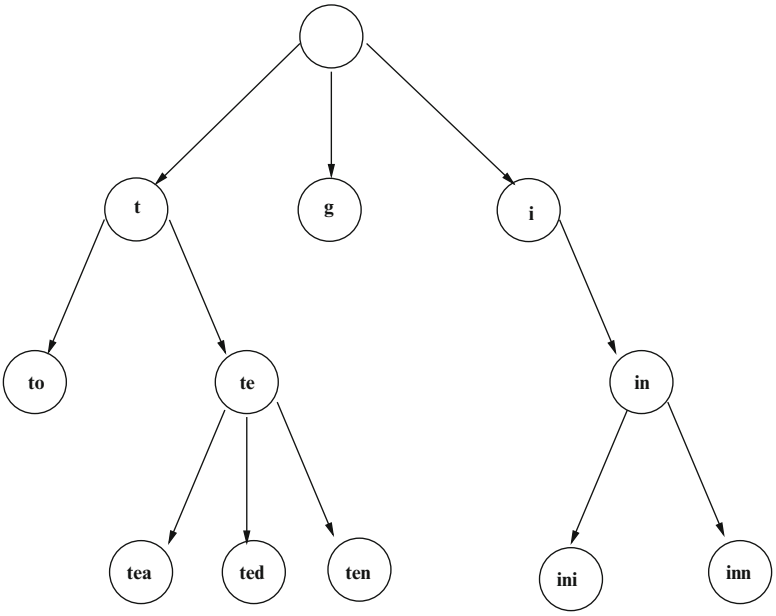

Fig. 2.8 Trie data structure

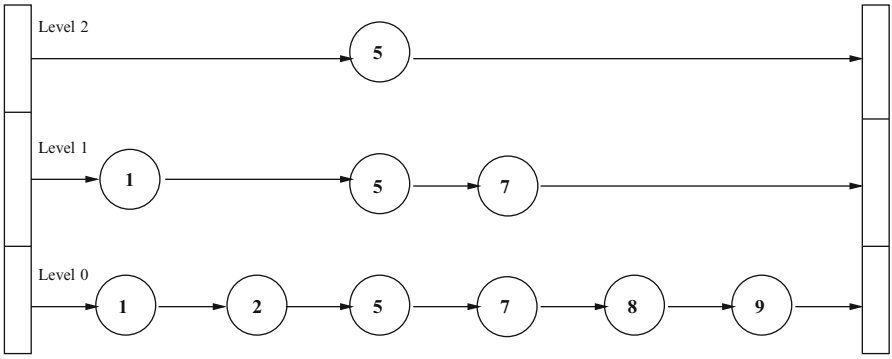

Fig. 2.9 Skip list data structure

2.4.2 Balanced Tree

Skip list is a data structure that consists of several linked lists that connect sorted items, as shown in Fig. 2.9.

Skip list was introduced by W. Pugh in 1990. Each higher list in the hierarchy jumps over multiple list items. The jump span can be selected randomly using a negative binomial distribution. The list item at $i-1$ level appears at i level with certain probability p, typically 0.5 or 0.25. Then the number of layers in a skip list is $\log(1/p)N$. By varying p it is possible to balance between the storage overhead versus the search time.

The search for a key starts from the top layer list to the right until the key is found or a greater key is located. In latter case, the search returns to the previous node and drops down one layer. The procedure is repeated until the key is found or a greater key is found on the bottom layer, which means the key is not present. Therefore the expected search time is $(\log(1/p)N)/p$.

Skip graphs [3, 4] and SkipNet [13, 14] are examples of DHTs using the skip list data structure. They are described in Sect. 4.5.

2.5 De Bruijn and Kautz Graphs

2.5.1 De Bruijn Graphs

De Bruijn graph of n-dimensions and m symbols is a directed graph of m^n vertices (nodes). Each vertex is a random sequence of any of m symbols of length n, repeating symbols allowed. A directed edge is formed to a neighbor vertex if its ID is formed by shifting one position to the left and adding a new arbitrary symbol.

