TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Netzarchitekturen und Netzdienste

Providing Efficient Key Based Routing for Multiple Applications

Pengfei Di

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Uwe Baumgarten
Prüfer der Dissertation:	
	1. Univ.-Prof. Dr. Georg Carle
	2. Univ.-Prof. Dr. Hans Michael Gerndt
	3. TUM Junior Fellow Dr. Thomas Fuhrmann

Die Dissertation wurde am 30.12.2010 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 12.03.2014 angenommen.

Abstract

In this dissertation, I present an overlay network protocol, Lite-Ring, which aims at providing efficient Key Based Routing (KBR) services to multiple applications, both in the Internet and in wireless ad-hoc networks. The key idea behind Lite-Ring is its predicable address assignment scheme. With this address scheme, a Lite-Ring node can almost exactly locally calculate the ID of the node that is responsible for a given key. By leveraging a public Distributed Hash Table (DHT) service, a Lite-Ring node needs only to maintain O(1) state. Hence, Lite-Ring is more efficient than conventional structured overlay protocols, which have to iteratively route the message to the destination node hop by hop.

For the application of Lite-Ring in wireless ad-hoc networks, I also present a novel system architecture as well as some optimizations with cross-layer designs such as Proximity Neighbor Selection (PNS), DHT caching, and link-layer broadcast. Simulations demonstrate the outstanding performance of these optimizations.

II

Zusammenfassung

In dieser Dissertation wird das Overlay-Netzwerk-Protokoll "Lite-Ring" vorgestellt, das sowohl im Internet als auch in Ad-hoc-Netzen einsetzbar ist. Aufgrund seiner speziellen Adressvergaberegeln erlaubt es den teilnehmenden Knoten, den Zielknoten einer Nachricht (mit einem beliebigen Schlüssel) lokal zu berechnen. Durch die Nutzung eines öffentlichen Distributed Hash Table (DHT) Dienstes, braucht ein Lite-Ring-Knoten nur O(1) Zustände zu speichern. Das macht Lite-Ring effizienter als herkömmliche strukturierte Overlay-Protokolle, die die Nachricht iterativ an den Zielknoten routen müssen.

Für die Anwendung des Lite-Ring-Protokolls in drahtlosen Ad-hoc-Netzen, wird eine neuartige Systemarchitektur vorgestellt. Weiterhin werden einige Optimierungen mit Cross-Layer-Designs präsentiert, die sich mit Proximity Neighbor Selection (PNS), DHT Caching und Link-Layer Broadcast befassen. Simulationen zeigen die herausragende Leistungsfähigkeit des Protokolls in Verbindung mit den Optimierungen.

Contents

Chapter 1

Introduction

1.1 Background

In the last decade, Peer-to-Peer (P2P) technologies have become overwhelmingly popular and P2P applications have dominated the major part of the Internet traffic [108]. Cisco has even forecast that P2P traffic will further increase by 23% every year in the next half decade [6]. The success of P2P applications can be attributed to their resource sharing strategies among different peers, and such resource sharing strategies are typically based on certain virtual overlays that are built by the peers in a self-organizing manner.

Originally, P2P applications were merely developed for (multimedia) file-sharing. In the meantime, they have been applied in wide areas, such as content distribution, Internet telephony and video streaming. With the increment of network connectivity, which happened not only for personal computers but also for small equipments, such as cell phones and embedded devices, P2P concept has also been applied in wireless ad-hoc networks.

Nowadays, various P2P applications could run on one host simultaneously. It is common that a P2P application builds its own overlay independently, regardless of the similar service that may be provided by other P2P applications. In order to keep high performance, each P2P application has to store a sizable state, which results in a considerable maintenance overhead.

1.2 Preliminary

According to the construction rules of the virtual overlays, P2P applications can be categorized into structured and unstructured P2P application. However, due to the benefits of the structured overlays, like consistent hashing [67] and efficient lookup, quite a few unstructured P2P applications, such as BitTorrent [27] and eMule [3], have also introduced structured overlay into their systems.

Despite the diversity of P2P applications, Dabek et al. [36] found that many of these apparently different P2P applications, such as DHT, Application Layer Multicast (ALM) and Decentralized Object Location and Routing (DOLR), can be built upon a common Application Programming Interface (API): the Key Based Routing (KBR) interface. The basic KBR API primitive is `route(key, value)`, where the application dependent `key` is consistently mapped to an

overlay instance. Depending on the application type, the received `value` is operated accordingly by the overlay instance. These endpoint operations may be more complex than just the storage and retrieval of the `value`, as a primitive DHT does. For instance, a distributed database may evaluate a non-trivial query, and an online game may perform a non-trivial update of its distributed state.

Due to the increasing popularity of P2P applications and the costly maintenance overhead of the individual overlays, the conventional solution (one full-fledged overlay per application) of providing KBR service is no more feasible. Particularly, P2P applications have been also introduced into wireless ad-hoc networks, where the network capacity is crucial to the maintenance overhead. Hence, a challenging question arises: *How can we efficiently provide generic KBR service to a potentially large set of applications in the Internet as well as in wireless ad-hoc networks?*

In this dissertation, I introduce a novel solution: Lite-Ring[1]. With light-weight overlays, Lite-Ring provides the same KBR services to multiple applications, as many conventional full-fledged structured overlays do, but in a much more efficient manner.

1.3 Structure of this Work

This work is composed of four major parts. In the first part, some foundations are outlined. They focus on P2P overlay networks in the Internet, routing protocols in wireless ad-hoc networks and some resent efforts on KBR service in the network layer.

In the second part, the Lite-Ring protocol is discussed in detail. Lite-Ring provides the generic KBR service to multiple applications with light-weight overlays by leveraging a public DHT service. From the application view, Lite-Ring is similar to many other structured overlay protocols. However, the difference to them lies in two factors: 1) Lite-Ring uses a predictable address assignment scheme. Hence it is able to estimate the destination nodes for arbitrary keys by local calculation. 2) Lite-Ring shifts the complicated and costly optimization jobs to the public DHT, such as the adaptation of the overlay towards the underlying network. These features enable Lite-Ring to provide efficient KBR services to a potentially large number of applications. Analysis and simulation demonstrate its advantages in smaller local state and less maintenance overhead comparing to the other approaches.

The third part is devoted to the construction of the Lite-Ring system, which aims at providing KBR services in wireless ad-hoc networks. The Lite-Ring system includes not only Lite-Ring modules that implement the Lite-Ring protocol, but also a DHT module and a Scalable Source Routing (SSR) module. To my best knowledge, the Lite-Ring system is the first exploratory study on providing KBR services for different applications in wireless ad-hoc networks. Some initial simulations also confirm its applicability.

In the fourth part, several cross-layer optimizations for the Lite-Ring system are proposed. These are Proximal Neighbor Selection (PNS) in the Lite-Ring module, intermediate caching in the DHT module and link-layer broadcast in the

[1] Lite-Ring stands for a light-weight ring-like KBR overlay.

SSR module. The simulation results demonstrate the outstanding performance of these optimizations.

Part I

Foundations

Chapter 2

Peer-to-Peer Networks

The term P2P means that the participants of the network have the similar roles and interact with each other in an "ad-hoc" manner, in contrast to the traditional "client-server" manner. With such an "ad-hoc" principle, the participants in a P2P network share their resource, which makes the network scalable. For example, Skype reported that it had 15 millions concurrent users in 2008 [5].

Besides the scalability, decentralization is also an important aspect of P2P applications. Decentralization makes a P2P network robust, i. e. the removal of some nodes in a P2P network will not cause significant impact on the network. For those fully decentralized P2P applications, there exists no central component, thus no single point of failure.

Most P2P applications are deployed at the hosts in the Internet, and they reside typically on top of the transport layer. There exists usually a virtual overlay network inherently in the P2P application. The connections between hosts in a P2P application are the *overlay links*. Depending on the rule of the generation of the overlay links, the virtual overlays can be classified into unstructured and structured overlays.

2.1 Unstructured Overlay Networks

In unstructured overlay networks the peers connect without special constrains on the generation of the overlay links. This means that, in principle, any two peers can create an overlay link between them. Based on whether using a central component or not, unstructured overlay networks can be further subdivided into hybrid decentralized system, e. g. BitTorrent [27], and purely decentralized system, e. g. Gnutella [65].

The following sections describe two well-known unstructured overlay protocols, which can be considered as the representatives of hybrid decentralized P2P system and purely decentralized P2P system.

2.1.1 BitTorrent

BitTorrent [27] is one of the most popular P2P file-sharing applications, it was released by Bram Cohen in 2001. Lately, its traffic has been estimated to occupy approximately 27-55% of all Internet traffic [108].

A BitTorrent system consists of a tracker, an initial seeder, many seeders and many downloaders. The peers which can provide the complete file are called seeders; and the first seeder is the initial seeder. The downloaders are normal users downloading the file. The tracker is a central server, which stores the information of the current seeders and downloaders of the files.

To share a file, the initial seeder splits its file into some small blocks, varying from 64KB to 4MB. Then it creates a meta-data file called *torrent-file*, which comprises of the Universal Resource Locator (URL) of the tracker and the checksum of each block, which is calculated with the Secure Hash Algorithm (SHA) [84]. After registering itself to the tracker, the initial seeder publishes its *torrent-file*, e. g. on some web page.

To download the file, the downloader usually gets the torrent file from the web server. With the torrent file, the downloader asks the tracker for the list of the seeders and other downloaders of this file. Now he can download different blocks from different peers/downloaders and upload his finished blocks to other downloaders. At the different downloading phases, various downloading strategies, e. g. random-block-first and rarest-block-first, are applied.

Obviously, the tracker of the BitTorrent system is the central component. Both the publisher and the downloaders need the tracker to exchange information. If it fails or the connection to it fails, the whole BitTorrent system breaks down. To prevent such single failure, DHT was adopted in the later versions of BitTorrent clients, e. g. Azureus(Vuze) [4].

2.1.2 Gnutella

Gnutella [65] was a full decentralized file-sharing P2P application. Unlike BitTorrent, Gnutella has no central component even in its earliest version.

In order to join the Gnutella network, the user must find an existing node in this network, e. g. via web caches, Internet Relay Chat (IRC), etc. After attaching to the network, the user exchanges different messages with his neighbors:

- `Ping/Pong` messages that are used to know other nodes and the information of the files on these nodes.
- `Query/Response` messages that are used to search certain files in the network.
- `Get/Push` messages that are used for the file transfer.

In the early versions of Gnutella (pre 0.4), pure flooding search was applied and this search strategy made the Gnutella network unscalable. In the later versions, a Time To Live (TTL) field was included in the `Query` messages. Since version 0.6, Gnutella has introduced two different peer modes: leaf peer and ultra peer. The leaf peers are connected with 3 ultra peers and perform inquiry only via the ultra peers. The ultra peers have a high out-degree and are responsible for routing the queries. With this two-tier structure, the efficiency and scalability are improved significantly.

Gnutella is not only a favorite P2P application for Internet users, but also a hot topic for academic researchers. Many suggestions have been proposed to further improve the performance. For example, Lv et al. [81] suggested a random walk for the query messages and a uniform random graph for the network

topology; Cohen et al. [28] proposed an optimal replication scheme for the random search, in which the number of the replications should be the square-root of the query rate; Castro et al. [19] further suggested a structured overlay for Gnutella to reduce the maintenance overhead.

2.2 Structured Overlay Networks and Key Based Routing

In contrast to unstructured overlay networks, structured overlay networks have special rules to generate the overlay links. The virtual structure in such a network can be a ring, a torus or a more complicated graph.

Structured overlay networks have the following common properties:

- Each overlay node has a unique Identifier (ID), i. e. address, in the overlay address space. The IDs of the nodes are generally assumed to be uniformly distributed in the address space.
- Each point (also called *key*) in the address space is consistently mapped to its "closest" overlay node according to a certain distance metric.
- Each node has a small amount of connections to other nodes following the constructive rules of the virtual structure.
- Messages are delivered to a key instead of a node's ID.
- Messages are routed towards the destination key by minimizing the distance to it.
- The corresponding node of a key, i. e. the closest node to the key, is guaranteed to be reached within a limited number of steps.

As just mentioned, the destination of a message in a structured overlay is a certain *key* in the address space, hence the routing scheme in a structured overlay network is called *Key Based Routing (KBR)*, which is different to the *end-to-end routing* scheme, where a node ID must be the destination.

Due to the enormous network size of overlay networks, the classic proactive routing protocols are not feasible any more. For example, a link state routing algorithm (e. g. Dijkstra algorithm [45]) and a distance vector routing algorithm (e. g. Bellman-Ford algorithm [9]) need to know the global topology. For a large overlay network with millions of nodes, the topology information which has to be stored and be propagated would overload the network. Unlike the proactive routing protocols, reactive routing protocols do not need to keep a large state of the network topology, but they usually flood the whole network for route discovery. Thus they are obviously unsuitable for a large network, too.

Instead of the conventional proactive or reactive routing protocols, structured overlay networks apply hybrid routing algorithms. Each overlay node saves limited overlay topology information cooperatively. When the destination cannot be directly reached, the message is delivered via some other known overlay node towards the destination. After several iterations, the message is guaranteed to reach the correct destination node.

It is worth noting that the KBR feature of the structured overlays is the base of many P2P applications in the Internet. In addition, KBR is also applied in the network layer, as will be illustrated in Chapter 3.2. In the following sections, some selected structured overlay network protocols will be described.

2.2.1 Chord

Chord [114] is one of the four earliest structured overlay protocols proposed in 2001, along with CAN [94], Tapestry [131] and Pastry [104].

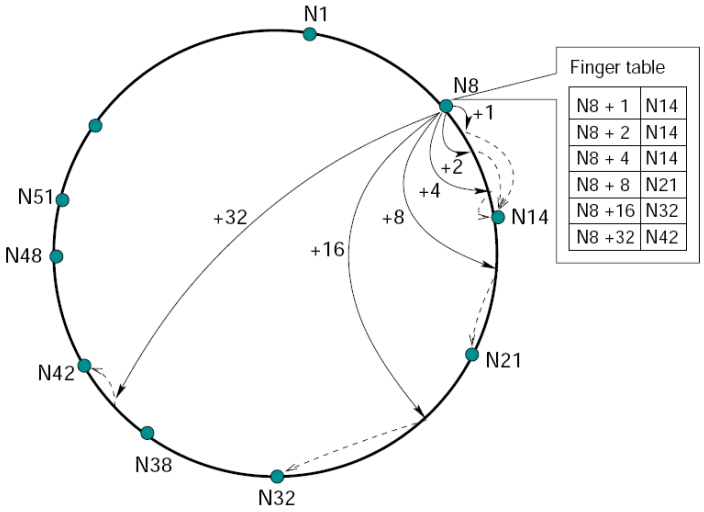


Figure 2.1: Chord ring and its finger table (from [115], Figure 4a)

The address space of Chord is a circle of numbers from 0 to $2^m - 1$, where m is the bit length of the node ID. The node IDs are unique, random and uniformly distributed in the address space. Usually the IDs are generated with the SHA-1 hash function [84]. The distance between two IDs is the modular difference between them, i. e. $d(a, b) = (b - a) \mod 2^m$.

The overlay structure is a ring with some chords, that are named *fingers* in [114] (see Figure 2.1.) To maintain the ring structure, each node keeps the connections to two other nodes, whose IDs are the closest to its own ID in the clockwise and counterclockwise directions respectively. They are called the *successor* and *predecessor* of this node. Besides the connections to the successor and predecessor, each node maintains at most $\log_2 N$ fingers, where N is the number of the nodes in the network. The other end points of the fingers are exponentially distant from the node.

When a node forwards a message towards its destination, i. e. a certain key in the address space, it chooses the *finger*, whose endpoint ID is the closest to the key. This is exemplarily shown in Figure 2.1: If node 8 want to send a message to key 33, it will use the 4th finger in its finger table. Since the end-pointer of this finger is the closest one to the target. As the distance to the key will be approximately halved by each forwarding step, the whole routing process will terminate after $O(\log_2 N)$ hops.

2.2.2 Content Addressable Network

The Content Addressable Network (CAN) uses a d-dimensional torus as its virtual overlay structure. The node ID in CAN is a d-dimensional Cartesian coordinate. The address space is so partitioned that each part of it is associated to exactly one node in this part. To maintain the overlay structure, each node has connections to its neighbors in the torus. The routing algorithm in CAN is similar to that in the Greedy Perimeter Stateless Routing (GPSR) protocol [70], i. e. greedily forwarding the message to the neighbor who is the closest one to the destination coordinate.

When a new node joins the CAN network, it first generates a random d-dimensional ID. Similar to Chord, it has to know one existing CAN node to bootstrap, i. e. to send a `join` message to it. The bootstrapping node then routes the `join` message to the destination node, who is the closest one to the newcomer's ID. Now the address space of the corresponding node is split, and one part is associated to the newcomer.

When the CAN overlay has N nodes and uses a d-dimensional address space, the complexity of the routing length is $O(d \cdot N^{\frac{1}{d}})$. The neighbor count of each node is $O(d)$. Obviously, if we set d equal to $\log N$, the complexity of the routing state and that of the routing length of CAN will be the same as those of Chord, i. e. $O(logN)$.

2.2.3 Overlays with de Bruijn Graph

Some structured overlay networks, such as Koorde [64] and Viceroy [82], use de Bruijn graph [107] as the virtual structures. A de Bruijn graph consists of m^k nodes, where the node ID is composed of m symbols and the size of the symbol set is k. Assume $m = 3$ and $k = 2$, the node IDs look like 101, 011, etc. Each node in the de Bruijn graph has directed connections to its m neighbors, whose IDs can be expressed by shifting all symbols in the node ID by one place to the left and adding a new symbol x at the end, for $x = 0, ..., k - 1$. For example, let $m = 3$ and $k = 2$ again, node 101 will have the neighbors of nodes $01x$, i. e. node 010 and node 011.

The routing process in de Bruijn graph can be described as follows. Suppose that the destination ID is d. Each node chooses the ID of the next hop in such a way that it shifts left its own ID by 1 alphabet and fills the most right place with one new alphabet of the destination ID. For example, the route from 101 to 001 will be: $101 \rightarrow 010 \rightarrow 100 \rightarrow 001$. Hence the routing complexity on de Bruijn graph is $O(m)$ and the complexity of the routing state is $O(k)$. Since k is usually very small, de Bruijn overlays are considered to have a constant node degree of $O(1)$ and a routing complexity of $O(\log N)$, where N is the number of the overlay nodes.

2.2.4 Kademlia

Kademlia [83] is the most widely applied structured overlay protocol in many well-know P2P applications, such as eMule/aMule [3, 2], BitTorrent [27] and Azureus(Vuze) [4].

The node ID in Kademlia is a random 160-bit[1] number. The Exclusive OR (XOR) operation is used as distance metric, i. e. $d(a, b) = a \oplus b$. The routing table of a Kademlia node consists of 160 buckets. Each of them contains up to k (such as 20) nodes, whose IDs have the same beginning i bits as the ID of this node, for $i \in (0, 159)$. Just for clarification, we assume a Kademlia implementation with 4-bit node ID and let $k = 2$. Then node 0110 will have four buckets, and they will be filled with nodes having IDs of 1xxx, 0xxx, 01xx and 011x. Each bucket contains up to 2 nodes.

The lookup process of Kademlia is to recursively query α (such as 3) of the k closest nodes to the destination key. In each recursion step, a Kademlia node knows more nodes closer to the target. The lookup terminates, when no closer node can be learned. The routing complexity of Kademlia is $O(\log N)$.

The XOR metric used in Kademlia has some useful properties:

- It is commutative, i. e. the distance from node A to node B is the same as that from node B to node A. When node A receives a message from node B, it adds node B into its routing table in the same bucket as node B has done with node A.
- The routing table can be extended by using prefix instead of bit. If the prefix has the length of b, the routing table size is then $\frac{160}{b} \cdot 2^{b-1} \cdot k$, and the routing complexity is correspondingly reduced to $O(\log_{2^b} N)$.

2.3 Distributed Hash Tables (DHTs)

A DHT is an implementation of a hash table in a distributed system. It is usually used for data storage and retrieval. Often a structured overlay that supports Key Based Routing can often be found as the substrate behind DHT. The data stored in a DHT is always associated with a certain key and the keys are mapped to the nodes consistently.

The most important property of the DHT is its hash consistency [66, 67]. It means that when nodes join or leave the system, the map between the keys and nodes will be just slightly changed.

Usually, a DHT implementation has the following common messages:

- `Put` message is used to store `data` under a certain `key` in the distributed system. The substrate, i. e. the structured overlay network, routes the message to the destination node, which will store the `key-data` pair in its cache for a certain time period.
- `Get` message is used to retrieve the `data` associated with the certain `key` from the distributed system. The routing process is the same as that of the `Put` message and the destination node will return the cached data to the inquirer.
- `Get-Reply` message returns the `data` specified by the `key` in the `Get` message. This message can be routed back to the inquirer via the structured overlay network with the KBR scheme, e. g. for censorship-resistance reasons, or it can also be directly sent back to the inquirer on the transport

[1] This length varies in different implementations, e. g. the KAD network uses 128-bit addresses.

layer (via TCP/UDP) for performance reasons. In the latter case, the `Get` message must contain the transport layer address of the inquirer.

These three messages correspond to the common DHT APIs: `put(key, data)` and `get(key)`. In a concrete DHT implementation, there could be some other kinds of messages like `Remove` and `Replace`.

2.3.1 OpenDHT

OpenDHT [100] is an application based on the Bamboo DHT [99] on the PlanetLab testbed [90]. The Bamboo DHT applies the Pastry [104] algorithm and has included some optimizations for node churn. OpenDHT extends the two common DHT APIs mentioned above by introducing authentication. Table 2.1 shows the extended DHT APIs of OpenDHT.

Procedure	Functionality
put(k,v,H(s), t)	Write (k,v) for t; (k,v) can be removed with secret s
get(k) returns (v,H(s), t)	Read all v stored under k Returned value(s) unauthenticated
remove(k,H(v), s, t)	Remove (k,v) put with secret s; if t>TTL remaining for put
put-immut(k,v, t)	Write (k,v) for t; immutable (k = H(v))
get-immut(k) returns (v, t)	Read v stored under k Returned value immutable
put-auth(k,v,n, t,K_P,σ)	Write (k,v), expires at t; public key K_P; private key K_S can be removed using nonce n; $\sigma = H(k, v, n, t)_{K_S}$
get-auth(k,H(K_P)) returns (v,n, t,σ)	Read v stored under (k,H(K_P)) Returned value authenticated
remove-auth(k,H(v),n, t,K_P,σ)	Remove (k,v) with nonce n; parameters as for put-auth

Table 2.1: The put/get interfaces of OpenDHT (from [100], Table 1)

OpenDHT provides a public DHT service via Sun RPC (over TCP) [118] as well as via XML RPC (over HTTP) [124]. The storage on OpenDHT has the limitation of one kilobyte per data item. Each item can persist for maximal one week if not refreshed. Since its publication in 2005, it has attracted a lot of academic interests and quite a few applications have been built based on OpenDHT so far. Unfortunately, due to the lack of maintenance, OpenDHT was shut down by its author on July 1, 2009.

2.3.2 KAD DHT

KAD is another DHT implementation. It is based on the Kademlia [83] algorithm and mainly used for file-sharing. Since the KAD network is supported by eMule/aMule and MLDonkey clients, which have millions of concurrent users, it is the most widely deployed DHT implementation nowadays.

Both the node ID and the key in the KAD network have 128 bits. There are two types of keys: *Source key* and *Keyword key*. The *Source key* is the identifier of a file, which is hashed from the file content; whereas the *keyword key* is computed by the words of the file name. The *Source key* and the *Keyword key* are published on their ten "closest" (in term of XOR distance) nodes every 5 and 24 hours, respectively [112].

Stutzbach et al. [116] gave out some detailed analysis of the lookup performance of the KAD DHT. They pointed out that the expected hop count of a KAD lookup is $1 + \frac{\log_2 n - 7.73}{6.98}$. In a large network with 1 million nodes, the expected hop count is merely 2.7. This efficient lookup is achieved by increasing the routing table size, i. e. adding more buckets as well as adding more contacts in the bucket.

2.4 KBR Applications other than DHT

Although many structured P2P overlay networks have been proposed for DHT applications, there are quite a few non-DHT applications based on the KBR feature of structured overlays. These applications aim at providing diverse and more complicated services:

- Application Layer Multicast (ALM) applications, e. g. Scribe [22] and Multicast-CAN [96], use structured overlays (Pastry [104] and CAN [94] respectively) to realize scalable multicast in the application layer.
- Internet Indirection Infrastructure (i3) [113] lets the overlay nodes redirect the Internet traffic to the end-host by decoupling the packet delivery from the packet receipt. In i3, each packet has an identity and is delivered to a certain overlay node, from which the receivers get the packet.
- Storage utility, such as PAST [46, 105], aims at large-scale and Internet-based Decentralized Object Location and Routing (DOLR).
- The distributed video recorder [33] schedules the recording tasks to different overlay nodes.
- Peer-to-Peer file systems, e. g. OceanStore [73], SiRiUS [62] and IgorFS [75], store file (blocks) fully decentralized on the overlay nodes and provide grained cryptographic Access Control List (ACL) support.

So, not only DHT applications but also many other applications exploit the KBR characteristics of structured overlay and make the P2P application more versatile.

2.5 Optimizations in Overlay Networks

Most of the overlay protocols build their virtual structures on top of the Internet without considering it otherwise than as a black box, which provides connection between two overlay nodes. Although the routing complexity of the overlay protocols are claimed to be small, e. g. $O(\log N)$, Kutzner et al. [74] found that the overlay links are so heterogeneous that the Round-Trip Times (RTTs) of the overlay links in the Overnet network follows a power-law distribution over many

orders of magnitude. The mean value of the RTTs in their experiment was more than 5 seconds. They concluded that the performance of a plain overlay network before optimizing the overlay links could be very bad. Hence, it is necessary to adjust the overlay structure in such a way that the small routing complexity remains but the routing delay reduces.

2.5.1 PRS, PNS and PIS

Gummadi et al. [54] pointed out that the end-to-end latencies in structured overlay networks can be significantly reduced by applying some proximity methods, i. e. Proximal Route Selection (PRS), Proximal Neighbor Selection (PNS) and Proximal Identifier Selection (PIS). In the following context, proximal nodes mean the nodes that can be reached with small cost, e. g. small RTT.

- PRS refers to a class of algorithms, where the routing process chooses the most proximal neighbor in the direction towards the destination as the next hop, instead of choosing the neighbor whose ID is the closest to the destination key in the virtual structure. It is obvious that the proximity method works only if there are more than one neighbors which lie in the direction towards the destination key. For example, in early stages, Chord can choose the most proximal finger in the direction instead of the one, who is closest to the target, to route the message; while Kademlia can query the most proximal α nodes from the k nodes in the bucket, that are closest to the target.

- PNS implies that when the overlay nodes choose their virtual neighbors to build the overlay, they prefer the proximal nodes. As the example in Figure 2.1 shows, the routing table (i. e. finger table) of node *N8* is built strictly following the Chord protocol: The routing table contains the nodes that have IDs directly following $(x + 2^i)$, where x is the node ID and $i = 1, 2, ..m$. Under this strict condition, PNS is impossible, since there is only one candidate for each routing table entry. However, if we extend the Chord protocol by allowing the nodes, whose ID belongs to $(x + 2^i, x + 2^{i+1})$, into the routing table, PNS can be put in execution. According to this extension, the routing length of $O(\log N)$ remains.

- PIS: Ratnasamy et al. [95] suggested that the nodes can choose their overlay IDs according to their localities. In their proposal, there are some well deployed landmarks. Each node gets a coordinate by measuring its distance to the landmarks and joins the Content Addressable Network (CAN) overlay with the coordinate. According to the addressing scheme, the proximal nodes have close overlay IDs, this reduces the maintenance overhead and thus improves network performance. The drawback of PIS lies in the potentially uneven ID distribution, which makes load balancing hard [54].

Gummadi et al. further showed in their experiments that the performance improvement by PNS is more significant than that by PRS. While PRS can be implemented straightforwardly by comparing the distance to each candidate in the routing table, it is difficult to implement PNS. We note that each overlay node knows only a small number of other overlay nodes in its routing table.

However, the task of PNS is to get to know more proximal nodes via the known nodes with small overhead. Several methods have been proposed to solve this problem. We will discuss some of them in the following sections.

2.5.2 Internet Distance Prediction

As mentioned above, the overlay network performance can be significantly improved by adding proximal nodes into the routing table. Obviously, it is costly to probe all candidate nodes iteratively. Hence, Internet distance prediction approaches have been proposed to predict the distance between two nodes in an efficient way to alleviate PNS. Since in most of the works, the RTT is used to define the distance between two nodes, we refer delay prediction to the distance prediction in the following context.

Landmark-based Delay Prediction

Some systems, e. g. IDmaps [48], Global Network Positioning (GNP) [85] and King [55], utilize landmarks to predict the host-to-host delay. In these systems, some reliable hosts in the Internet are chosen as landmarks and the distance between every two landmarks is measured. Based on the landmark-to-landmark delay, the host-to-host delay can be estimated as follows:

- IDmaps [48] deploys some special hosts as landmarks (tracers in [48]), and stores the delay matrix of the landmarks on several servers. The latency of two hosts is estimated as the sum of the latencies of the hosts to their closest landmarks plus the latency between the two landmarks.

- GNP [85] deploys some landmarks in the Internet and stores the distances between every two landmarks in a latency matrix. Based on this matrix, the landmarks are embedded into a geometric space, e. g. 3-dimensional Euclidean space. A host can now download the coordinate set of the landmarks, measure its distance to the landmarks and then calculate its own coordinate. The latency between two hosts can be easily estimated according to their coordinates.

- King [55] is similar to IDmaps, but uses DNS servers as landmarks and approximates the latency between two hosts according to the latency between their nearby DNS servers. This approach saves the deployment of the infrastructure of landmarks and has small estimation error. More important, as it uses DNS server, this method scales well.

Vivaldi - Infrastructure-less Prediction

Landmark-based delay prediction could introduce large errors, since it approximates the host distance with the landmark distance. Moreover, it requires some well deployed landmarks, which is infeasible for many P2P applications.

Vivaldi [34] is an infrastructure-less Internet distance prediction approach, which assigns synthetic coordinates in an n dimensional Euclidean space to the overlay nodes. As with GNP it estimates the distance between two overlay nodes according to their coordinates. The fundamental idea of Vivaldi is to minimize the whole system prediction error iteratively. At first, each node is

assigned with a random coordinate. During the operation, the node collects the coordinates from its overlay neighbors as well as the distances (RTTs) to them. It updates its coordinate regularly to reduce the system prediction error according to

$$x_i = x_i + \delta \times (rtt - \parallel x_i - x_j \parallel) \times u(x_i - x_j)$$

where x_i is the node's own coordinate, x_j is the coordinate of neighbor j, δ is a constant of step size and u is the unit vector.

After some iterations, the coordinates of all nodes converge and the sum of the prediction errors is minimized.

Meridian - Coordinate-less Prediction

The distance prediction based on virtual coordinates, such as Vivaldi, is often inaccurate, as it maps the high-dimensional Internet infrastructure into a low-dimensional Euclidean space. Meridian [126] circumvents this problem by eliminating virtual coordinates.

Meridian groups the known nodes into several concentric rings, whose diameters increase exponentially. Each ring stores at most k nodes ($k = 16$ in [126]). Each node periodically randomly selects a list of nodes, one node per ring. After that, it sends a gossip message to the each node on the list. The gossip message contains again some randomly chosen nodes, one from each ring. Upon receiving such a message, the receiving node measures the distance to the nodes in the gossip message, and puts them into its concentric rings.

After some iteration, the Meridian node would fill each concentric ring fully, if there are enough nodes. In the innermost ring, the closest nodes are stored. Meridian can also be considered as a kind of special loosen structured overlay network, where overlay links are generated according to the physical distance, i. e. the closer a node, the more probably a link to it. Meridian can not only find the closest nodes but also cope with some special problems: e. g. closest node (to some target) discovery, central leader election, etc.

Triangle Inequality Violation

Vivaldi and Meridian are two representative solutions of the Internet delay prediction. Note that their efficiency and accuracy are based on the assumption of triangle inequality for Internet delays, i. e. for node a, b and c:

$$d(a, b) + d(b, c) \geq d(a, c)$$

However many studies [133, 130, 78] found that this assumption is not always true. Wang et al. [123] demonstrated that the Triangle Inequality Violation (TIV) of Internet delay has severe impact on the accuracy of theses prediction methods. Since naively excluding the TIV links won't help to improve the performance of Vivaldi and Meridian, several optimizations were proposed to mitigate the TIV problem, such as adding non-Euclidean adjustment to Euclidean coordinate [78] and TIV alert mechanism [123].

2.5.3 Internet Service Provider and P2P

The overwhelming popularity of P2P application has also attracted much interest of the Internet Service Providers (ISPs), since they not only have to route

the enormous P2P traffic, but also have to deal with the large cross-ISP cost.

Aggarwal et al. [7] suggested that ISPs can provide an "oracle" service to its P2P users, so that the peers can choose "good" neighbors in the same or proximal Autonomous System (AS), thus to reduce the cross-ISP traffic. Bindal et al. [10] attempted to enable biased neighbor selection for BitTorrent. This can be done either by importing Internet/AS topology maps to the BitTorrent tracker and clients, or by using P2P traffic shaping devices to propagate the information of the internal peers. Choffnes et al. [25] proposed an approach to reduce the P2P cross-ISP traffic without cooperation from ISPs. In their proposals, the peers utilize the Content Distribution Network (CDN) redirection as the locality hints to choose overlay neighbors in the same AS.

2.6 KBR Service for Multiple Applications

As mentioned in Section 2.4, structured overlay networks can be used to build versatile P2P applications. Although the purposes of some applications are very similar, e. g. the Kademlia-based DHTs are used in both eMule and BitTorrent, it is usual to build one structured overlay for each application. This is however a very inefficient solution due to expensive maintenance and bootstrapping overhead of each application.

Nowadays, it is expected that a host will run multiple P2P applications simultaneously. Since in current solutions the structured overlays are isolated from each other, the maintenance overhead of the host would be enormous. Consider the KAD DHT used by aMule, whose routing table size is [116]:

$$\log_2 n \times 2^{b-1} \times k$$

Let $b = 4$, $k = 10$ and $n = 1M$, then this results in a routing table with 1594 entries! (Note that this result is only an upper bound, since not all the buckets are fully filled.)

Figure 2.2 is a snapshot of the routing table status of an aMule client for the first 30 minutes. From this figure, we observe that the routing table size of the KAD network of 1.6 million nodes is beyond 650. It is very costly to keep such a large routing table up-to-date. Although a longer maintenance interval can reduce the maintenance overhead, it will cause many stale routing entries, which reduce the routing performance.

It is worth noting that some structured overlay networks apply proximity methods to improve the network performance, e. g. using Vivaldi and Meridian. This will further increase the local state and the maintenance overhead. (As illustrated in [126], in a 2000 node Meridian system, the node has to maintain state for hundreds of nodes.)

Based on these observations, several approaches were proposed with the aim at how to provide KBR services for multiple applications in a more efficient way. Depending on the authors, the group of application instances is referred to as "namespace" [100], "subgroup" [68] or "cluster" [8].

2.6.1 ReDiR

Rhea et al. [100] proposed the Recursive Distributed Rendezvous scheme (ReDiR), which uses OpenDHT to construct a per-namespace KBR system.

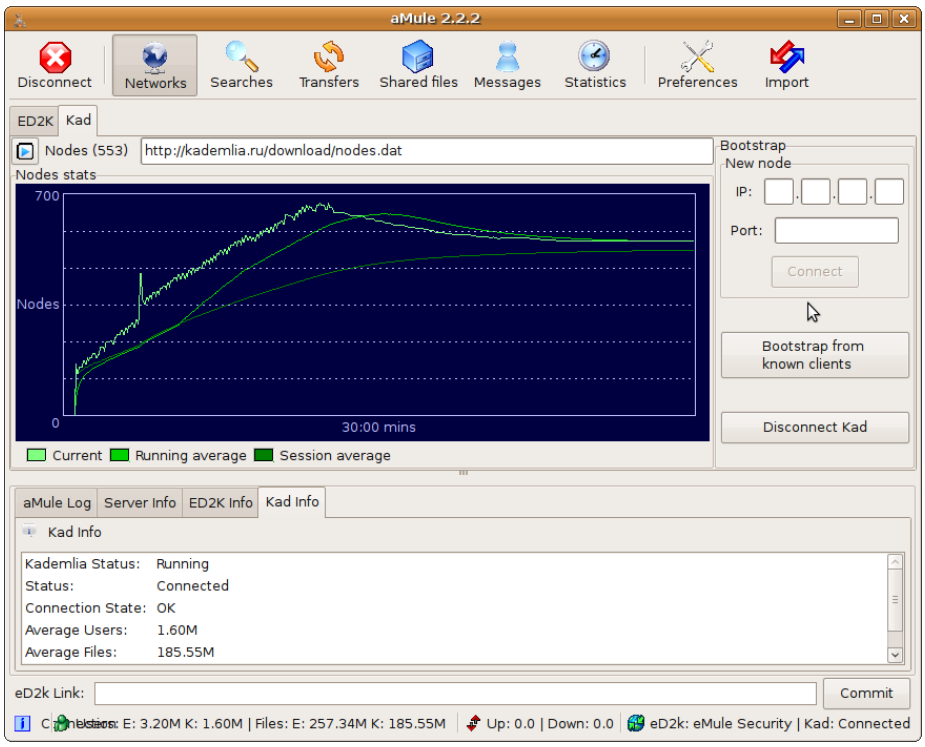


Figure 2.2: Screenshot of aMule on 01.Jan.2010

ReDiR divides the application address space into *partitions* at different *levels*. Each partition at level i covers the b sub-partitions at level $i+1$, where b is called branching factor which is often set to 2.

For the example shown in Figure 2.3, the address space is partitioned into 4 levels, and level i contains 2^i partitions. All these partitions at different levels uniformly divide the whole address space and all the partitions are mapped to OpenDHT under the key that is hashed from the tuple of `("application-name", level, position)`. For instance, the 4th partition at level 3 of application "i3" is mapped to OpenDHT under the key: `hash(‘‘i3’’,3,4)`.

When a new node joins the application KBR system, it begins its registration at a certain initial level, say l. At first, it calculates which partition at level l includes its ID. Then it registers itself under that partition. It registers itself further to the partition at the coarser levels (go up) until it is not the extremal node in that partition. It also registers itself to the partition at the finer levels (go down) until it is the only node in that partition.

Consider the example in Figure 2.3 again, node B starts at level 2. It will further register itself at level 1 because it is still the extremal node in the partition at level 2. It doesn't register at level 0, because it is not the extremal node at level 1. Node B stops the registration at level 3, because it is the only node in the partition at that level. Such registration process in OpenDHT will be repeated regularly, and the lowest level at which the registration previously completed will be used as the next initial level.

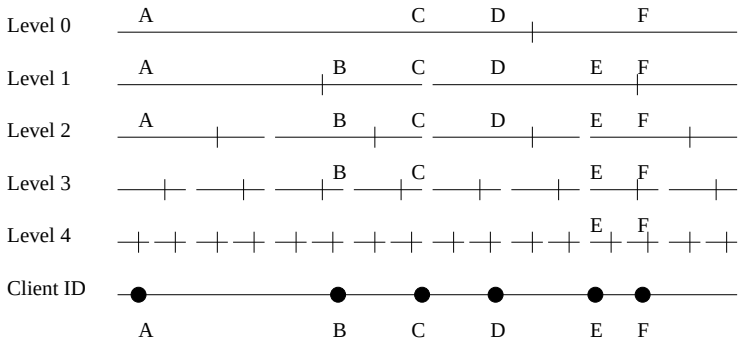


Figure 2.3: Example of a ReDiR tree with branching factor b=2.

When ReDiR routes a lookup message to a certain key k in a given namespace, it first computes which partition at level l_{lookup} includes the key k, and gets a list of the registered nodes in this partition from OpenDHT. The sender should either get the nodes list at the coarser level (go up) if there is no node succeeds the key, or get the nodes list at the finer level (go down) if the key is between the node IDs in the retrieved nodes list. If the key proceeds all the node IDs in one partition or the partition contains only one node, the node whose ID succeeds the key is the destination node.

The selection of l_{lookup} is an important factor of the lookup efficiency. A precise estimation of the level of destination node can stop the lookup process in 1∼2 steps. [100] suggested that the levels, at which the last 16 lookups completed, can be used as hints for the next lookup. The simulations in [100] showed that the lookup overhead of ReDiR is approximately 1.3 DHT get operations.