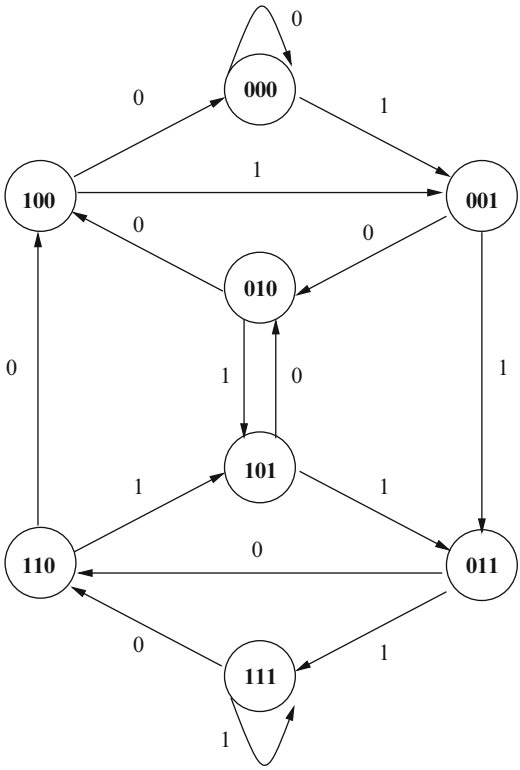

Fig. 2.10 De Bruijn graph for three dimensions and two symbols

Figure 2.10 illustrates a de Bruijn graph of three dimensions constructed with two symbols (0,1). It can be seen that each vertex has exactly m incoming and m outgoing edges. Each de Bruijn graph has an Eulerian path (which visits every edge exactly once) and Hamiltonian path (that visits each vertex exactly once).

Several DHTs utilize de Bruijn graphs in their topology, including Koorde, D2B, Distance-halving [24, 25], ODRI [18], and Broose [10].

Koorde [15] is a DHT that modifies Chord to follow the topology of de Bruijn graphs. Koorde means a chord in Dutch language. Even with two neighbors per node, Koorde is able to route requests in $O(\log N)$ hops. When users can tune up the number of neighbor per node up to $O(\log N)$, the number of hop decreases to $O(\log N / \log logN)$. Naturally, this comes at the cost of increasing maintenance overhead. Koorde inherits some algorithms, such as handling concurrent joins, from Chord.

D2B [8] is another DHT using de Bruijn graphs, but in probabilistic manner. It ensures with high probability that the nodes have $O(1)$ neighbors. D2B requires special form of node identifiers to construct a de Bruijn graph.

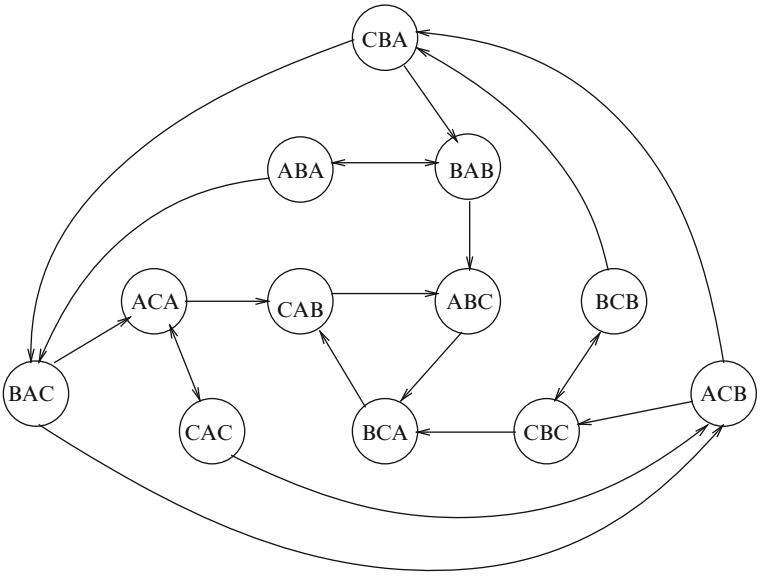

Fig. 2.11 The Kautz graph (2,2)

2.5.2 Kautz Graph

Kautz graph is a directed graph where vertices are labeled as all possible combinations of $M+1$ distinct symbols under condition that adjacent symbols are not the same. M is called the degree of the graph. The dimension of the graph, N defines the length of each vertex label $(N+1)$. An edge is created to a vertex if its label starts with a different symbol with regard to the source vertex. Figure 2.11 illustrates the Kautz graph for $M=2$ and $N=2$.

The Kautz graphs are closely related to De Bruijn graphs. Likewise, they posses Eulerian and Hamiltonian cycles. The Kautz graph has the smallest diameter of any directed graph with degree M and V vertices. The number of vertices in a graph with degree M equals to $(M+1)M^{N+1}$. The edges of a K_M^{N+1} graph correspond to vertices of K_M^{N+2} graph.