The maintenance overhead of ReDiR is approximately 3 `put` messages and 4 `get` messages per maintenance interval. Suppose there are 2^L ReDiR nodes, where L is an arbitrary number less than the ID's length. Suppose again the IDs of the ReDiR nodes are strictly uniformly distributed, we can derive that the partitions at the deepest levels (level L) contains one ReDiR node and all the other partitions contain two ReDiR nodes (the two extremal nodes in each partition). In total, there are 3×2^L nodes registered on the partitions, i. e. each ReDiR node registers itself 3 times on average. Note that before a node decides whether to register itself or not, a `get` operation is needed to poll the destined partition. Therefore, each ReDiR node performs 4 `get` operations on average, because three of the `get` operations would have positive answers, that triger the registrations; and one of the `get` operations would have a negative answer, that terminates the registration process.

2.6.2 Diminished Chord

Karger et al. [68] proposed *Diminished Chord* for the formation of subgroups in a Chord overlay. Each subgroup corresponds to a binary search tree that is embedded in the Chord ring. Each tree is rooted at the hash value of the subgroup name. When a node joins a certain subgroup, it registers itself to all its ancestors in the binary tree. Every inner node stores only the smallest subgroup member ID, which succeeds its ID. Each node can know its closest subgroup

member by querying the ancestor nodes towards the root of the subgroup tree. Usually a match can be found on some inner nodes before the root. Hence the lookup complexity is $O(\log N)$, where N the whole network size (not the subgroup size). Since the inquiry is upwards to the root, the subgroup member can register itself only at the highest level in the binary tree, where the inner node should store its information. This reduces the space usage from $O(\log N)$ to $O(1)$ per subgroup member, while the lookup delay increases slightly.

To route a message with a certain key within a subgroup, the message is first routed to the successor of the key according to the regular Chord rule. After that, the successor of the key finds the successor of the key within the subgroup with aforementioned algorithm (i. e. traversing the binary search tree to find the destination node). Both steps take $O(\log N)$ overlay hops.

Comparing to ReDiR, there are several drawbacks in Diminished Chord:

- Larger lookup delay. The first step of lookup is based on the regular KBR in Chord, which needs $O(\log N)$ overlay hops. With proximity methods, the real delay of the lookup can be significantly reduced [54]. The second lookup step is the traversal of the binary tree. Since the tree structure is fixed and it is difficult to apply proximity methods, this step causes a large delay.

- Uneven node load. In the embedding algorithm, the inner tree node and its right child are always mapped to the same overlay node [68]. This means some overlay nodes have $O(\log N)$ more load than the other nodes per subgroup.

- Dependency of Chord. This approach is only proposed for the Chord protocol. Additionally, it requires the Chord protocol to equip additional *pre-fingers*. Otherwise, the lookup complexity will increase and the implementation will be more complicated. Moreover, unlike ReDiR, the subgroup member must join the prime Chord ring.

2.7 Security Issues

Since each peer has only to contain very limited knowledge of the whole system, P2P networks achieve remarkable scalability. However, the limited view of the whole system also makes P2P networks vulnerable to a few security threats, e. g. Sybil attacks, Eclipse attacks and routing and storage attacks.

- Sybil attack means that a malicious node generates enormous amounts of bogus nodes into the P2P network and tries to take control of the whole system.

- Eclipse attack means that an attacker tries to isolate some honest nodes by filling their routing tables with references to the malicious nodes.

- Routing and storage attack means that an attacker puts corrupted routing information and data into the system.

For more details about the security issues in P2P networks, refer to [119], which gives out a comprehensive survey of these security threats and the corresponding solutions.

2.8 Summary

In this chapter, I have briefly presented peer-to-peer overlay networks. After a short overview of several unstructured overlay networks, I have introduced some popular structured overlay protocols, that offer Key Based Routing (KBR) as their fundamental service. Based on KBR, there are various applications, including the widely applied DHT. In addition, I have discussed some optimization approaches that aim at improving the overlay performance by coupling the overlay topology to the underlying Internet infrastructure. After that I have also illustrated some approaches that focus on providing the KBR service to multiple applications efficiently and at the end some security issues of P2P overlay networks have been briefly mentioned.

Chapter 3

Routing and KBR in Wireless Ad-hoc Networks

Routing protocols have evolved all the time since the appearance of computer networks. In the meantime, multiple taxonomies have been proposed for routing protocols. In the beginning, routing protocols were mostly designed for infrastructure-based networks. The protocols were usually classified into interior and exterior protocols according to the location where these protocols are applied. For example, the Routing Information Protocol (RIP) and the Open Shortest Path First (OSPF) are two famous interior routing protocols used within a single Autonomous System (AS); while the Border Gateway Protocol (BGP) is the most widely used exterior routing protocol among different ASs in the Internet.

More recently, due to the wide deployment of wireless ad-hoc networks, new routing protocols have been developed. Routing protocols can then be categorized into infrastructure-based routing protocols and ad-hoc routing protocols. As P2P applications become more and more popular, the routing concept is further extended to the application layer. Thus, routing protocols can be further divided into overlay routing protocols and network layer routing protocols. Note that in overlay routing protocols, the destination of a message is a *key* instead of a concrete node ID. Hence, these overlay routing protocols are called Key Based Routing (KBR) protocols, while the network layer routing protocols provide end-to-end routing.

While the KBR protocols are widely studied in the Internet, some researches [129, 39, 50, 97, 47] showed that KBR can also be implemented in wireless ad-hoc networks. In this chapter, I briefly overview several classic wireless ad-hoc routing protocols and some resent works that aim at providing KBR services in wireless ad-hoc networks.

3.1 Routing Protocols for Wireless Ad-hoc Networks

Due to their limited capacity and low link reliability, the classic infrastructure-based routing protocols are not suitable in wireless ad-hoc networks. Big routing

table size and large maintenance overhead would overload the wireless ad-hoc network. Therefore, quite a few wireless ad-hoc routing protocols have been proposed, which aim at small routing state and mild maintenance traffic. In addition, some of these protocols, e. g. the Extremely Opportunistic Routing (ExOR) protocol, utilize the special characteristics of the wireless medium to improve the network performance.

3.1.1 OLSR

Optimized Link State Routing (OLSR) [26] is a proactive link state routing protocol optimized for wireless ad-hoc networks. As a link state routing protocol, OLSR nodes use `hello` messages to discover their neighboring nodes. Each node broadcasts Topology Control (TC) messages to disseminate its link state to all the other nodes. With the thereby obtained global topology information, each node can calculate the next hop to any destination with Dijkstra's algorithm [45].

The major optimization effort of OLSR lies in the use of Multipoint Relays (MPRs) nodes to route the payload messages and broadcast the TC messages. The MPR nodes are so selected that any 2-hop neighbors of a node can be reached via an MPR node. MPR node selection is an NP problem [122]. A simple but good heuristics solutions for this problem was proposed by Amir Qayyum in [93]:

1. Among the neighbor nodes of node x, the one, which is not yet selected as MPR node and has the highest node degree, is chosen as the MPR node.
2. The neighbors of the newly chosen MPR node are excluded from the 2-hop neighbor-set of node x.
3. The first two steps will be repeated until the 2-hop neighbor-set of node x is empty.

OLSR reduces the routing table size as well as the maintenance traffic significantly, since only the MPR nodes are considered as the active nodes for both routing and broadcasting, which avoids many duplications of the TC messages. However, if the network size is very large, the TC messages, that flood the whole network, can still overload the network.

3.1.2 AODV and DSR

Since proactive routing protocols generate too much topology maintenance traffic, reactive (on-demand) routing protocols were introduced for wireless ad-hoc networks. The Ad-hoc On-demand Distance Vector (AODV) [88] protocol and the Dynamic Source Routing (DSR) [63] protocol are two often studied representatives of this class.

Both AODV and DSR are stateless routing protocols, i. e. node state is generated only when there is a routing demand. Upon routing demand, the sender floods the whole network to discover a path to the destination. In AODV, the discovered routing information is stored along the path, i. e. each intermediate node stores the next hop to the destination. In DSR, the discovered path is stored at the sender in form of a source route and the intermediate nodes don't store the path information.

In both protocols, a discovered path has to be refreshed after a certain time interval. If the path breaks, DSR tries to salvage the route locally, while AODV immediately initiates a new Route Request (RREQ) message to discover a new route.

Reactive routing protocols have no regular topology maintenance overhead and discover the path only on demand. Although the on-demand property lets these routing protocols scale well in wireless ad-hoc networks under a small traffic load, these protocols will generate enormous amounts of route discovery messages, which will overload the network, when the application traffic load is medium or large.

3.1.3 Geographic Routing

In some special wireless ad-hoc networks, the node's geographic location information is available, e. g. via Global Positioning System (GPS) or some indoor location system [86]. For these networks, geographic routing (also called georouting) protocols are usually applied.

The georouting protocols greedily forward the packet to the neighbor who is geographically closest to the destination. (The destination is usually given as a 2-dimensional coordinate.) Different strategies have been proposed to circumvent the local minimum problem, that occurs when greedy forwarding comes into a dead end. E. g. the GPSR [70] protocol suggests the right hand rule: in case that no neighbor is closer to the packet destination than itself, the node chooses its first neighbor in the couterclockwise direction to route the packet and excludes it from routing consideration when the packet turns back.

If the nodes move very quickly, the respective forwarder might not have the accurate position information of its neighbors. Thus, it is beneficial to determine the next hop node on the fly. Implicit Geographic Forwarding (IGF) [13] is a stateless geographic routing protocol. It modifies the 802.11 DCF MAC protocol [59] such that the RTS/CTS handshaking frames contain the information about the destination. The closer a node is to the destination, the earlier it responds with a CTS frame. The sender sends then the DATA frame to the node that first responds.

As stated above, the destination of a packet in georouting protocols is a certain node's coordinate. To solve the coordinate of the destination node, various location services have been proposed. [80, 58] are two grid-based distributed location solutions that periodically update the node's location information to certain parts of the network. The Last Encounter Routing (LER) [53] protocol even merges the explicit locating process into the routing process, in which each LER node maintains the time and the location of the last encounter with every other nodes in the network.

3.1.4 Opportunistic Routing

In error-prone wireless ad-hoc networks, the channel quality changes frequently and the selected next hop might be temporarily unavailable. Thus it makes sense to select the next hop from a set of possible forwarding candidates.

Biswas et al. [12] proposed the Extremely Opportunistic Routing (ExOR) protocol. Here, the sender assembles its known neighbors into a candidate list. The nodes in the list are prioritized according to the distance to the destination.

The candidate with the highest priority broadcasts the packets in its buffer first. After that, other candidates broadcast the not yet transmitted packets sequentially according to their priorities. To avoid loops, packets are uniquely identified and transmitted only upon their first arrival.

Chachulski et al. [23] proposed the MAC-independent Opportunistic Routing and Encoding (MORE) protocol. In MORE, the node randomly mixes packets before broadcasting them. This ensures that routers hearing the same transmission do not forward the same packet. Thus the packets received from different nodes introduce less superfluous traffic as before. Their experimental results demonstrated a significant performance gain.

Sanchez et al. [106] suggested a sender-based algorithm for the next hop selection. The nodes that have received a data packet acknowledge their receipts using a delay function according to their closeness to the destination. The sender sends a selection packet on the first ACK, thereby suppressing data duplication.

3.2 KBR in Wireless Ad-hoc Networks

While more and more P2P applications have been successfully applied in the Internet, applying P2P techniques in wireless ad-hoc networks has also attracted many research interests. Some efforts tried to deploy structured overlay directly on top of a conventional ad-hoc routing protocol; others used cross-layer or integrated approaches for the overlay construction. Since the former solutions have scalability problems due to the enormous overlay maintenance overhead in the underlay [31, 18], only the latter approaches are discussed in the following sections.

3.2.1 Cross-layer Solutions

Cross-layer solutions aim at the interaction between different layers, e. g. the overlay protocols utilize some features of the routing protocols to improve the performance.

CrossROAD

CrossROAD [39] is a cross-layer solution for a KBR system in wireless ad-hoc networks. It builds the Pastry [104] overlay on top of OLSR [26]. In order to reduce overlay maintenance overhead, a CrossROAD node lets the Link State Update (LSU) packets of the routing protocol piggyback the information of its overlay instance. The node, that receives an LSU packet, stores that information in the routing table and shares the information to the overlay. Since these LSU packets flood the whole network regularly, after a while each node will have a complete global view of the overlay network.

The idea of CrossROAD is applicable to any Link State (LS) routing protocol as well as to any overlay protocol. However, as stated in Section 3.1.1, proactive routing protocols have a poor scalability in wireless ad-hoc networks due to their maintenance cost. In wireless environments, links can break and nodes can go down at any time. Moreover, overlay instances can join and leave spontaneously. All that makes a consistent global overview of the network impossible.

MADPastry

MADPastry [129] is another cross-layer solution for structured overlays in wireless ad-hoc networks. It implements Pastry on top of AODV. MADPastry applies Random Landmarking [125] to group the proximal nodes into clusters with the same prefixes in their overlay IDs.

When forwarding a Pastry message to a certain Pastry node, the intermediate node, which runs MADPastry, can intercept the Pastry message. If its Pastry ID is closer to the destination key than the current destination ID, it takes the message and routes it further. In the case that there is no AODV route to the next hop of the Pastry routing, and if the forwarder is already in the destination cluster, the message will be flooded inside the cluster; otherwise, an original AODV route discovery procedure will be triggered.

MADPastry considers physical locality by assigning close Pastry IDs to proximal nodes. This reduces the maintenance traffic of the Pastry overlay. However, with this Proximal Identifier Selection (PIS), the distribution of the overlay IDs could be uneven, which would make the load distribution skewed. Moreover, the AODV based routing discovery procedure can generate enormous amounts of RREQ messages in the network, which in turn hurts the network scalability.

3.2.2 Integrated Solutions

The integrated solutions aim at merging different layers into one single layer. Hence, the maintenance overhead of multiple layers can be avoided.

Scalable Source Routing

Scalable Source Routing (SSR) [50] is a DHT-inspired routing protocol at the network layer. It combines source routing with Chord-like routing [114]. All nodes are assumed to have a unique and uniformly distributed address. Unlike the proactive routing protocols, SSR nodes do not maintain route information for all the destinations in the network. Unlike the reactive routing protocols, SSR nodes maintain some state, thus they do not need to discover the route by flooding. Therefore, SSR can be considered a hybrid routing protocol.

In SSR, a node only needs to maintain source routes to its physical neighbors and to its *predecessor/successor* (i. e. virtual neighbors) in the address space. Optionally, the node may also cache the source routes to the recently contacted nodes. These optional routes are stored according to a Least Recently Used (LRU) policy. Fuhrmann et al. [50] recommended a small routing cache containing only 255 entries and demonstrated that this suffices even for very large networks of 100 000 nodes.

The routing algorithm of SSR contains Chord's routing rule as well as the PRS principle. The selection of the next intermediate node from the local routing cache obeys the following three rules:

1. The address of the next intermediate node must be closer to the destination address than the previous intermediate node.
2. The most proximal nodes satisfying the first rule are preferred.
3. The one, who satisfies the first two rules and whose address is the closest to the destination address, is selected as the next intermediate node.

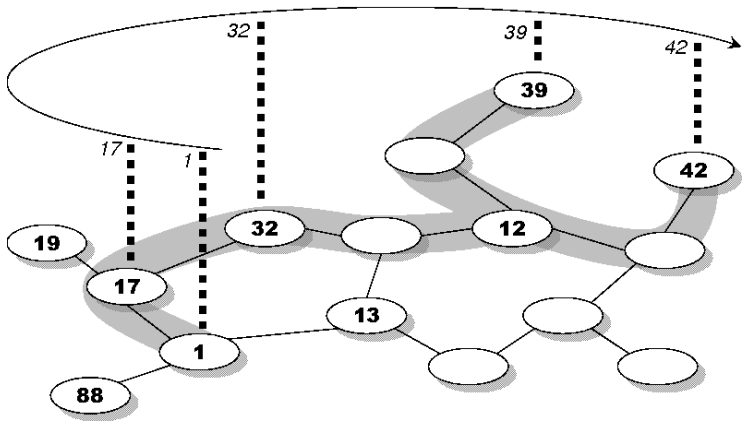


Figure 3.1: Illustration of SSR's routing process (from [51], Figure 1).

Note that the next intermediate node could be several hops away. That's why we use "the next intermediate node" instead of "the next hop" here.

Figure 3.1 illustrates the SSR routing with a simple example taken from [51]. In this example the routing caches of the nodes contain only the source routes to the nodes' physical and virtual neighbors. We assume that node 1 wants to send a packet to node 42. Node 88 is first excluded for the routing process (according to rule 1). The packet is forwarded to node 17 because that node is the physically closest node to node 1 (according to rule 2). Node 17 is preferred over node 13 since node 17 is virtually closer to node 42 than node 13 (according to rule 3). For the same reasons, node 17 forwards the packet to node 32. Node 32 forwards the packet to its successor, i. e. node 39, which in turn forwards the packet to its successor that coincides with the destination, i. e. node 42.

Since each SSR node maintains the source route to its successor, the routing process is guaranteed to proceed (towards the destination) at each step. The routing process will end at the destination node or the node, whose address is the closest to the destination. That means SSR supports both the end-to-end routing scheme and the Key Based Routing scheme.

It is worth noting that there are several SSR-related works: [76] and [29] discuss the bootstrapping algorithms of SSR, [77] proposes a cryptographic certificate scheme for securing SSR and [11] extends SSR to utilize unidirectional links.

Virtual Ring Routing

Virtual Ring Routing (VRR) [15] is another DHT-inspired network layer routing protocol that supports KBR. Like SSR, each node in VRR has a random unique ID. VRR forms a Chord-like virtual ring in the network by setting up paths from the VRR node to its r virtual neighbors in the virtual ring (r is set to 4 in [15]). Unlike SSR, the path information is distributed on the intermediate nodes as AODV does: each node stores only the next hop to the destination node. Suppose the network diameter is $O(\sqrt{N})$, each path to the virtual neighbor will generate $O(\sqrt{N})$ state cross the network. Thus, the per-node state of VRR is $(r\sqrt{N})$, where N is the node number in the network.

When routing a packet to a certain destination, the sender and each intermediate node use the path, whose end-point address is the closest to the destination, to forward the packet. In order to repair local link failure quickly, VRR nodes store not only the next hop but also the next-next hop of each path.

Although the average routing state of VRR is $O(\sqrt{N})$, the distribution of state per node in the network could be very uneven. Consider a network with two parts, which are connected via a single node. Since half of the virtual neighbors are probabilistically located in the other part of the network, the paths to these virtual neighbors have to cross this single node and the state on this node would be $O(r \cdot N)$. This large state could cause problem in some wireless ad-hoc networks with small capacity, e. g. in Wireless Sensor Networks (WSNs).

Geographic Hash Tables

Geographic Hash Tables (GHTs) [97] are designed as a Data-Centric Storage (DCS) approach for sensor networks, where each node knows its geographic coordinate. A GHT provides a similar service as a DHT does, i. e. `put(key, value)` and `get(key)`. In particular, when a node routes a `put` or a `get` message to a certain `key`, it first maps the `key` into a 2-dimensional coordinate inside the network area. After that it uses the GPSR [70] protocol to route the message to the node, whose coordinate is the closest to the destination.

Different to SSR and VRR, GHT has two application obstacles: First, the GHT node must know its geographic coordinate. Although this can be achieved by deploying GPS on each node or by using some other localization method, to do so is very costly and unrealistic in many ad-hoc networks. Second, the mapping function in a GHT system must map the `key`s uniformly inside the network area. This could be complicated, when the network area is not in a regular form. Moreover, when the network area changes, e. g. by adding more sensor nodes to expand the network area, the mapping function must be updated. Hence some additional protocols have to be introduced to update the mapping function.

3.3 Summary

In this chapter, I have given a rough overview of some classic end-to-end routing protocols in wireless ad-hoc networks. After that, I have presented some recent works that aim at providing a KBR service in wireless ad-hoc networks. Some of them focus on an integrated solution by pushing the structured overlay down into the network layer, e. g. SSR and VRR, others aim at the interaction between overlay and underlay, e. g. CrossROAD and MADPastry. Since the integrated solutions generate only small routing state and little maintenance overhead for a single layer, they are used as an important component of the Lite-Ring system that I will propose in the latter part of this thesis.

Part II

The Overlay Network Lite-Ring

Chapter 4

Overview

The Lite-Ring overlay protocol aims at providing an efficient Key Based Routing (KBR) service for multiple applications. The name of "Lite-Ring" means light-weight ring-like overlay. This part of my dissertation is based on my earlier work: "Providing KBR Service for Multiple Applications"[44].

Similar to conventional structured P2P overlay networks in the Internet, Lite-Ring runs on top of the transport layer (see Figure 4.1). Lite-Ring can also be applied on top of the network layer in wireless ad-hoc networks. This will be described in Part III of this dissertation.

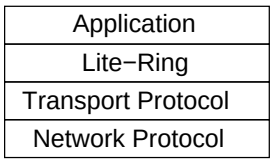


Figure 4.1: Lite-Ring stack in the Internet

4.1 Goal and Tasks

As a structured overlay protocol, Lite-Ring offers the same KBR service as many other conventional structured P2P protocols do, e. g. Chord and Pastry. Given a message with a certain key as the destination, the Lite-Ring overlay will route the message to the node whose ID is the closest to this key.

As stated in Chapter 2.3 and Chapter 2.4, many P2P applications can benefit from a Key Based Routing (KBR) service. The de-facto standard solution is to build a full-fledged overlay for each application independently. In case that a large number of applications need the KBR service, this solution will cause a lot of node state and maintenance overhead. One might think that the $O(\log N)$ state and maintenance overhead caused by a structured overlay typically are not very costly. However, as described in Chapter 2.6, the real implementation of a full-fledged structured overlay can generate a much larger state and overhead than the theoretical results indicate, e. g. the aMule client can generate more than 650 entries in its routing table to maintain its DHT

network (see Figure 2.2). Moreover, the bootstrapping process of the individual overlay can introduce additional overhead, e. g. the node has to find an existing overlay node for every P2P application that it runs.

Similar to the de-facto standard solution, Lite-Ring will also build an individual overlay for each application. However, by leveraging a public DHT service, the Lite-Ring overlay provides the same KBR service as the full-fledged overlay with very small node state and maintenance overhead. In addition, with the help of the public DHT service, the bootstrapping problem is solved with no extra cost. Based on these observations, Lite-Ring is able to efficiently provide generic KBR services to a large set of applications.

4.2 A Naive Solution

Before we go further to the design insights of the Lite-Ring protocol, let's examine one naive solution for the problem of providing KBR services to multiple applications, i. e. to build a single shared structured overlay for all applications, which is often suggested as an alternative to overcome the drawbacks of the de-facto standard solution.

Although a single global overlay keeps both the node state and the maintenance overhead to a limited size. This solution is impractical because the nodes and the applications might be too heterogeneous to build one public overlay. For example, a bandwidth restricted node, e. g. cell phone, would run a small bandwidth KBR application, and not be willing to be bothered with high volume traffic, e. g. from file-sharing programs. Moreover, it is not unrealistic to imagine that some applications don't want their traffic to be seen by other foreign nodes for security reasons. Even worse, some unstable nodes of a single application could degrade the performance of the whole overlay, that is however undesirable.

Besides the problems mentioned above, this solution encounters other technical problems. For example, a node N with application A_i could join with the concatenated ID $(A_i : N)$, as illustrated in Figure 4.2. This means that the different application instances populate different regions in the KBR address space and this makes load balancing difficult.

Similarly, if a node N with application A_i joins the shared overlay with the concatenated IDs $(N : A_i)$, as illustrated in Figure 4.3, each node has to store

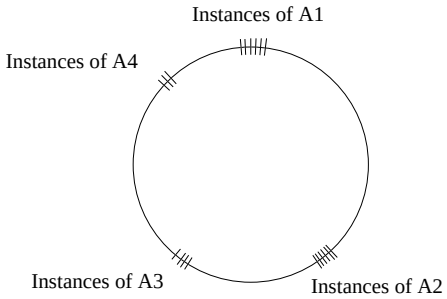


Figure 4.2: Overlay shared by different applications with the concatenated IDs $(A_i : N)$

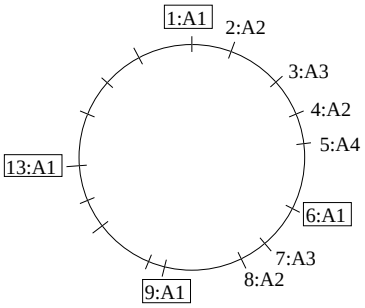


Figure 4.3: Overlay shared by different applications with the concatenated IDs $(N : A_i)$

and maintain additional state for each application that it does *not* run, so that it can forward the message to the node that *does* run that application and whose ID is the closest to the destination.

For the example shown in Figure 4.3, assume that the shared overlay uses a Chord-like structure without additional state. According to the KBR semantic in Chord, the message targeted to $(6 : A_2)$ will be routed to $(6 : A_1)$. But the node with ID 6 doesn't run the application A_2, i. e. the delivery is wrong in the application's view. To solve this problem, node 6 has to store additional state for other applications.

Based on the observations above, we conclude that the naive solution of a shared overlay is not feasible to provide KBR services for multiple applications.

4.3 Design Insights

The design of Lite-Ring is based on two observations.

1. In a structured overlay such as Chord [114], each overlay node stores $O(\log N)$ fingers, where N is the total number of nodes. Thus, there are totally $N \cdot O(\log N)$ fingers stored in the entire system.

   Assume that the overlay address space is I and the transport layer address space is T. Each finger is a tuple that maps an overlay address I_x to a transport endpoint T_x in the underlying network, e. g. an IP address and a port number. With the help of these fingers, the structured overlay is a distributed implementation of the mapping $I \to T$.

   In fact, this implementation is quite redundant: In an overlay with N uniquely identified nodes there exist actually only N different fingers altogether. This redundancy is costly, not only in terms of space but more importantly in terms of maintenance overhead.

   Although some overlays [82, 64] use de Bruijn(-like) graphs[107], which have a constant node degree, practical considerations suggest to have $O(\log N)$ fingers per node to improve the overlay stability [64].

2. Assume an idealized Chord ring with address space $[0, 1[$ and assume further that the nodes join sequentially at addresses 0, $\frac{1}{2}$, $\frac{1}{4}$, $\frac{3}{4}$, $\frac{1}{8}$, $\frac{3}{8}$, $\ldots$ Then, the address space is always uniformly divided into 2^n parts by the

initial 2^n nodes, for $n = 0, 1, 2, 3...$ Given the own address and the predecessor's address, a node can (almost exactly) calculate the node (I_1) that is responsible for any given key I_k in the address space I. Here, "almost exactly" means that with small probability, the calculation may erroneously yield the responsible node's direct virtual neighbor (I_2). Summarizing this insight we can say that if the nodes join the KBR overlay in such a predictable pattern, we have a mapping $I_k \rightarrow I_{1,2}$, where either I_1 or I_2 is responsible for I_k. A public DHT service can then map $I_{1,2} \rightarrow T_{1,2}$. So far, we get the mapping $I_k \rightarrow T_{1,2}$.

From these two observations we can construct efficient KBR overlays[1] for multiple applications. Assume the nodes of application A are assigned their application specific addresses in the predictable pattern as described above. Then we know that there is an application specific mapping $(I_k, A) \rightarrow (I_{1,2}, A)$ where either I_1 or I_2 are responsible for I_k with respect to application A. The public DHT service can then map $(I_{1,2}, A) \rightarrow T_{1,2}$ and we get the application specific mapping $(I_k, A) \rightarrow T_{1,2}$. Note that in this way there is no duplicated finger any more, which highlights the efficiency of Lite-Ring.

[1] By KBR overlay we mean a structured overlay that supports a KBR service.

Chapter 5

Design

This chapter describes the design of the Lite-Ring protocol. It details the address assignment scheme, the overlay structure, the join procedure, the routing process and the maintenance work. At the end of the chapter, the complexity of Lite-Ring will be compared with ReDiR [100] and Diminished Chord [68], which also aim at the efficient KBR service for multiple applications.

5.1 Address Assignment Scheme

Lite-Ring is based on a virtual ring structure. Its address space is similar to that of the Chord protocol [114], i. e. a circle of numbers from 0 to $2^m - 1$, where m is the bit length of the address. Unlike many other KBR overlays where node IDs[1] are assigned randomly, Lite-Ring addresses are assigned sequentially according to the following pattern:

$$\begin{cases} addr_0 &= 0 \\ addr_i &= 2^{m-k} \cdot (i - 2^k + \frac{1}{2}) \quad \text{for} \quad i \geq 1 \end{cases} \tag{5.1}$$

where i is the sequence number of the joining node, $k = \lceil \log_2 i \rceil$ denotes the division depth of the address space.

In the example of Figure 5.1, the Lite-Ring address space can accommodate up to 16 nodes ($m = 4$). As shown, 10 addresses have already been assigned according to allocation sequence {0, 8, 4, 12, 2, 6, 10, 14, 1, 3, 5, 7, 9, 11, 13, 15}. This is actually a pre-order walk through a binary tree (see Figure 5.2).

This addressing scheme distributes the application addresses uniformly in the address space. If there are exactly 2^k application instances, this distribution is strictly uniform. The address pattern is also used by Karger et al. for load balancing in [69], where only one of the $\log N$ random virtual servers is activated to approach this ideal uniform address distribution.

Lemma 1 *Suppose that there are N nodes in the application overlay ring and their addresses are assigned according to the form given in Equation 5.1. Then a node can calculate the addresses of at least $\frac{N}{2}$ other nodes in that overlay, given its own address and the address of its predecessor.*

[1] We use node ID and node address for the same meaning in this thesis.

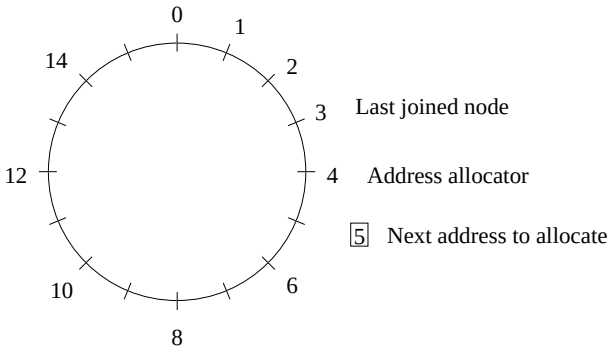


Figure 5.1: Application overlay ring with 10 application instances

Let's illustrate the situation by an example, before giving the proof. In Figure 5.1, an application overlay ring contains 10 nodes and has an address space size of 4 bits. According to the join sequence, the node addresses are in the order of {0, 8, 4, 12, 2, 6, 10, 14, 1, 3}. Node 8 knows its predecessor (node 6), so that it knows which addresses have already been assigned, i. e. {0, 8, 4, 12, 2, 6}. But it cannot tell if the addresses {10, 14, 1, 3, 5} have been assigned. Nevertheless, all of these possible nodes are successors of the known existing nodes.

Proof: Without loss of generality, assume N is an even number. Because the address allocation scheme is a pre-order walk through a full binary tree, the last allocated $\frac{N}{2}$ addresses are the leaf nodes, while the other $\frac{N}{2}$ addresses are the inner nodes. These $\frac{N}{2}$ leaf nodes have at least the knowledge of all the inner nodes.

Because of the ring structure, the last allocated $\frac{N}{2}$ addresses (leaf nodes) are uniformly inserted into the previously allocated $\frac{N}{2}$ addresses (inner nodes). Hence each inner node must know at least one leaf node, i. e. its predecessor, from which it also knows of the existence of all the inner nodes.

So both the leaf nodes and the inner nodes have the knowledge of the existence of $\frac{N}{2}$ nodes of the network.

□

The assumption of sequential joins is difficult to realize in practice, but it gives a good insight how we can exploit such an addressing scheme. In Chapter 5.3, some improved address allocation schemes are introduced. They enable concurrent joins and still preserve the described address prediction property.

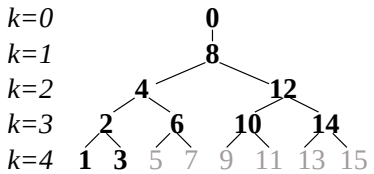


Figure 5.2: Binary address tree structure

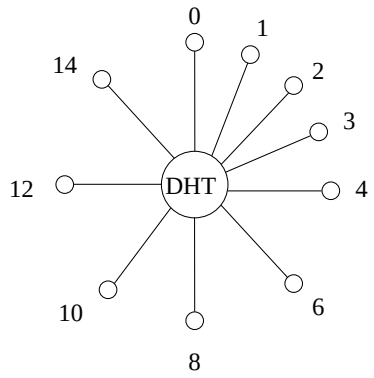


Figure 5.3: Star structure with a DHT as the central node

5.2 Overlay Structures

Lite-Ring comprises three virtual structures: a ring (cf. Figure 5.1), a binary tree (cf. Figure 5.2), and a star (cf. Figure 5.3). Each node belongs to all the three structures.

- The ring structure is the base of the consistent hashing property, i. e. the uniform and consistent mapping from key to nodes. Moreover, it enables the nodes also to roughly know the overlay size (cf. Lemma 1). In order to maintain the ring structure, each node stores its successor and predecessor.
- The tree structure is used to distribute and aggregate information that allows us to balance the address distribution. To maintain the tree structure, each node stores its two child nodes and its parent node.
- The star structure contains the connections of the nodes to the DHT, which is used for the mapping between the overlay addresses and the transport addresses, i. e. the mapping of $(I_x, A) \rightarrow T_x$. This star structure also ensures the connectivity of the Lite-Ring overlay.

5.3 Join Mechanisms and Address Allocation

Lite-Ring's key idea is a well-defined address assignment scheme. It allows the nodes in the overlay to calculate locally, which node is responsible for a given key. Depending on the different usage scenarios, we can choose from several different join mechanisms, which are briefly discussed in the following.

5.3.1 Unique Allocator

If the nodes join the Lite-Ring overlay slowly enough, we can assume an almost sequential order. In this case, we say that the node that has non-equal distances to its successor and predecessor has the role of an *address allocator*. (As an exception, node 0 becomes *address allocator* when the distances to its successor and predecessor are equal.) After a node has joined the overlay, the address allocator role moves to the joining node's successor in the ring.

In the example of Figure 5.1, node 3 has most recently joined the overlay. Thus, its successor, node 4, becomes the address allocator. It will assign the address 5 to the node that joins the overlay next.

When a node leaves the overlay, it may happen that for a short moment, multiple address allocators co-exist. For example, when node 1 in Figure 5.1 leaves, both node 0 and node 4 assume the role of address allocators. Both allocators will assign different addresses. After node 0 fills the address hole by assigning the address 1 to a newly joining node, only node 4 remains as the unique address allocator.

There are two ways for a joining node to find the address allocator:

1. The address allocator registers itself in the DHT with the key "APP-ALLOCATOR", and the newly joining node queries the DHT for the transport address of the address allocator.

2. The joining nodes register themselves in the DHT under the key of "APP-NEWNODE", and the address allocator regularly polls the DHT.

Note that both keys are application-specific, i. e. each application overlay registers either of the two keys in the DHT.

This method can assign the newly joining nodes with addresses that strictly follow the form given in Equation 5.1. Unfortunately, these methods are only practical, when the nodes join slowly, almost sequentially, and remain in the overlay for some time. When many nodes join simultaneously or when the node churn rate is high, this address allocation mechanism does not work: either the DHT node responsible for the key "APP-ALLOCATOR" is overloaded with the concurrent `get(`$k_{APP-ALLOCATOR}$`)` operations from the newly joining nodes; or the DHT node responsible for the key "APP-NEWNODE" is overloaded with the `put(`$k_{APP-NEWNODE}, T$`)` operations. Luckily, the strictly sequential address assignment scheme (cf. Equation 5.1) is not necessary, as will be described now.

5.3.2 Multiple Allocators

As stated in Lemma 1, the sequential address assignment allows the nodes to calculate the correct overlay addresses of at least $N/2$ other nodes. A slight deviation from the strictly sequential order does not render Lemma 1 invalid, provided the address distance between neighboring nodes (in the ring) does not differ by more than a factor of two. If the difference exceeds a factor of two, the address tree will become imbalanced. As a result, some nodes erroneously calculate a larger or smaller total number of nodes.

Consider, for example, again Figure 5.2. If a node at the lowest level of the address tree, e. g. node 3, or a node whose successor is at the lowest level, e. g. node 2, assigns an address to a joining node, the tree becomes imbalanced.

This imbalance will be analyzed in detail in Chapter 5.5. Moreover, as will be shown there, it can be cured by a subsequent maintenance step. Hence, a non-sequential address allocation sequence does not break our mechanism in principle, so that any node can serve as *address allocator* and simply assign a new node the address that lies half way between itself and its successor.

The join procedure is thus reduced to finding an arbitrary node in the Lite-Ring overlay and requesting an address from it. If the DHT supports aggregation, both the address allocators and the joining nodes could register themselves in the DHT (see "unique allocator" method). But unlike that in the non-aggregating DHT, the registrations do not propagate to the node responsible for the "APP-ALLOCATOR" or "APP-NEWNODE" key, if the DHT can create a response locally. Moreover, if the DHT applies proximity route selection, the request traffic is likely to remain in the vicinity of the joining node with respect to the underlying network topology.

If the DHT does not support aggregation, such a registration would cause the same overload that we just wanted to avoid. Thus we need to find an address allocator at random. An according method is introduced in the following.

5.3.3 Address Picking and Probing

One straightforward way to find an address allocator is that the joining nodes draw a random address and poll the DHT for an address allocator (i. e. an existing node) closest to this address.

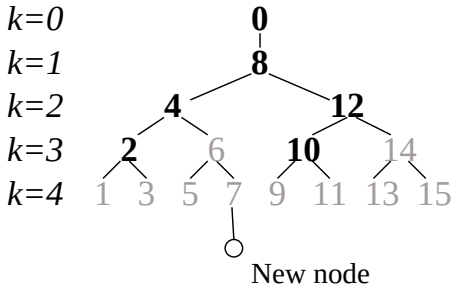


Figure 5.4: Join mechanism with address picking and probing

Figure 5.4 gives an illustrative example: A newly joining node randomly picks an address, say 7. The node then queries the DHT for this address and its potential predecessors at the different levels of the address tree, i. e. 7 (0111), 6 (0110), 4 (0100) and 0 (0000). Node 4 is the first existing node in that list. That means, the DHT can provide a transport address for that node. The joining node then contacts node 4 which assigns it the address 6.

If we applied this probing approach in strict order, it would have a worst case time and message overhead of L, where L is the address length in bit. But since the addresses of the potential predecessors of the random address are in a strict sequential order, we can apply a binary search. Thus, we have a worst case time and message overhead of $\log L$ only. For example, in an 128-bit address space a joining node needs only maximal 7 DHT queries to find an address allocator.

5.3.4 Centralized Assignment

Before we move on and discuss Lite-Ring's routing, I would like to mention yet another address assignment method. It is based on the observation that many P2P overlays use a centralized bootstrapping mechanism, e. g. a server with a peer list. Assuming that we have such a server in place, it can directly assign the

joining node's address. However, to be able to properly assign the addresses, it needs a (nearly) exact knowledge of all the peers for the respective application. In a scenario where peers may leave the system ungracefully, such knowledge cannot be easily maintained. Clearly, the central server could be a single point of failure of the entire system. Hence, we do not consider this option further.

5.3.5 Summary

The different join mechanisms are briefly compared in table 5.1, where

- M is the number of DHT nodes.
- N is the node number of a Lite-Ring overlay.

The lookup complexity of the DHT is supposed to be $\log M$, as in many conventional DHTs, e. g. Chord and Pastry.

From the discussion above, the "multiple allocators" approach is considered the best choice in an aggregating DHT. However, such aggregating service is not popular in the common DHT implementations. Thus, since the "picking and probing" approach supports concurrent joins, too, it was chosen as basis for the Lite-Ring implementation.

Unlike the universal ring for bootstrapping proposed in [21], Lite-Ring requires only a primitive DHT without any additional functionality such as multicast. More importantly, it is not necessary for the Lite-Ring instance to join the DHT overlay.

	Unique allocator	Multiple allocators	Picking and Probing
Join Overhead	$O(\log M)$	up to $O(\log M)$	$O(\log\log N \cdot \log M)$
Load Ratio	$O(N)$	up to $O(\frac{N}{M})$	$O(1)$
Remark	No support for concurrent joins	Needs aggregation in DHT	Preferred

Table 5.1: Comparison of different join mechanisms

5.4 Routing in Lite-Ring

If each application overlay contains exactly 2^k nodes and if the address tree is balanced, each node can perfectly calculate the responsible node address for any key. In practice neither of these two idealistic assumptions holds.