Following DHTs use the Kautz graph as basic topology: FISSIONE [16], Moore [11], BAKE [12], and SKY [35]. They are described in Sect. 4.7.

2.6 Butterfly

Viceroy [19] utilizes the butterfly topology with constant expected number of neighbors. The node degree is less than $O(\log N)$. Figure 2.12 illustrates the multi-layer butterfly topology. The example network contains eight nodes with IDs

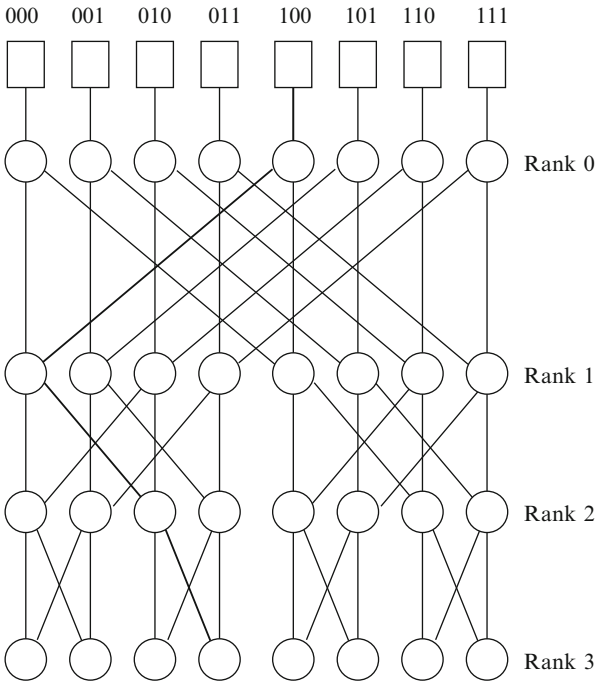

Fig. 2.12 The butterfly network

000..111. The nodes are linked at four layers, increasingly closer to neighbors. Thick line shows a path from node 100 to node 011. The lookup reaches the destination after three hops, which represents the worst case scenario in a stable network.

Viceroy estimates the size of network to select placement of nodes at several layers. Having a constant node degree helps in managing the cost of joins and leaves, since only a small number of neighbors need to be informed of leave, pinged for keep-alive, or exchanged state information. A small number of neighbors also helps to reduce the number of incoming and outgoing connections for smaller overhead. A possible draw back of this is difficulty in maintaining the topology during high churn.

Other DHTs using butterfly topology include CRN [6, 30], Ulysses [34], Cycloid [31], Pappilon [2], and Mariposa [20]. They are described in Sect. 4.2.

2.7 $O(1)$-Hop

Researchers proposed several DHTs that are able to perform lookups with constant time irrespective of the network size. Such DHTs, known as $O(1)$ hop DHTs include OneHop [7], 1h-Calot [33], and D1HT [22]. Their comparison can be found in [23] and [33]. OneHop is described in detail in Sect. 7.3.1.4 among other hierarchical DHTs.

Most of DHTs presented in this chapter use multi-hop routing which requires less maintenance bandwidth and storage costs compared to one-hop DHTs. However, the use of multi-hop DHTs for performance-sensitive applications is problematic due to variable lookup delay. In some cases, one-hop DHTs can consume even less bandwidth compared to multi-hop DHTs when lookup frequency is high because multi-hop DHTs spend more on hop-by-hop forwarding. In general, bandwidth in future networks tends to grow much faster than the latency reduces, therefore technology solutions with small latency have clear potential.

To keep maintenance traffic within acceptable boundaries, D1HT proposes to combine several events, such as peer joins or leaves, within one message. That requires buffering event announcements until a sufficient number is collected to fill a packet. This naturally increases the delay in which other nodes learn about the system changes. To balance between maintenance traffic and event propagation latency, the maximum event holding time is determined from the application requirements. Typically it is assumed that at least 99 % of application requests should be resolved within one hop.

Both D1HT and 1h-Calot use logarithmic tree to distribute updates in a DHT. 1h-Calot uses peer IDs to build the logarithmic tree, while D1HT uses TTL of messages. D1HT is reactive in detecting events through maintenance message but 1h-Calot sends periodic heartbeats with proactive approach. D1HT buffers the events to reduce the traffic, but 1h-Calot requires at least one message and its acknowledgment per event and all DHT nodes.