The resulting problems can be demonstrated with the following examples: Consider node 4 in Figure 5.2. Its predecessor in the ring (node 3) lies in level $k = 4$. Based on its locally available information the node would thus naively assume that the overlay contains 2^4 nodes. Similarly, node 12, which has its predecessor (node 10) in level $k = 3$ assumes that there are 2^3 nodes. Altogether, node 4 overestimates the node count whereas node 12 underestimates it.

Suppose first that both node 4 and node 12 want to send a message to a key that is close to 7, say $6\frac{2}{3}$. Node 4 erroneously assumes that node 7 exists and shall thus handle the message. Accordingly, it inquires the DHT for the transport address of node 7. Only after receiving a negative reply (or worse after a timeout), node 4 learns that there is no node at position 7, and that

hence node 6 shall handle the message. In contrast to node 4, node 12 correctly assumes that node 6 should handle that key in the first place.

Now suppose that both node 4 and node 12 want to send a message to a key that is close to 3, say 3.1415. Now, node 4 correctly assumes the existence of the responsible node 3, whereas node 12 erroneously sends the message to node 4. Node 4 must then forward the message to node 3.

From these examples we see that if a node overestimates the node count, it might need one additional DHT query under some circumstances. Conversely, if a node underestimates the node count, the message might need one additional overlay hop.

Lemma 2 states this effect more precisely:

Lemma 2 *Suppose there are N nodes in the application overlay. Suppose the overlay is balanced and each node knows only its successor. When a node calculates the node address that is responsible for a random key k (from a uniform distribution), in up to 26% of the cases, the calculation is wrong because either the node does not exist or it is not the correct one. On average 17% of the calculations yield the wrong node.*

Proof: Suppose a of the N nodes lie in the fully populated part of the tree, and the other n nodes lie in the lowest level of the tree. We know there are at most a positions in the deepest level of this binary tree, thus $n \in [0, a]$ and there are $m = a - n$ vacant positions in the lowest level. $2n$ nodes have knowledge about the lowest level, namely n nodes in this level and the other n nodes that have their successors in the lowest level. These $2n$ nodes will overestimate the overlay size. Thus they might erroneously send messages to the m vacant positions in the lowest level of the tree. The other $a + n - 2n = m$ nodes underestimate the overlay size. Thus they incorrectly route the messages that are destined to the n nodes in the lowest level.

Noting that the $2n + m$ is the whole node count and $2a$ describes all the possible positions, the error probability is hence

$$P = \frac{2n}{2n+m} \cdot \frac{m}{2a} + \frac{m}{2n+m} \cdot \frac{n}{2a} = \frac{3nm}{(2n+m) \cdot 2a}$$

(using $m = a - n$ and $n = b \cdot a$ where $b \in [0, 1]$)

$$P = \frac{3n(a-n)}{(n+a) \cdot 2a} = \frac{3b - 3b^2}{2 + 2b} \in [0, \frac{3}{2}(3 - 2\sqrt{2})] \approx [0, 0.26]$$

The resulting function is plotted in Figure 5.5.

The average probability of a wrong calculation is then:

$$\int_0^1 \frac{3b - 3b^2}{2 + 2b}\, db = \frac{9}{4} - 3 \ln 2 \approx 0.17 \tag{5.2}$$

□

Lemma 2 describes a worst case assumption. In practice, the nodes might have more knowledge about the population of the overlay. For example, nodes could share their knowledge piggybacked with other protocol messages. The additional benefit is however limited, as shown with the following lemma:

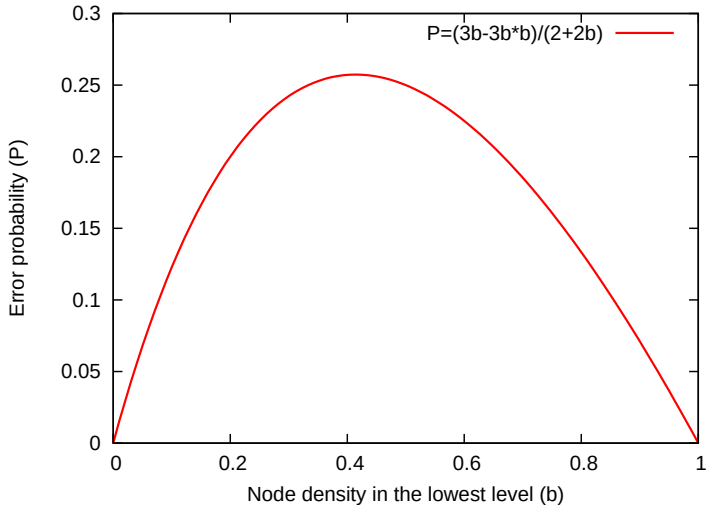


Figure 5.5: Probability of a wrong calculation with local knowledge

Lemma 3 *Suppose there are N nodes in the application overlay. Suppose the address tree is balanced and each node knows the precise node count. Then, up to 25% of the destination calculations are wrong. On average, the calculation is wrong in 12.5% of the cases.*

Proof: Even though a node knows the exact number of nodes in the overlay, it does not know the vacant positions in the lowest layer. It can thus either assume that a given position is populated (potential overvaluation) or not (potential undervaluation). Using the same parameters as above we find the following error probabilities: When the node applies the undervaluation strategy, an error occurs for the messages that are destined to the n nodes in the lowest level, i. e. for n out of $2a$ possible destinations: $P_{under} = \frac{n}{2a} = \frac{b}{2}$. With the overvaluation strategy the error occurs for the $m = a - n$ vacant positions in the lowest level, i. e. $P_{over} = \frac{m}{2a} = \frac{1-b}{2}$.

Both cases are plotted in Figure 5.6. Obviously there is a threshold at a population of 50% in the lowest level. If the number of nodes is beyond that threshold, the overvaluation strategy is better, otherwise the node should apply the undervaluation strategy. From the figure we see that the maximal probability of a wrong calculation is 25%, and the average probability is 12.5%. □

In practice the preferred strategy depends on the cost metrics. Still we can apply the following rule of thumb: A DHT query has a complexity of $O(\log N)$ hops in the DHT overlay. Typically, an additional DHT query will thus be more expensive than one additional hop in the application overlay. Hence, in that case, nodes should prefer the undervaluation strategy.

Alternatively, assuming that the application nodes are rather stable, each node could register a synthetic node that occupies the position between itself and its successor. (Here, a synthetic node is just an additional mapping in the DHT.) In the example of Figure 5.7 the black nodes are the actual overlay

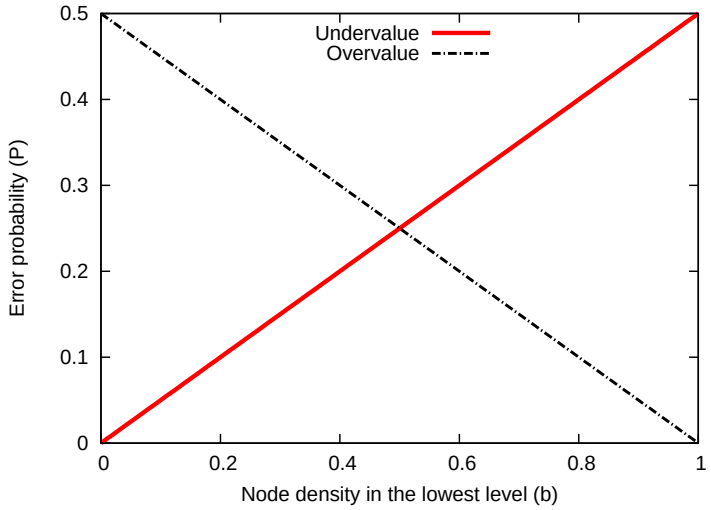


Figure 5.6: Probability of a wrong calculation with precise node count information

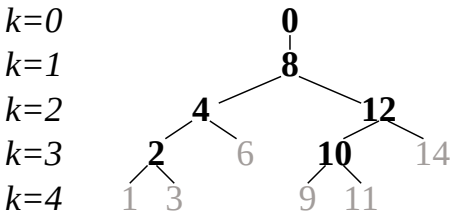


Figure 5.7: Address tree with synthetic nodes

nodes, whereas the gray ones are the synthetic nodes. Now, the overvaluation strategy does not lead to an error, and hence, there are no additional hops any more. As a drawback, each node has to maintain two entries in the DHT. This modification is thus only beneficial, if the saved forwarding hops had been more expensive than the increased maintenance traffic. The decision thus depends on the application traffic pattern and the observed amount of node churn.

5.4.1 Routing with Random Address Tree

Although a balanced address tree makes the routing success with $1 \sim 2$ steps, it is worth noting that Lite-Ring works even without balancing the address tree. In such cases, the address tree may be thought as a special random binary search tree. Devroye [40] proved that the height and saturation level of a random binary search tree with N nodes is bounded to $O(\log N)$. Hence, Lite-Ring without balancing process would also have a maximal route stretch of $O(\log N)$ and a very small average route stretch (cf. the simulations in Chapter 6.2.3).

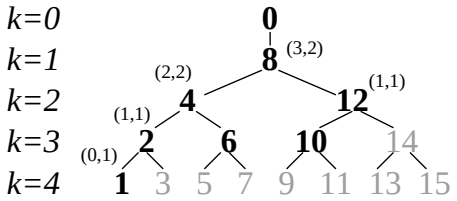


Figure 5.8: Unbalanced tree defined by depth. (The numbers in brace are k_{max} and k_{min})

5.5 Address Tree Maintenance

Two processes drive a balanced tree into imbalance: Not all of the aforementioned join mechanisms maintain the balance. Moreover, leaving nodes can cause an imbalance. In order to calculate which node is responsible for a given key precisely, Lite-Ring needs to keep the address tree balanced.

5.5.1 Tree Balancing with Depth

Balanced can be defined as follows: Let k_{max} be the maximal depth of the subtree rooted at node A. Let k_{min} be the minimal depth of a vacant position in the subtree. We call a node A unbalanced if

$$k_{max} - k_{min} \geq 1 \tag{5.3}$$

According to this definition, node 8 is the only unbalanced node in Figure 5.8. (Position 14 at $k = 3$ is vacant whereas position 1 at $k = 4$ is occupied.)

We thus require the nodes to run an additional balancing protocol: As part of the regular maintenance routine, each node reports its k_{max} and k_{min} together with the corresponding addresses to its parent node. Thus, unbalanced nodes can be recommended an address swap that can drive the tree into balance again. For example, node 8 in Figure 5.8 knows that node 1 can fill the vacant position 14 below node 12.

The advantage of this method is that the root of an unbalanced subtree knows exactly which node causes the imbalance and where the vacant position is. If there is only one unbalanced node, the tree can be balanced by one address swap,

A node should recommend such a balancing operation only if its two children are balanced. Thereby, the address swap occurs most deeply in the address tree and this operation can be performed parallelly in the subtrees.

5.5.2 Tree Balancing with Node Count

When the depth is used to determine the imbalance, each node can only recommend one address swap at once, because the information aggregated with this mechanism hides multiple unbalanced nodes. Lite-Ring thus contains a second mechanism that balances the tree. It is especially useful when a bunch of nodes has joined or left. To this end, Lite-Ring uses the aggregated node count to determine an imbalance.

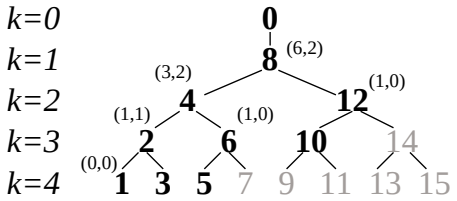


Figure 5.9: Unbalanced tree defined by node count. (The numbers in brace are N_l and N_r)

Let N_l be the number of nodes in the left subtree and N_r the number of nodes in the right subtree of some node A. We call A also imbalanced if

$$|N_l - N_r| > 1 \wedge \lceil \log_2 N_l \rceil \neq \lceil \log_2 N_r \rceil \tag{5.4}$$

Each node regularly reports N_l and N_r to its parent node. If a node detects an imbalance according to this definition, it recommends that $\frac{1}{2}|N_l - N_r|$ leaf nodes leave the overpopulated subtree and join again in the address range of the underpopulated subtree. For example, in Figure 5.9, node 8 recommends that two nodes from its left subtree move to its right subtree.

The advantage of this method is that multiple nodes can be shifted at once. The drawback is that the recommendation is more coarse-grained as with the first mechanism. Thus, such a shift might cause another imbalance in the underpopulated subtree. If this happens, Lite-Ring has to perform a second balancing operation.

Let's illustrate this with another example. In Figure 5.9, node 8 asks two leaf nodes from the left subtree to move to its right subtree, i. e. into the range $[9, 15]$. The recommendation propagates from node 8 to the leaf nodes, say to node 1 and node 5. Node 1 and 5 then choose a random address from the interval $[9, 15]$ and join the overlay as described in Chapter 5.3. Assume they choose addresses 9 and 11. Then the tree is still unbalanced. Node 12 now recommends one of these two nodes further to the range $[13, 15]$, and the tree is finally balanced.

The advantage of the balancing mechanism is to guarantee that within $\log N$ steps the unbalanced tree will be balanced regardless how large N_r and N_l differ.

5.5.3 Joint Balancing Mechanism

Implementation-wise both described balancing mechanisms run jointly. The aggregated position information is used to fill several vacuum positions with overweight nodes, and at the same time the node count is used to shift an appropriate amount of nodes from the overweight subtree to the other side. This makes the tree achieve its balanced state more quickly and more efficiently.

For example, again in Figure 5.9, node 8 can specify that one of the overweight nodes, say node 1, swaps to position 14, and another one moves to the range $[9, 15]$. Consequently, the tree is balanced in one operation.

5.5.4 Balancing Overhead

Let us now briefly examine the cost of the first balancing mechanism: Assume at first that the time interval between two successive joins is longer than the time to resolve a potential imbalance. The probability that a joining node needs to be swapped to a vacant position depends on the node density $b \in [0, 1)$ in the lowest level of the address tree. With probability b the new node happens to join as a child of a leaf node in the lowest level. Only in such a case, the tree becomes imbalanced. On average, this happens in 50% of the cases and the balancing overhead is $\frac{1}{2}$ shift operation per node. (Assume that the joining node naively joins at a random position, rather than using a proper address allocator.)

For the second balancing mechanism that uses the subtree size to shift some of the leaf nodes, any node can reach a position that balances the tree after at most $\log_2 N$ steps. A detailed analysis and simulation (cf. Appendix A and Chapter 6.2.1) shows that the average shift effort is slightly less than one operation.

The information about the current node count and tree depth is propagated from the leaf nodes to the root of the address tree periodically. The higher a node's level, the later it receives the information. Hence, the frequency of the imbalance check as well as the information propagation (i. e. regular subtree size and/or depth report) should be proportional to the node's depth.

Note that the assumption that the overlay has been balanced before a node joins is not realistic, because that would require an instant imbalance detection and correction. Nevertheless, Chapter 6.2.2 will show that even when many nodes join simultaneously, the maintenance effort is small and comparable to the results of the analysis in this section.

5.6 Handling Node Churn

When a node leaves the application overlay gracefully, it should remove its entry from the DHT, and inform its parent and children (in the address tree) and its neighbors (in the ring). If a node leaves ungracefully, the parent node, children node or the neighbors will detect the resulting inconsistency, for example, as part of the overlay maintenance protocol.

With respect to the tree structure, upon noticing that the parent node has left, one of the potentially two child nodes immediately fills the vacant position. If there is a choice between two children, it should prefer the older node. The respective child calculates the required information such as the address of its new parent node and its new child node. As a consequence of this mechanism, the stable nodes tend to move up in the address tree (assuming a power law distribution of node live times). This reduces the probability of churn for the inner nodes of the tree.

Since unstable nodes in the lowest level of the tree do not trigger address swaps or node shifts, the amount of balancing traffic reduces over time. (This will be shown in Chapter 6.2.5).

With respect to the ring structure, the 2-hop virtual neighbors can be cached to increase the ring consistency. Moreover, the overlay node can inquiry DHT for the potential virtual neighbors in case of its 2-hop virtual neighbor also fails.

5.7 Duplicate Address Detection

Temporary inconsistencies may lead to address collisions. For example, when a node leaves ungracefully and its two children cannot find each other quickly enough, they both might swap into the vacant position. In order to detect such a duplicate address allocation, all nodes regularly inquire the DHT for their own overlay address. If the DHT returns more than one result, one of the affected nodes has to swap to another address. Again node stability can be used as criterion to decide which node has to swap: The younger node, who has a smaller (expected) remaining life-time, has to re-join the overlay at a new address.

5.8 Complexity Comparison

In this section the complexity of Lite-Ring is compared to those of ReDiR [100], Diminished Chord [68] and the de-facto approach of providing KBR service to multiple applications (i. e. one Chord-like full-fledged overlay per application). Let M be the overall number of nodes, M^* be the number of nodes in DHT and N the average number of instances per application.

Table 5.2 gives the complexity classes for the important overlay operations and states. Their derivation can be found in [100] [68] and [114].

Note that by distinguishing M and M^* we also incorporate the effect of Proximal Neighbor Selection (PNS) and Proximal Route Selection (PRS) in DHTs. Even though an operation in a DHT takes $O(\log M^*)$ overlay hops, these hops have greatly differing actual cost and latency. As a result, the cost of DHT operations scales sub-logarithmically in practice [54][98].

As shown in Table 5.2, Lite-Ring has several advantages. Most notably, Lite-Ring requires only one lookup in the DHT to route a message to a given application-and-key pair. The reason is the predictable overlay address assignment which enables the local calculation of the responsible node's overlay address. And only one DHT lookup is needed to resolve its transport address. Moreover, Lite-Ring can even achieve a very small join complexity as described in Chapter 5.3.

ReDiR uses a level predictor to guess the proper partition where the responsible node may lies. If it fails or if the requested key happens to fall into the subsequent partition, more lookups are required – in the worst case up to $O(\log N)$ lookups. More worse, since ReDiR has no overlay structure and depends strongly on the information it stored on the DHT. The address lookup will fail if the related DHT operations malfunction, e. g. the capacity of some DHT nodes is full.

	ReDiR	Chord	DimChord	Lite-Ring
App. Inst. Join	$O(\log N \cdot \log M^*)$	$O(\log^2 N)$	$O(\log M)$	$O(\log\log N \cdot \log M^*)$
State per App. Inst.	$O(1)$	$O(\log N)$	$O(\log M)$	$O(1)$
Load Ratio	$O(1)$	$O(1)$	$O(\log M)$	$O(1)$
Routing (worst case)	$O(\log N \cdot \log M^*)$	$O(\log N)$	$O(\log M^* + \log M)$	$O(\log M^*)$

Table 5.2: Complexity comparison between Lite-Ring and other approaches

Unlike ReDiR, Lite-Ring has a light-weight but complete ring structure, therefore it can guarantee the consistency of message delivery even when some DHT operations fail. This feature will be shown in Chapter 6.2.4 and Chapter 6.3.

Chapter 6

Simulation and Test

In this chapter, I present some simulation results of the Lite-Ring overlay protocol with the OMNeT++ simulator [121]. Moreover, at the end of this chapter, I also present the results of a real-world test using a Lite-Ring implementation based on OpenDHT [100].

6.1 OMNeT++ Simulator

The OMNeT++ simulator [121] is an extensible, modular, component-based C++ simulation library and framework. More importantly it is free for the academic use. There are quite a few groups in Germany, who have contributed so-called frameworks that provide various protocol implementations. OMNeT++ features a simple, object oriented design, which leads to good scalability. Therefore, OMNeT++ is particularly well suited for performance evaluations of large networks.

In OMNeT++, the network packets are exchanged between "modules" and the events occur inside "modules". The events and the network packets are called "messages".Modules are further classified into simple modules and compound module. A simple module is the basic construction block of OMNeT++ simulations because it implements various computer components, such as a node's CPU, Ethernet stack, etc. A simple module has the following virtual functions:

- `virtual void initialize()`,
- `virtual void handleMessage(cMessage *msg)`,
- `virtual void finish()`.

A compound module in OMNeT++ is a group of simple modules. Moreover, compound modules can group other compound modules at a higher level. A network in OMNeT++ is a compound module at the highest level.

6.2 Simulations

The OMNeT++ simulator was used to simulate Lite-Ring and to evaluate its performance. Each simulation ran 10∼30 times with different random seeds.

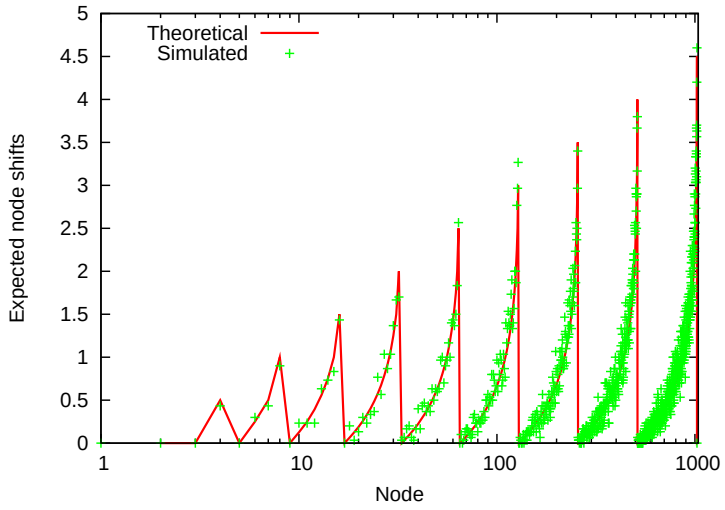


Figure 6.1: Expected node shifts at the n^{th} node for balancing the address tree with node count.

6.2.1 Sequential Joins

In the first scenario, 1024 nodes join the application overlay sequentially. The nodes start to join only after the overlay has become balanced.

Figure 6.1 shows the per node balancing effort when the nodes use the node count to discover an imbalance. The balancing effort is the expected number of address shifts that a joining node might perform when joining the application overlay. As expected from the analysis in Chapter 5.5, the worst case occurs when the binary tree is almost fully populated. The balancing effort in that case is logarithmic to the network size. (The derivation of the theoretical result can be found in Appendix A.)

Figure 6.2 shows the balancing effort as average over all nodes that have already joined. We see that the average effort is limited by a threshold of one additional operation. Again, the simulation result matches the expected small cost of the balancing process.

Figure 6.3 and Figure 6.4 show the balancing effort with the depth criterion, i. e. when the nodes can use precise information about the vacant positions in the address tree. From Figure 6.3, we see that even the maximal expected number of additional operations does not exceed one, whereas in Figure 6.4, the average balancing effort is below 0.5 operation.

Due to the fact that at most one node needs to be replaced in this scenario, the combined version with both depth and count performs identically to the depth criterion.

6.2.2 Concurrent Joins

The sequential join process that is assumed so far is unrealistic. In practice, nodes might join the overlay concurrently, maybe even in bursts. They will not wait for the overlay to become balanced.

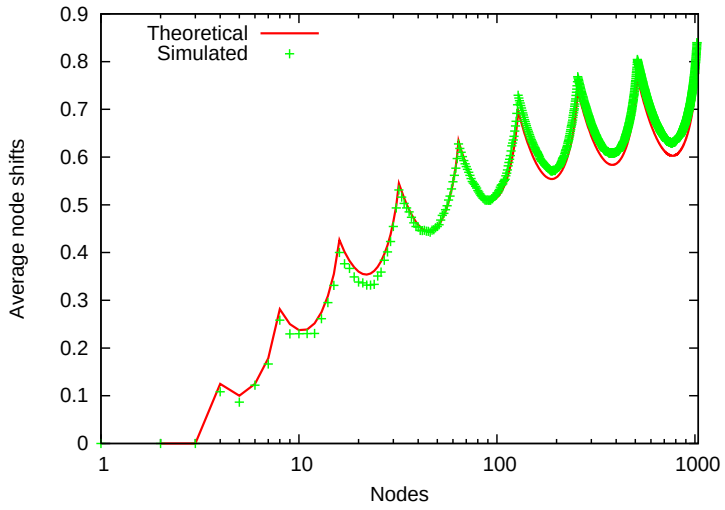


Figure 6.2: Average number of the node shifts of the first n nodes for balancing the address tree with node count.

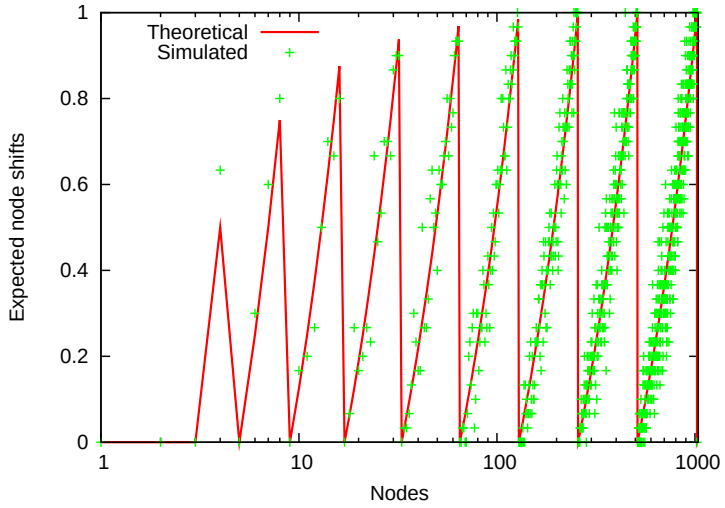


Figure 6.3: Expected node shifts at the n^{th} node for balancing the address tree with vacant position.

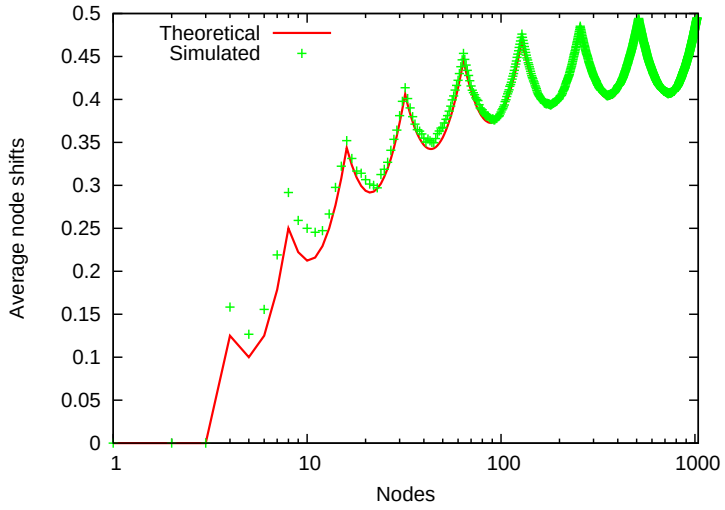


Figure 6.4: Average number of the node shifts of the first n nodes for balancing the address tree with vacant position.

In order to simulate concurrent joins, we let 16-1024 nodes join the overlay at a rate of one node per second and set the time interval of maintenance operations as 5 seconds. Figure 6.5 and Figure 6.6 show a comparison of the three balancing methods that we have presented in Chapter 5.5. Although the combined balancing method has some overhead in terms of node shifts, it balances the tree most quickly.

We can find out that the balancing effort per node remains below one operation (cf. Figure 6.5). This agrees with the analysis in Chapter 5.5.

Figure 6.7 depicts the distribution of the join attempts of 1024 nodes with different join patterns. The node-count based balancing mechanism (cf. Chapter 5.5) was applied. Surprisingly, the distributions are similar and the only difference is that the distribution of the concurrent join scenario has a slightly heavier tail. Based on this observation, we can conclude that the analysis based on sequential join in Chapter 6.2.1 is also applicable to the scenario of concurrent join in some degree.

6.2.3 Random Address Tree

As stated in Chapter 5.4, the routing overhead of Lite-Ring can be very small in cases where the address tree is balanced. However, it is also interesting to know the routing overhead in cases where the address tree is not balanced. For the analysis in this section, the balancing procedure is switched off and no synthetic node is used.

In Figure 6.8, 10^5 nodes join the Lite-Ring overlay sequentially. From the figure, we can see that the maximal height (H_{max}), the average node depth (H_{ave}) and the saturated level (H_{sat}) of the address tree are logarithmic to the node count.

Suppose that each node in the Lite-Ring overlay knows the node count, thus knows the average depth (H_{ave}) of the address tree. When routing a message to

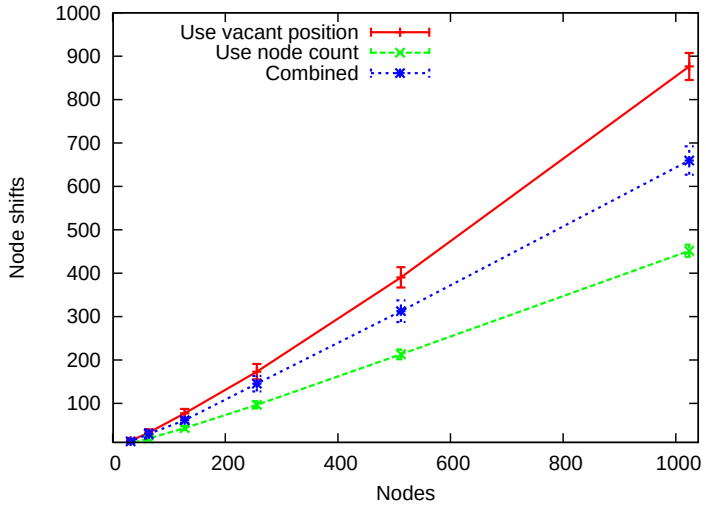


Figure 6.5: Comparison of different balancing mechanisms (node shifts)

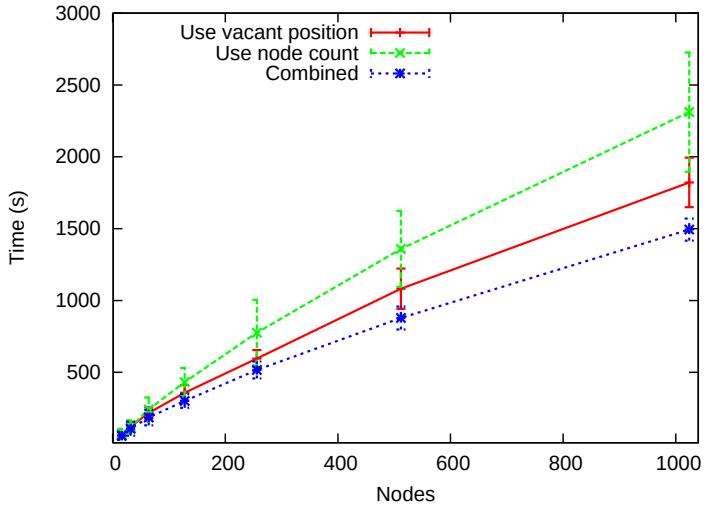


Figure 6.6: Comparison of different balancing mechanisms (time)

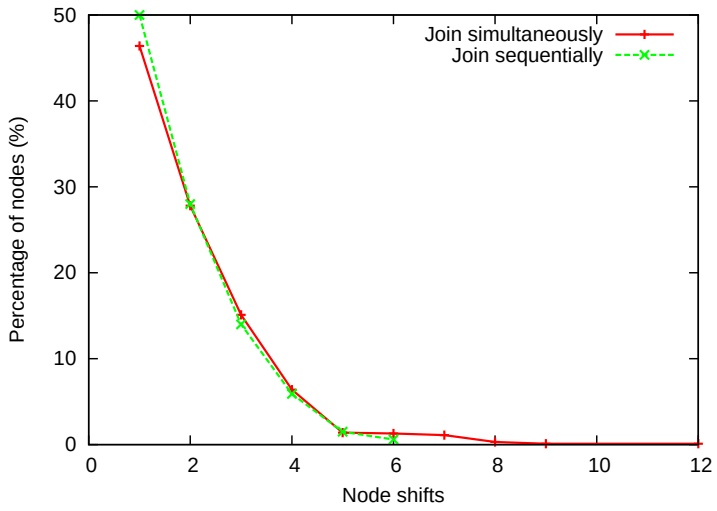


Figure 6.7: Distribution of the node shifts with 1024 nodes

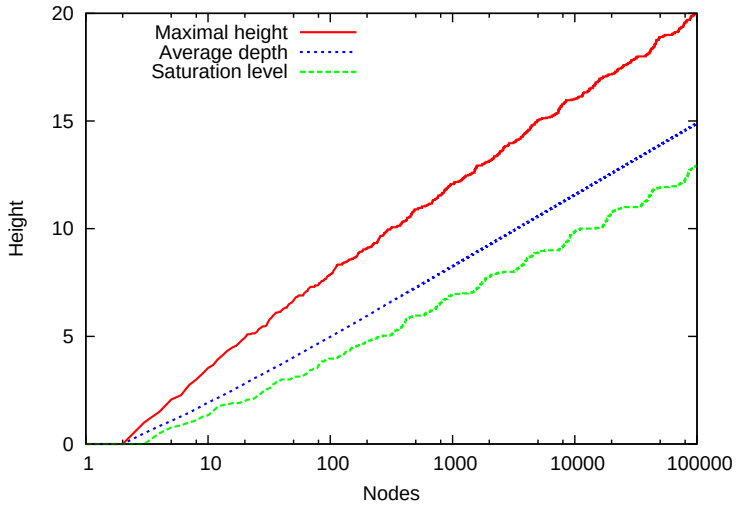


Figure 6.8: The maximal height, the average node depth and the saturated level of the unbalanced address trees with up to 10^5 Lite-Ring overlay instances

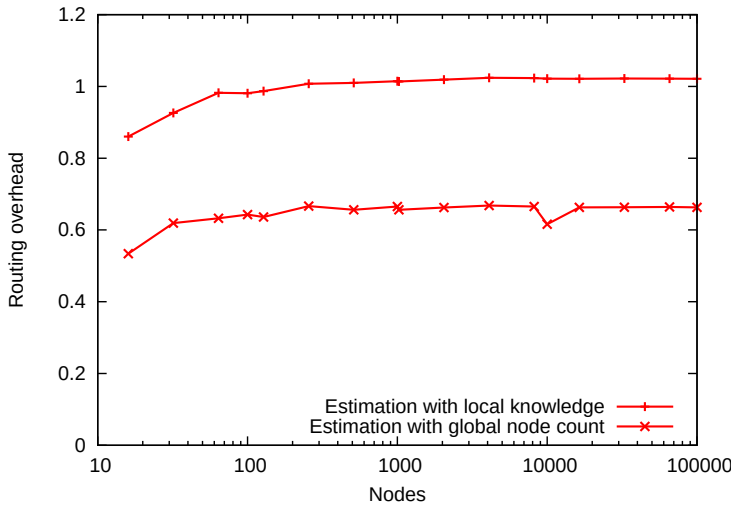


Figure 6.9: Routing overhead with random address tree (with different estimation strategies)

a certain key, the node assumes a fully populated address tree of height H_{ave}, then it polls the DHT for the transport address of the node responsible to the key (see Chapter 5.4). As shown in Figure 6.8, when the Lite-Ring network consists of 10^5 nodes, the routing overhead in worst case is $H_{max} - H_{ave} = 5$. (An overhead of one means either one additional DHT inquiry or one extra route step within Lite-Ring.)

Without the global knowledge of the node count, the node can use the local knowledge (i. e. the predecessor ID or the successor ID) to estimate the node count (see Chapter 5.4). A simple estimation strategy used in this simulation is:

$$H_{ave}^{'} = max(H_{self}, H_{successor}) - 1$$

In the worst case that the destination node lies at the depth H_{max} and both the node's own ID and its successor ID lie in the saturated region, the node gets $H_{ave}^{'} = H_{sat}$ and its routing overhead is then $H_{max} - H_{sat} = 7$. (As shown in Figure 6.8 with the network size of 10^5.)

Figure 6.9 depicts the average routing overhead of Lite-Ring using the aforementioned two estimation strategies. The network size varies from 16 to 10^5. The figure shows that the routing overhead with global node count knowledge is below 0.7; whereas the overhead with local knowledge is about 1.0. That means, Lite-Ring can work with unbalanced address tree with only small routing overhead,

In practice, a node might have more knowledge than knowing only its successor. For example, nodes could learn the existence of other nodes by receiving or routing payload messages. The more knowledge a node has, the more precise it can estimate the overlay size and the smaller routing overhead it generates. (Note that the reduction of the routing overhead is limited to 0.7.)

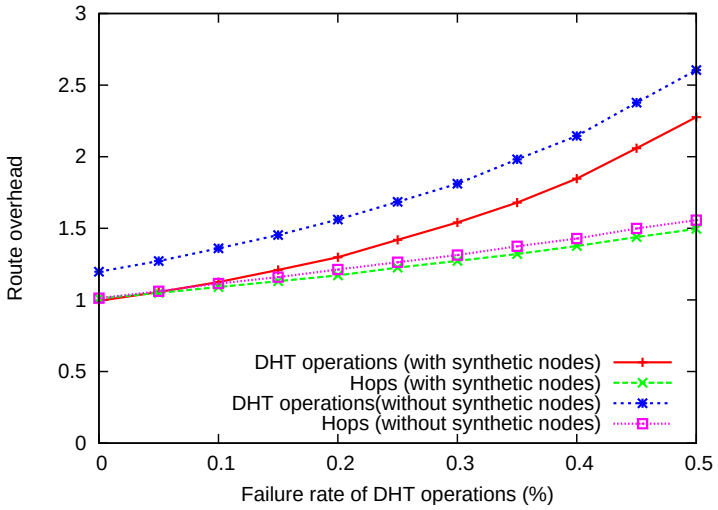


Figure 6.10: Routing overhead with an unstable DHT

6.2.4 Using an Unstable DHT

The Lite-Ring protocol depends on a public DHT service. Regardless of using a third party service or integrating the DHT into Lite-Ring, such a service might be unstable. It might considerably delay some messages, and operations might fail sporadically.

In the following simulation, 800 nodes have joined a Lite-Ring overlay and an unstable DHT service is used, in which up to 50% of the DHT operations fail. The performance of the Lite-Ring overlay that uses synthetic nodes is compared to the version that does not.

From Figure 6.10 we see that with an increasing failure rate, the number of required DHT operations also increases. In the case where synthetic nodes are used, that number is approximately 20% below the case without synthetic addresses. This corresponds to the analysis result (cf. Equation 5.2). When the failure rate increases, the number of forwarding hops also increases slightly.

It is worth pointing out that in this simulation all messages reach their correct destinations. Thus, this simulation also demonstrates that Lite-Ring is robust even in face of an unstable DHT.

6.2.5 Node Churn

Not only the DHT can be error-prone, but the overlay nodes themselves might also operate irregularly. In the next simulation, 50% out of 64 overlay nodes die ungracefully after an average of 2 minutes (from an exponential distribution). When a node dies, another node joins so that the entire number of nodes remains constant.

As stated in Chapter 5.6, the nodes have to update their address when their parent node is gone. In such a case, the respective node moves up the address tree. As a consequence, the stable nodes tend to move up and stay in the upper levels of the address tree, whereas the unstable nodes remain at the lowest level

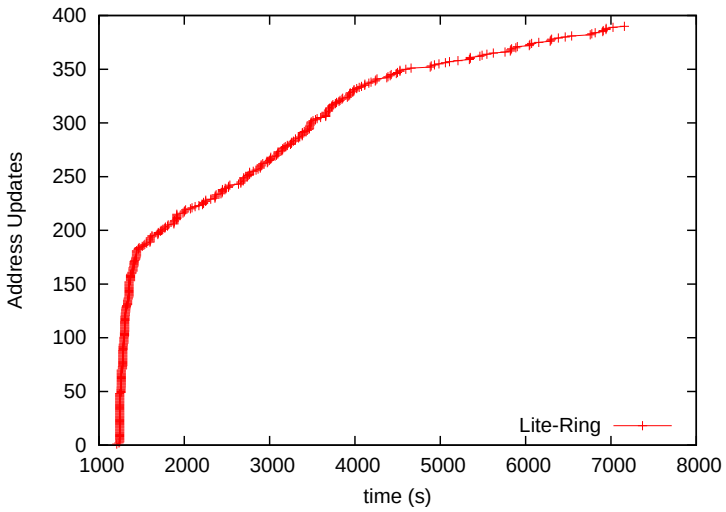


Figure 6.11: Cumulative address updates during node churn

where they joined the overlay.

From Figure 6.11, we see that the update frequency decreases over time, because more and more stable nodes populate the upper levels of the address tree. Churn at the lowest level – where the unstable nodes sit – does not trigger address updates. From this simulation, we thus conclude that Lite-Ring is in fact adaptive. It adjusts the node distribution in the overlay so that it achieves a robust and stable performance.

It is notable that the length of the forwarding path is still approximately one hop, because Lite-Ring does not use leaf nodes in the address tree as intermediate hops for routing. Hence their churn does not have an impact on the message delivery among the stable nodes in the fully populated level of the address tree.

6.3 Tests with OpenDHT

In order to compare the performance between Lite-Ring and ReDiR, Lite-Ring has been also implemented in a Linux environment and run with OpenDHT's XML-RPC interface [100]. The ReDiR performance measured is based on the code provided by the OpenDHT web page [1].