2.8 Summary

We introduced basic classes of DHT topologies and described popular DHTs representing these classes. The first classic DHTs of CAN, Chord, Pastry and Tapestry are based on d-dimensional torus, a ring, and PRR tree topologies. We also introduced De Bruijn and Kautz graphs and their use in DHTs, skip lists, trie, and butterfly topologies. The chapter is concluded by description of $O(1)$-hop DHTs that aim at resolving most lookups with minimal latency.

References

1. Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R.: P-Grid: a self-organizing structured P2P system. SIGMOD Rec. **32**(3), 29–33 (2003). doi: http://doi.acm.org/10.1145/945721.945729
2. Abraham, I., Malkhi, D., Manku, G.S.: Papillon: Greedy routing in rings. In: DISC '05: Proceedings of 19th International Conference on Distributed Computing. Lecture Notes in Computer Science, vol. 3724, pp. 514–515. Springer, Berlin (2005)
3. Aspnes, J., Shah, G.: Skip graphs. In: SODA '03: Proceedings of 14th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 384–393. Society for Industrial and Applied Mathematics (2003)

4. Aspnes, J., Wieder, U.: The expansion and mixing time of skip graphs with applications. In: SPAA '05: Proceedings of 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures, pp. 126–134. ACM, New York (2005).doi: http://doi.acm.org/10.1145/1073970.1073989
5. Datta, A., Girdzijauskas, S., Aberer, K.: On de bruijn routing in distributed hash tables: There and back again. In: IEEE P2P '04: Proceedings of 4th International Conference on Peer-to-Peer Computing, pp. 159–166. IEEE Computer Society (2004).doi: http://dx.doi.org/10.1109/P2P.2004.29
6. Fiat, A., Saia, J.: Censorship resistant peer-to-peer content addressable networks. In: SODA '02: Proceedings of 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 94–103. Society for Industrial and Applied Mathematics (2002)
7. Fonseca, P., Rodrigues, R., Gupta, A., Liskov, B.: Full-information lookups for peer-to-peer overlays. IEEE Trans. Parallel Distrib. Syst. **20**(9), 1339–1351 (2009)
8. Fraigniaud, P., Gauron, P.: D2B: a de Bruijn based content-addressable network. Theor. Comput. Sci. **355**(1), 65–79 (2006). doi: http://dx.doi.org/10.1016/j.tcs.2005.12.006
9. Freedman, M.J., Vingralek, R.: Efficient peer-to-peer lookup based on a distributed trie. In: Revised Papers from 1st International Workshop on Peer-to-Peer Systems (IPTPS '01), pp. 66–75. Springer, Berlin (2002)
10. Gai, A.T., Viennot, L.: Broose: A practical distributed hashtable based on the De-Bruijn topology. In: Proceedings of IEEE 4th International Conference on Peer-to-Peer Computing (P2P '04), pp. 167–164. IEEE Computer Society (2004). doi: http://dx.doi.org/10.1109/P2P.2004.10
11. Guo, D., Wu, J., Chen, H., Luo, X.: Moore: An extendable peer-to-peer network based on incomplete Kautz digraph with constant degree. In: Proceedings of IEEE INFOCOM'07, pp. 821–829. IEEE (2007)
12. Guo, D., Liu, Y., Li, X.Y.: BAKE: A balanced Kautz tree structure for peer-to-peer networks. In: Proceedings of IEEEINFOCOM'08, pp. 2450–2457. IEEE (2008)
13. Harvey, N.J.A., Munro, J.I.: Deterministic SkipNet. Inf. Process. Lett. **90**(4), 205–208 (2004). doi: http://dx.doi.org/10.1016/j.ipl.2004.01.019
14. Harvey, N.J.A., Jones, M.B., Saroiu, S., Theimer, M., Wolman, A.: SkipNet: a scalable overlay network with practical locality properties. In: USITS'03: Proceedings of 4th USENIX Symposium on Internet Technologies and Systems. USENIX Association (2003)
15. Kaashoek, M.F., Karger, D.R.: Koorde: A simple degree-optimal distributed hash table. In: IPTPS '03: Proceedings of 2nd International Workshop on Peer-to-Peer Systems. Lecture Notes in Computer Science, vol. 2735, pp. 98–107. Springer, Berlin (2003)
16. Li, D., Lu, X., Wu, J.: FISSIONE: a scalable constant degree and low congestion DHT scheme based on Kautz graphs. In: Proceedings of IEEE INFOCOM'05, pp. 1677–1688. IEEE (2005)
17. Li, J., Stribling, J., Morris, R., Kaashoek, M.F.: Bandwidth-efficient management of DHT routing tables. In: Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI '05), pp. 99–114 (2005)
18. Loguinov, D., Kumar, A., Rai, V., Ganesh, S.: Graph-theoretic analysis of structured peer-to-peer systems: Routing distances and fault resilience. IEEE/ACM Trans. Netw. **13**(5), 1107–1120 (2005)
19. Malkhi, D., Naor, M., Ratajczak, D.: Viceroy: a scalable and dynamic emulation of the butterfly. In: PODC '02: Proceedings of 21st Annual Symposium on Principles of Distributed Computing, pp. 183–192. ACM, New York (2002). doi: http://doi.acm.org/10.1145/571825.571857
20. Manku, G.S.: Routing networks for distributed hash tables. In: PODC '03: Proceedings of 22nd Annual Symposium on Principles of Distributed Computing, pp. 133–142. ACM (2003). doi: http://doi.acm.org/10.1145/872035.872054
21. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In: IPTPS '02: Proceedings of 1st International Workshop on Peer-to-Peer Systems. Lecture Notes in Computer Science, vol. 2429, pp. 53–65. Springer, New York (2002)