In this experiment, 30 overlay instances connect to the 10 closest OpenDHT nodes and the synthetic nodes are also used. Each instance sends 100 messages to 10 randomly selected keys at a rate of one message per second. After the experiment with Lite-Ring, ReDiR is run under the same conditions.

Fig 6.12 shows the message delays on a logarithmic scale. For more than 90% of the messages, Lite-Ring has a significantly smaller delay, because Lite-Ring needs only one DHT `get` operation to correctly route any message. On contrary, ReDiR needs $1.2 \sim 1.3$ DHT `get` operations on average [100].

Since the nodes in Lite-Ring maintain additional state about some other nodes, e. g. in the ring and address tree, some messages can be delivered without any DHT operation. That's the reason why there are some messages having

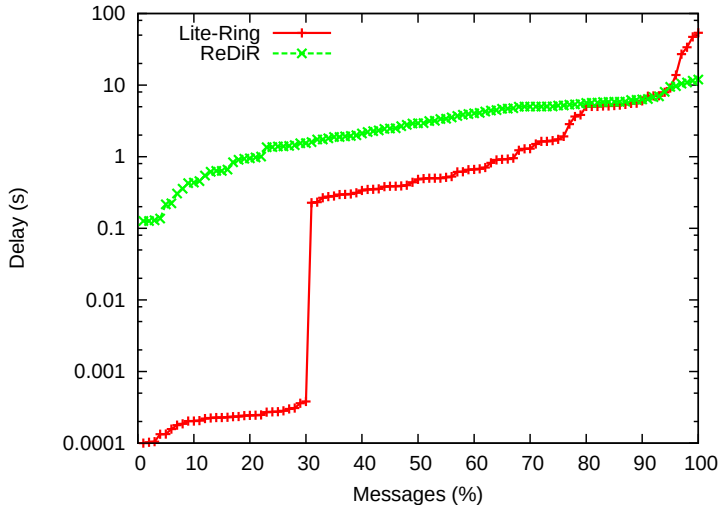


Figure 6.12: Comparison between Lite-Ring and ReDiR, both run with OpenDHT

extremely low delay.

Most importantly, *Lite-Ring achieves 100% correct message delivery*, whereas ReDiR achieved only 40%. The reason that Lite-Ring remains always consistent, is Lite-Ring's light-weight overlay structure. It enables the receiver of a message to detect an inconsistency (cf. Chapter 5.4). It can then either trigger further DHT operations that resolve the inconsistency, or it can route the message directly within the application overlay. The increased effort causes some packets (10%) to have larger delay than the maximal delay in ReDiR.

Finally, we note that OpenDHT was also a good example how Lite-Ring copes with an unstable DHT. In this experiment, the connections to the OpenDHT nodes had to be re-established 21 times during the 3 minutes measurement. Moreover, about 17% DHT put operations failed due to insufficient capacity on the OpenDHT nodes.

Chapter 7

Conclusions

7.1 Summary

In this part of the dissertation, I have presented Lite-Ring, a light-weight overlay structure that efficiently creates and maintains key-based routing overlays for multiple applications.

Lite-Ring requires only $O(1)$ state per application instance. Similarly to ReDiR, a previous proposal for such a scenario, Lite-Ring stores some of its state in a DHT. But unlike ReDiR, it can route messages directly to their destination after only one DHT lookup. This reduces the message delivery latency significantly (as compared to ReDiR). Moreover, Lite-Ring detects inconsistencies in the overlay structure so that it achieves 100% correct message delivery (as compared to 40% with ReDiR) even in case of an unreliable DHT.

I have presented some mathematical analysis of Lite-Ring's performance. I have also confirmed the analysis with simulations and a measurement in PlanetLab. In addition, it has been demonstrated that Lite-Ring is stable in face of node churn and an unreliable DHT.

7.2 Practical Issues

Before the end of this part of work, I would briefly discuss several important issues that are related to the real world deployment of Lite-Ring.

7.2.1 Impact of NAT

As many other P2P protocols, Lite-Ring has to deal with the Network Address Translation (NAT) [110] problem in the Internet. A straightforward solution is to use Interactive Connectivity Establishment (ICE) [103] or Session Traversal Utilities for NAT (STUN) [102] to obtain the public IP address and port number that the NAT has allocated for the node.

Nevertheless, in some cases, a node cannot obtain its public transport address. Even worse, it may erroneously use its private address to join the Lite-Ring overlay. In such a case, the other nodes cannot deliver messages to that node directly, but rather route them to the predecessor of that node. This would result in a slightly unbalanced load in the overlay. Moreover, such nodes cannot

properly serve as address allocators, so that the tree structure would become imbalanced and the route stretch would increase.

To avoid these problems, Lite-Ring does not allow to register private addresses. Moreover, if e. g. a firewall blocks a public address for address allocation, Lite-Ring assumes that the respective node ungracefully failed and restarts the bootstrapping process at other position. Hence, the described imbalance is a transient problem, only.

7.2.2 Load Balancing

Karger et al. [69] pointed out that the load balancing depends on two factors: address distribution and load distribution. In case of a uniformly distributed load, the address distribution should be as uniform as possible. Accordingly, Karger et al. suggested $\log N$ virtual servers to achieve a uniform address distribution.

Instead of introducing redundant virtual servers, Lite-Ring's address assignment scheme enforces a strictly uniform addresses distribution. Typically, the size of the address intervals differs only by a factor of two at most, Hence, Lite-Ring achieves predictable load balancing properties for uniformly distributed load without additional overhead.

For a non-uniform load distribution, the address space has to be skewed correspondingly. Instead of balancing the tree in terms of node depth or node count, Lite-Ring can then use the node load to define the balanced state of the tree. It is not difficult to imagine that the heavily loaded address space contain more overlay nodes. (A detailed description and analysis of this feature is beyond the scope of this thesis.)

7.2.3 Public DHT Service

When we began our work on Lite-Ring, OpenDHT was the most well-known and stable DHT service. Meanwhile, it has been abandoned. As an alternative to OpenDHT, Rhea suggested Amazon Dynamo [38], a commercial DHT storage service. Another alternative would be to use some open DHT library, e. g. Pastry [104] or Bamboo [99], to build a new public DHT service. However this would require a lot of development and maintenance work.

7.2.4 Security Aspects

Lite-Ring has been designed as a generic structured peer-to-peer overlay system. Such systems generally suffer from vulnerabilities, especially several different types of denial-of-services attacks. Lite-Ring's predictable address assignment makes some of these attacks easy, while impeding others. Overall, Lite-Ring has to be protected against malicious nodes, as it is the case with all P2P overlays [20]. (A discussion of such security enhancements would apply to structured P2P overlays in general. This is considered beyond the scope of this thesis.)

7.2.5 Address Swap

In case of node churn, Lite-Ring swaps node addresses. This causes an increased traffic for applications that then have to exchange data among the respective

peers. Luckily, this address swap is predictable: A child node always takes the place of its parent node. Hence, a Lite-Ring based application could replicate its data preferably to its child nodes in the address tree so that the churn-induced address swap does not create additional traffic.

In case of a mere tree balancing operation, the address swap is not predictable. However, this balancing process happens only rarely. Moreover, as mentioned in Section 5.4, Lite-Ring does not require the tree to be strictly balanced. It is up to the specific application to choose the trade-off between data exchange traffic and routing overhead.

Part III

Lite-Ring in Wireless Ad-hoc Networks

Chapter 8

Overview

Since their appearance in 2001, structured overlay networks in the Internet have been a hot topic and been extensively studied. At the same time, routing in large wireless ad-hoc networks, e. g. Mobile Ad-hoc Networks (MANETs), Wireless Sensor Networks (WSNs) and Wireless Mesh Networks (WMNs), has also attracted a large amount of attention.

Although these two research topics appear to be orthogonal, they have some common properties, such as self-organization and decentralization. Since years, there have been efforts trying to combine peer-to-peer and ad-hoc networks, i. e. building structured overlays in wireless ad-hoc networks [50, 15, 129, 39, 47, 41]. However, all these approaches aim at one single KBR application. None of them is able to offer an efficient KBR service to multiple applications simultaneously.

This chapter discusses the feasibility of using Lite-Ring to solve this problem, i. e. to provide an efficient KBR service to multiple applications in a wireless ad-hoc network. In the following, I first give a case study to illustrate an application scenario of a structured overlay in a wireless ad-hoc network; after that the common properties of P2P networks and wireless ad-hoc networks are stated. Next, I present a snapshot of the current solutions and discuss their infeasibility of providing an efficient KBR service for multiple applications. At last, the goal and challenges of the current work are discussed.

8.1 Case Study - Object Tracking with DHT

Our case study in [43] uses object tracking in a WSN as an example to demonstrate the feasibility as well as the performance improvement of applying P2P techniques in a wireless ad-hoc network.

8.1.1 Background

With the technological improvements, large WSNs tends to be more and more used in practice. Along object tracking is one of the major applications, other applications can be remote monitoring, event detection, etc. In object tracking systems, there is typically a central instance, which needs timely information of, e. g. the location of one or more objects in the WSNs.

For example, a researcher may want to track the wild deers that live in a large national park. She might want to do so in real-time and she might also want to locate any individual at any instant in time. But she will typically only request the location of a few deers at a time. The deers might have preferred places and typical times to move between these places. They might move individually or in herds. Many other such applications are conceivable, including commercially relevant applications such as asset tracking and applications to national security.

For the purpose of this case study, sensor nodes and tracked objects are distinguished. The tracked objects are even more resource-limited than the sensor nodes. One may consider the tracked objects as being tagged with Radio Frequency Identification (RFID) tags or small Wireless Local Area Network (WLAN) tags. The node that needs the information about the tracked object is called the *inquirer* or *sink*, and the sensor node that contains the requested (location) information about the tracked object is the *source*.

8.1.2 Straightforward Solutions

There are two straightforward solutions for object tracking: the "pull" approach and the "push" approach. In the "pull" approach, the sink floods the network with a request message whenever it needs the location data of a particular object. A sensor node that receives such a request will answer with the location data of the requested object. Obviously, such an approach does not scale to the high frequency of the location requests in large networks.

The other straightforward approach is to push the location information to a central location registry. The inquirer can then retrieve the data that it needs directly from the registry without flooding the whole network. Note that the push approach does not scale either. In particular, it is not suited for large networks with many highly mobile objects: Because whenever an object moves from the vicinity of one sensor node to that of another sensor node, both of these nodes have to send an update to the central location registry. Most likely, many of the updates will be overwritten in the registry without having been read by the monitoring application. In our assumed scenario the publishing nodes cannot know which data might be requested. So they have to publish permanently all the data. However, increasing the update intervals of the sensor nodes does not help, since this reduces the timeliness of the data that happens to be requested.

8.1.3 Application of DHT

To overcome the scalability problems of the aforementioned straightforward approaches, a DHT can be used to hold the (location) information about the objects. Since Scalable Source Routing (SSR) is a DHT-inspired routing protocol, it supports native KBR semantic. Thus, it is feasible and easy to build the DHT on top of SSR.

When detecting an event of an object, the sensor node hashes the object ID to some key (inside the SSR address space) and lets SSR route the object information to the sensor node, whose address is the closest to that key. Only the respective node stores or updates the object information. When the sink needs to get the information of that object, it also hashes the object ID to the (same) key and lets SSR route the inquiry to that respective sensor node.

The simulation results in [43] show the significant performance improvement of the DHT approach to the two straightforward solutions for reasonably expected event and request rates.

8.1.4 Summary

From this case study, we can see that although P2P techniques originally became popular in the Internet, these techniques are also applicable in wireless ad-hoc networks.

8.2 Characteristics of P2P and Wireless Ad-hoc Networks

Both kinds of the networks, i. e. Peer-to-Peer (P2P) networks and wireless ad-hoc networks, are usually deployed in totally different areas: one as overlay in the Internet and the other in standalone and often autonomous region. However, they have some common characteristics [17].

- *Node dynamicity:* The nodes, i. e. the network participants, can join and leave the network at any time. This leads to a frequently changing topology.
- *Self-organization:* Both networks have to bootstrap on their own and they have to adapt to the changing topology all the time.
- *Decentralization:* Neither of these two networks needs central nodes to administrate the participants. The participants in the network have only local knowledge, from that they need to derive their (sometimes very limited) global information.
- *Multi-hop routing:* Both networks require multi-hop communication. The routing algorithm should be scalable to a potentially very large network size.

Of course, there are also differences between these two networks:

- *Infrastructure:* P2P networks were originally designed as applications in the Internet, hence they rely on the IP routing infrastructure; whereas wireless ad-hoc networks have no underlying infrastructure and have to solve the end-to-end routing problem.
- *Bandwidth:* P2P networks are supposed to have a sufficient bandwidth, e. g. normal peers typically have the usual Digital Subscriber Line (DSL) bandwidth and some super nodes may have Internet connection through a campus network. In contrast, the bandwidth in wireless ad-hoc network depends on the environment and the hardware, which could be IEEE802.11 [59], Bluetooth [60], IEEE802.15.4 [61], etc. Accordingly, the bandwidth varies from a few KBit/s to several MBit/s.
- *Service:* P2P networks are assumed to be able to make use of many diverse services in the Internet, e. g. King [55] project utilizes the DNS servers

as the landmarks to predict end-to-end delay, others [25] use Content Distribution Network (CDN). On the contrary, the services in wireless ad-hoc networks are usually quite limited and sometimes very costly, e. g. location service through the accommodation of GPS receivers.

- *Reliable Transmission:* Since P2P networks use the pre-existing Internet infrastructure, message transmission can be considered quite reliable especially when the Transmission Control Protocol (TCP) is used. However, in wireless ad-hoc networks, the performance of the conventional TCP protocols is not satisfactory, as TCP cannot distinguish the packet loss from congestion and interference. Recently, many optimizations or alternative methods have been proposed for TCP in wireless environments [49, 16, 127]. However, all of them lack wide validations and tests.

8.3 KBR Solutions in Wireless Ad-hoc Networks

Due to the aforementioned properties, the application of P2P overlays in wireless ad-hoc networks is more complicated than that in the Internet. Nevertheless, many efforts [50, 15, 129, 43] have been undertaken to integrate structured overlays into wireless ad-hoc networks. Castro et al. [17] summarized these efforts into three categories:

- *Layered solutions* deploy a structured overlay protocol directly on top of a conventional ad-hoc routing protocol. These solutions do not introduce any changes either in the overlay protocol or in the ad-hoc routing protocol. However, the overlay maintenance procedure generates a lot of traffic in the network layer, which leads to a poor scalability [31, 18].

- *Cross-layer solutions*, e. g. CrossROAD [39] and MADPastry [129], modify the ad-hoc routing protocol and integrate the overlay information into the routing control messages. The advantage of these solutions is the significant reduction of the overlay maintenance overhead in the network layer.

- *Integrated solutions*, e. g. SSR [50], VRR [15] and DART [47], merge the structured overlay into the network layer and achieve higher performance by maintaining only one unified layer.

These solutions can provide a plain KBR service to a single application in a wireless ad-hoc network. However, they will not work, at least not efficiently, if multiple applications require the KBR service simultaneously. The layered solution and the cross-layer solution build multiple independent overlays, which generate an enormous maintenance overhead which could overload the network. As for the integrated solutions, all the node are assumed to be homogeneous and only one KBR application is supposed to be deployed on the node. Hence, they can not be applied for multiple applications, either.

8.4 Goal and Challenges

To my best knowledge, there is no applicable solution of providing the KBR service to multiple applications in wireless ad-hoc networks so far. This part of the dissertation discusses the feasibility of using Lite-Ring to solve this problem.

From the view of the application, Lite-Ring will work transparently as a conventional KBR routing protocol, i. e. route the `key-data` pair to the destination node, which runs the same application and whose ID is the closest one to the key.

Although there are some common properties of P2P networks and wireless ad-hoc networks, the different characteristics between these two kinds of networks make the construction of the Lite-Ring overlay in wireless ad-hoc networks more difficult than in the Internet. These challenges include end-to-end routing, reliable delivery, a DHT service and bootstrapping.

8.4.1 End-to-end Routing

The existing Internet infrastructure enables end-to-end routing on the global scope, In theory, any two nodes can establish a connection as long as they know their mutual IP addresses. However, in wireless ad-hoc networks, due to the resource restriction, e. g. memory and bandwidth limitation, end-to-end routing is still an open topic. Since neither the proactive routing protocols nor the reactive routing protocols work well in large wireless ad-hoc networks, a scalable end-to-end routing solution has yet to be found before we can build the Lite-Ring overlay networks.

8.4.2 Reliable Delivery

In wireless ad-hoc networks, packet loss can be caused by many reasons, such as congestion or interference. Although quite a few reliable transport layer protocols [49, 16, 127] have been proposed for wireless networks, none of them is well standardized or widely applied. Therefore, the datagram mode is used for message delivery in Lite-Ring. The task of reliable communication is thus shift to the applications that use the Lite-Ring.

8.4.3 DHT Service

In the Internet, there are some public DHT services which can be utilized to build the Lite-Ring networks. However, no such services is available in wireless ad-hoc networks. As mentioned in Chapter 3.2.2, SSR offers the KBR semantic. Therefore, it is feasible to build a primitive DHT service on top of SSR.

Obviously, Lite-Ring can benefit from a reliable DHT. However, it is not trivial to implement a reliable DHT in an unstable environment [35, 101]. Fortunately, as illustrated in Chapter 6.2.4, Lite-Ring can work well even with an unreliable DHT. So, in stead of building a complicated and reliable DHT, a simple and thus unreliable DHT suffices for the Lite-Ring network.

8.4.4 Bootstrapping

As is well known, one of the principles of many P2P applications is that every overlay node can serve as a bootstrapping node. Thus, the bootstrapping problem is how to let the new coming nodes find one of the existing nodes.

A widely applied solution is to use a relatively reliable public resolver, e. g. a web page, where some stable overlay nodes are published. These nodes are usually very stable and online in most time. With the help of these nodes, the new coming nodes can bootstrap properly.

In wireless ad-hoc networks, such a centralized bootstrapping method is not feasible, since there is no reliable third party for such publications. In contrast to the centralized method, the Lite-Ring protocol solves the bootstrapping problem in a decentralized manner in that a random overlay node can be found by leveraging the DHT (cf. Chapter 5.3).

Chapter 9

Design

This chapter briefly describes the design of a system, that can provide a KBR service to multiple applications simultaneously in wireless ad-hoc networks. In the following, this system will be called *Lite-Ring system*. This system includes not only Lite-Ring modules, which implements the Lite-Ring protocol, but also further two modules: a DHT module and an end-to-end routing module.

9.1 The Lite-Ring System

The Lite-Ring system provides the extended key based routing semantics to various applications, i. e. `route({key, app-id}, value)`. For each application instance, there exists a corresponding Lite-Ring module. That means, multiple Lite-Ring modules could exist in a Lite-Ring system.

As stated in Chapter 4.3, Lite-Ring requires a DHT service to build its overlay. However, there is no public DHT service available in the wireless ad-hoc networks. Hence, a DHT module has to be integrated into the Lite-Ring system. Moreover, since there is no IP routing infrastructure in wireless ad-hoc networks, an end-to-end routing module must also be comprised in the Lite-Ring system. The resulting structure is illustrated in Figure 9.1. Concerning the adaptivity of the Lite-Ring system to networks with different hardware techniques, the MAC-layer module is currently excluded.

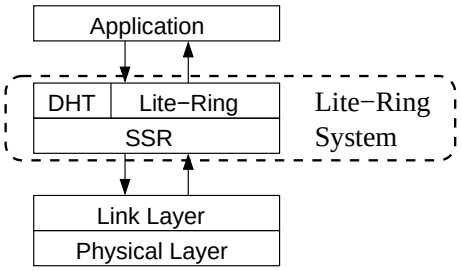


Figure 9.1: The Lite-Ring system

9.1.1 End-to-End Routing

In order to ensure connectivity between any two nodes, the end-to-end routing module has to be deployed on each node. As stated in Chapter 3.2.2, the Scalable Source Routing (SSR) protocol is a DHT-inspired routing protocol in the network layer. It achieves good performance with only a small routing cache and little maintenance overhead. It is a scalable routing solution for large wireless ad-hoc networks and requires neither flooding to determine a forwarding path nor localization of the nodes on a 2-dimensional surface. More importantly, it supports both the end-to-end routing semantic and the Key Based Routing (KBR) semantic. Based on the observations above, SSR is chosen to build the routing module. Thus, in the following context, the end-to-end routing module is also called SSR module.