22. Monnerat, L.R., Amorim, C.L.: D1HT: a distributed one hop hash table. In: Proceedings of 20th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2006). IEEE Computer Society (2006). doi: http://doi.ieeecomputersociety.org/10.1109/IPDPS.2006.1639278
23. Monnerat, L.R., Amorim, C.L.: Peer-to-peer single hop distributed hash tables. In: Proceedings of IEEE Globecom'09, pp. 4250–4257, IEEE (2009)
24. Naor, M., Wieder, U.: A simple fault tolerant distributed hash table. In: IPTPS '03: Proceedings of 2nd International Workshop on Peer-to-Peer Systems. Lecture Notes in Computer Science, vol. 2735, pp. 88–97. Springer, New York (2003)
25. Naor, M., Wieder, U.: Novel architectures for P2P applications: the continuous-discrete approach. ACM Trans. Algorithms **3**(3), 37 (2007). doi: http://doi.acm.org/10.1145/1273340.1273350
26. Plaxton, C.G., Rajaraman, R., Richa, A.W.: Accessing nearby copies of replicated objects in a distributed environment. In: Proceedings of 9th Annual Symposium on Parallel Algorithms and Architectures (SPAA '97), pp. 311–320 (1997)
27. Ratnasamy, S., Handley, P.F.M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proceedings of ACM SIGCOMM'01, pp. 161–172. ACM, New York (2001)
28. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a DHT. In: Proceedings of the USENIX Annual Technical Conference (2004)
29. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Middleware'01: Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms. Lecture Notes in Computer Science, vol. 2218, pp. 329–350. Springer, Berlin (2001)
30. Saia, J., Fiat, A., Gribble, S.D., Karlin, A.R., Saroiu, S.: Dynamically fault-tolerant content addressable networks. In: IPTPS '01: Revised Papers from 1st International Workshop on Peer-to-Peer Systems, pp. 270–279. Springer, New York (2002)
31. Shen, H., Xu, C.Z., Chen, G.: Cycloid: a constant-degree and lookup-efficient p2p overlay network. Perform. Eval. **63**(3), 195–216 (2006). doi: http://dx.doi.org/10.1016/j.peva.2005.01.004
32. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for Internet applications. IEEE/ACM Trans. Netw. **11**(1), 17–32 (2003)
33. Tang, C., Buco, M.J., Chang, R.N., Dwarkadas, S., Luan, L.Z., So, E., Ward, C.: Low traffic overlay networks with large routing tables. SIGMETRICS Perform. Eval. Rev. **33**(1), 14–25 (2005). doi: http://doi.acm.org/10.1145/1071690.1064216
34. Xu, J., Kumar, A., Yu, X.: On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. IEEE J. Sel. Areas Commun. **22**(1), 151–163 (2004)
35. Zhang, Y., Lu, X., Li, D.: SKY: efficient peer-to-peer networks based on distributed Kautz graphs. Sci. China Ser. F Inf. Sci. **52**(4), 588–601 (2009)
36. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.D.: Tapestry: A resilient global-scale overlay for service deployment. IEEE J. Sel. Areas Commun. **22**(1), 41–53 (2004)

http://www.springer.com/978-1-4614-5482-3

Structured Peer-to-Peer Systems
Fundamentals of Hierarchical Organization, Routing, Scaling, and Security
Korzun, D.; Gurtov, A.
2013, XXII, 366 p., Hardcover
ISBN: 978-1-4614-5482-3