9.1.2 Light Weight DHT

A DHT module is required in the Lite-Ring system and it should support at least three primitive storage services: `put`, `get` and `remove`. Although it would have been favorable to include an efficient and reliable DHT module (eventually with aggregation function) into the Lite-Ring system, the development cost and the maintenance overhead would be enormous. Considering the fact that the Lite-Ring can achieve good performance even with an unreliable DHT (rf. Chapter 6.2.4), a light-weight primitive DHT module suffices for the Lite-Ring system.

In this light-weight DHT, the `key-data` entries are stored as soft-state without replication. The users of this DHT must refresh (and replicate for reliability) their data regularly. Besides the Lite-Ring system, the applications may also directly use the light-weight DHT, no matter if they use the Lite-Ring system or not.

The DHT module is built on top of the SSR module and is implemented on each node within the network. This construction is based on the following consideration:

- As a network layer protocol, SSR runs on every node in the network. Its key-based routing function can be used to build the DHT module.
- If only some nodes run the DHT, an extra metric is required to determine whether a node should be deployed with the DHT module or not. That would make the construction more complicated.
- It is even more challenging to let a node without DHT module find a node with DHT module to perform DHT operations than to include a DHT module on each node in the Lite-Ring system.

9.1.3 The Lite-Ring Module

The Lite-Ring module implements the Lite-Ring protocol and uses the SSR module to establish its overlay links. To this end, the end-to-end routing mode of SSR is used. Each Lite-Ring module is initialized by a certain application instance on top of the Lite-Ring system, and provides the KBR service only to that application.

9.2 Implementation

In order to evaluate its feasibility in wireless ad-hoc networks, the Lite-Ring system has been implemented in the OMNeT++ simulator [121]. This section describes some details of the implementation.

9.2.1 Network Topology

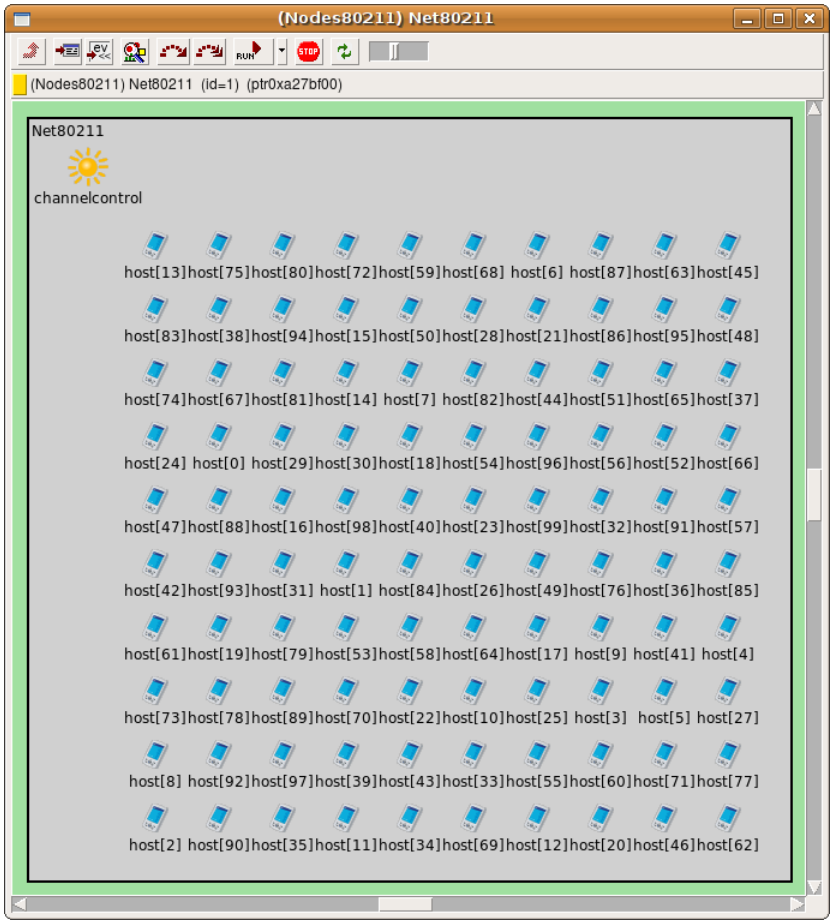


Figure 9.2: Network topology with 100 nodes

Figure 9.2 shows a screen shot of the network topology of the Lite-Ring system in OMNeT++. In this figure, there are one "channel control" module and 100 nodes (compound modules) positioned in a lattice structure. The "channel control" module is taken from the INET framework [120] of OMNeT++. It stores the global information of locations and movements of the nodes, and determines the interference level and the connection probability of any two nodes. More details about the "channel control" module can be found in [120].

In the scenario of Figure 9.2, the distance between neighbor nodes is configured to 50 meter and the transmission range of each node is 55 meter. According

to this settlement, except the nodes on the edges and at the corners, each node has 4 neighbors.

9.2.2 Node Structure

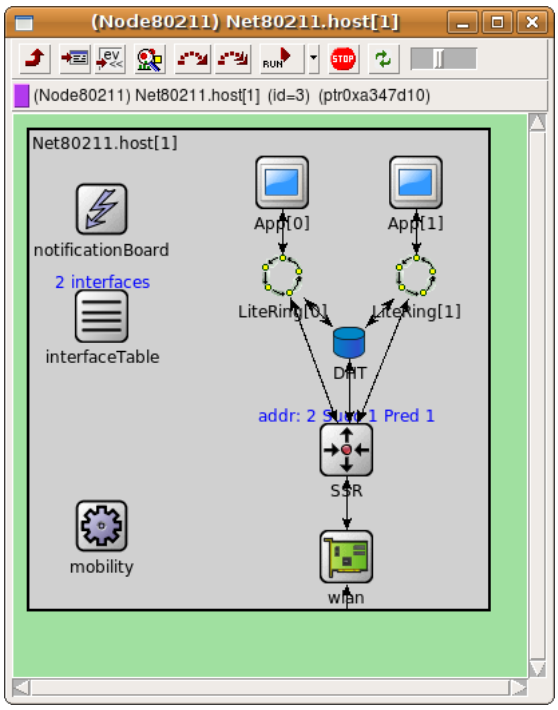


Figure 9.3: Node structure in the Lite-Ring system

Figure 9.3 shows the node structure in the network. It consists of several simple modules and one compound module.

The "notificationBoard", "interfaceTable", "mobility" and "wlan" are modules derived from the OMNeT++ INET framework. The "wlan" module is the only compound module in this structure, see Figure 9.4.

- The "notificationBoard" module is used as a public blackboard for modules to share information with each other.
- The "interfaceTable" module is used to register every layer-2 modules, e. g. wlan module, PPP module, loop-back interface, etc.
- The "mobility" module implements a variant of random direction mobility models, proposed by Perkins and Wang [89].

The "DHT", "SSR", "Lite-Ring" and "Application" modules are the newly implemented simple modules. In the node structure, there is only one SSR module and one DHT module. The Lite-Ring and Application modules co-exist on each node. The numbers of the Lite-Ring or Application modules on different nodes may vary. In the example of Figure 9.3, there are two Lite-Ring modules and two Application modules.

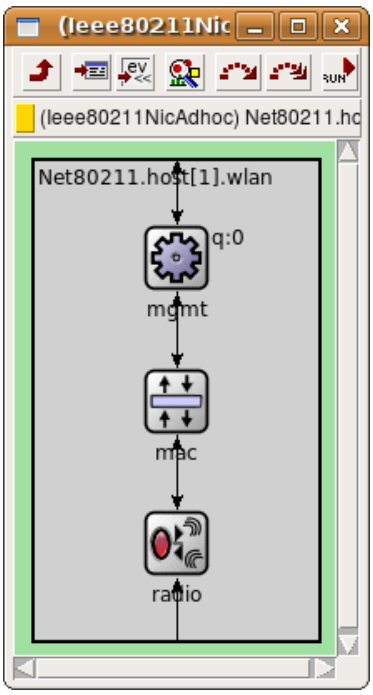


Figure 9.4: Sub-modules inside the wlan module

- The DHT module provides a primitive DHT service with `put`, `get` and `remove` operations. (See Chapter 9.2.3 for the details of the DHT module.)
- The SSR module provides two routing modes, i. e. the key based routing for the DHT module and the end-to-end routing for the Lite-Ring module(s). (For the detail of the SSR module we refer to Section 9.2.4.)
- Each Lite-Ring module is associated with an application and provides the KBR service to that application.
- The Application module is used for statistic purposes. It regularly sends a message to a randomly chosen destination. At the same time, it registers the received messages. At the end of the simulation, some statistics concerning, e. g. the loss rate and delay, can be achieved, based on the information collected by the Application module.

9.2.3 The DHT Module

The DHT module is built on top of the SSR module. It has a local cache, in which the key-data pairs are stored. Note that a key can be associated to more than one data entries.

Messages

There are six types of messages in the DHT module: `cDhtPutMsg`, `cDhtGetMsg`, `cDhtRmMsg`, `cDhtPutResultMsg`, `cDhtGetResultMsg` and `cDhtRmResultMsg`.

- `cDhtPutMsg` is used to publish data in the DHT.

  It contains three mandatory fields, i. e. `Key`, `Data` and `TTL`, as well as an optional field, `Issuer`, which specifies the sender of this message.

- `cDhtGetMsg` is used to inquire the data for certain key. It consists of three fields: `Key` and `Number` of required results, as well as `Inquirer`, which is the destination to which the result should be returned.

- `cDhtRmMsg` is used to remove a certain key-data pair. It has two mandatory fields, i. e. `Key` and `Data`, and one optional field, `Issuer`. As there may be multiple values associated with one key, it is necessary to specify the `Data` or the hash of the `Data` in the `cDhtRmMsg`.

- `cDhtPutResultMsg` is generated when `cDhtPutMsg` contains the optional `Issuer` field. It has three fields: `Key`, `Issuer` and `Error`. The `Error` field is used to indicate if the `put` operation was successful or not. In case of an unsuccessful operation, the error type is specified.

- `cDhtGetResultMsg` returns the data associated with a certain key. It has four fields: `Key`, `Inquirer`, `Number` and `Results`. `Number` refers to the number of results while `Results` are the results array.

- `cDhtRmResultMsg` specifies if the remove operation was successful. Analogously to `cDhtPutResultMsg`, it has also three fields: `Key`, `Issuer` and `Error` with similar meanings.

The key space in the DHT module is the same as the address space in the SSR module. In particular, the DHT module uses the SSR address as its own address. In this implementation, it is a 4-byte (unsigned) random integer.

Routine

If a node needs to store data with a certain key, the DHT module generates a `cDhtPutMsg` message and the SSR module routes this message to the destination node using the KBR mode. When the `cDhtPutMsg` message arrives at the destination node, the DHT module on the destination node stores the key-data pair in its local cache for `TTL` seconds as specified in the message.

If required, the DHT module sends a `cDhtPutResultMsg` message back to the `Issuer` of the `cDhtPutMsg` message, using SSR's end-to-end routing mode. If the `Issuer` doesn't receive a reply, the `cDhtPutMsg` message may be retransmitted depending on the applications.

When a DHT module tries to get some data for a certain `key`, a `cDhtGetMsg` message will be sent to the destination node with SSR's KBR mode. At the destination node, the DHT module looks up the `key` in its local cache and sends a `cDhtGetResultMsg` message back to the `inquirer` using SSR's end-to-end routing mode.

In order to remove the data with certain key, the DHT module sends a `cDhtRmMsg` message to the destination node using SSR's KBR mode. Upon receiving the message, the destination node checks its local cache and deletes the matched data entry. Analogously, an optional `cDhtPutResultMsg` message may be sent back to the `Issuer`.

Parameters

The DHT module has the following major parameters:

- `int capacity`: The maximum number of key-data entries, which can be stored in the DHT module.
- `double dhtGetRetryTimes`: The maximum times of the retransmission of a `cDhtGetMsg` message.
- `double dhtGetRetryInterval`: The time interval between two retransmissions of a `cDhtGetMsg` message.

9.2.4 The SSR Module

The SSR module provides the following API for both end-to-end routing and key-base routing:

```
SendPayload(cAddr Dest, int SizeByte, void* pBuf, bool UpCall,
int AppId, bool RouteMode);
```

- `Dest` is the destination of the message, which can be the SSR address of a certain node or a key in the SSR address space.
- `RouteMode` specifies the routing semantic: `true` for end-to-end routing; `false` for key based routing.
- `pBuf` and `SizeByte` specify the pointer and the length of the payload message.
- `AppId` specifies the application type of the message. It is similar to the port concept in the TCP/IP stack. If the destination node doesn't run the application as denoted in the `AppId`, this message will be dropped.
- `UpCall` indicates whether the modules on top of SSR at the intermediate nodes can intercept the message or not.

The *upcall* concept is often used in the recursive overlay networks, in which messages are forwarded through the network without route discovery process. (In contrast to the recursive overlay network, the iterative overlay networks let the sender discover a route to the destination before delivering messages.) Some P2P applications, e. g. overlay multicast [22], require such an upcall feature.

The upcall concept is not only applied in overlay networks but also in some other end-to-end network layer route protocols. For instance, the Resource Reservation Protocol (RSVP)[14] has included the Router Alert Option [71] in the IP header to let routers examine the contents of the IP packet more closely. Some applications in wireless networks with cross-layer design [109, 111] benefit from such intermediate interceptions as well.

The SSR module has the following major configuration parameters:

- `int cacheSize`: The size of the routing cache.
- `double broadcastInterval`: The time interval between two `hello` messages.

- `double neighborTimeoutInterval`: The timeout period of a routing cache entry.
- `double notificationInterval`: The time interval of the consistency check of the virtual ring, i. e. when to notify the predecessor and successor.
- `unsigned int addr`: a unique and randomly assigned address.

9.2.5 The Lite-Ring Module

The Lite-Ring Module is implemented as specified in Chapter 5 and has the following interfaces:

- `Join(char* appName);`
- `Route(cKey key, int size, void* message, int AppId);`

The Lite-Ring module uses the `Join` function to join the overlay of the application "`appName`". After doing this, it is able to send `message`s to any `key` in the address space of the Lite-Ring module that is associated with the application "`appName`". The `AppId` is the hash of the application name.

The major configuration parameters of the Lite-Ring module are:

- `int AppId`: The ID of the application. It is hashed from the application name.
- `double notificationInterval`: The time interval to maintain the overlay structures of Lite-Ring, i. e. the ring and the tree structure (rf. Chapter 5.2).
- `double dhtRefreshInterval`: The time interval to refresh a data entry in DHT (rf. Chapter 5.2).
- `bool useSytheticAddr`: A flag that specifies whether synthetic addresses shall be used or not (rf. Chapter 5.4).
- `char balanceMode`: The balance mechanism that is to be applied (rf. Chapter 5.5).

9.2.6 The Application Module

The application module on top of the Lite-Ring system carries out the statistic works. It sends messages to random keys and records the delivery ratio and packet delays. It has the following parameters:

- `char* name`: The name of the application.
- `int payloadSize`: The length of a payload message.
- `double interval`: The delivery time interval of payload messages.
- `double initTime`: The starting time of message delivery.

Chapter 10

Simulation and Test

This chapter describes the simulations and the evaluation of the performance of the Lite-Ring system in a wireless ad-hoc network with the OMNeT++ simulator.

10.1 Simulation Setting

The simulation scenarios are based on a network with 100 nodes according to a lattice topology (cf. Figure 9.2). The network topology and the node structure are specified in Chapter 9.2.1 and Chapter 9.2.2. The MAC layer module is the IEEE 802.11 module derived from the INET framework. The data rate of the MAC module is configured to 11 Mbps and size of the application packet is set to 200 bytes. In order to increase the reliability, retransmission option is used for the DHT messages in the Lite-Ring system.

There are three phases in the simulation:

- *SSR bootstrapping phase*, i. e. the first 50 seconds of the simulation. Note that, the SSR modules are initialized during the first 2 seconds. The bootstrapping phase is set to 50 seconds in order to ensure that the overlay structure of SSR converges before the second phase starts.

- *Lite-Ring bootstrapping phase*, i. e. the next 50 seconds (from 51st to 100th seconds) of the simulation. The Lite-Ring modules are initialized at a randomly chosen point in time within this period.

- *Evaluation phase*, which extends from the 101st second to the simulation end. In this phase, the application module sends packets to random keys (inside the Lite-Ring address space).

10.2 Scenario with a Single Application

The first scenario is a static one with only one application that is deployed on every node in the network. Every 20 seconds, the application sends a message to a randomly chosen destination.

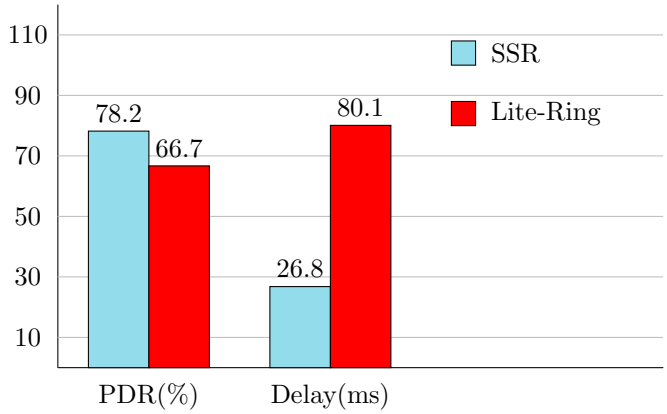


Figure 10.1: Network performance with one application

Figure 10.1 depicts the Packet Delivery Ratio (PDR) and the median packet delay of Lite-Ring and SSR. The PDR of SSR is 78.2%, while the PDR of Lite-Ring is 66.7%. Since there is no packet retransmission mechanism in Lite-Ring, the PDR of Lite-Ring has to be lower than that of SSR. Recall that the message delivery procedure in Lite-Ring comprises two steps, *address lookup* and *message delivery*. The *address lookup* can be done via b DHT `get` operations on average. (Theoretically, if synthetic nodes are applied $b = 1$, otherwise $b = 1.17$. rf. Chapter 5.4.) Given the PDR of SSR, the PDR of Lite-Ring message can be estimated according to:

$$PDR_{Lite-Ring} = PDR_{SSR} \times {P_{DHT}}^{b} \tag{10.1}$$

where P_{DHT} is the success ratio of a lookup operation in the DHT.

Since one `get` operation consists of two DHT messages, a `cDhtGetMsg` and a `cDhtGetResultMsg`, we get $P_{DHT} = PDR_{SSR} \times PDR_{SSR}$ if no retransmission is applied to the DHT messages. Insert it into Function 10.1, we obtain

$$PDR_{Lite-Ring} = PDR_{SSR} \times (PDR_{SSR} \times PDR_{SSR})^{b} = PDR_{SSR}^{2b+1} \tag{10.2}$$

In the current simulation, the PDR_{SSR} is 78.2% and no synthetic node is used. Thus according to Equation 10.2, the $PDR_{Lite-Ring}$ is $0.782^{2\times1.17+1} = 0.44$. This PDR is for practical use too low. In order to raise the PDR of Lite-Ring, the DHT module enables its retransmission option for `get` operations: If there is no `reply` of a DHT `get` message within one second, the `get` message will be retransmitted. The retransmission happens at most two times. With the retransmission, the probability of a failed `get` operation is: $1 - PDR_{SSR}^2$. The success ratio of the address lookup is

$$P_{DHT} = 1 - (1 - PDR_{SSR}^2)^3$$

With PDR_{SSR} equals 78.2%, we get

$$P_{DHT} = 1 - (1 - 0.782^2)^3 = 0.941$$

Inserting the P_{DHT} into Function 10.1, we have

$$PDR_{Lite-Ring} = 0.782 \cdot 0.941^{1.17} = 72.8\%$$

This result is close to the simulation result of 66.7% (see Figure 10.1).

The median packet delay of SSR is 26.8 ms, while the median packet delay of Lite-Ring is 80.1 ms, which is approximately three times of the former. This increased packet delay of Lite-Ring is caused by the address lookup procedure: If there is no packet retransmission,

$$Delay_{Lite-Ring} = Delay_{SSR} + b \cdot Delay_{DHT}. \tag{10.3}$$

Since the DHT `get` operation needs two messages, a `cDhtGetMsg` and a `cDhtGetResultMsg`, we get

$$Delay_{Lite-Ring} = (2b + 1) \cdot Delay_{SSR}$$

With $b = 1.17$, we obtain $Delay_{Lite-Ring} = 89.5$. The result is also close to the median packet delay in the simulation, which is 80.1 ms in Figure 10.1.

Note that instead of mean delay, median delay is used in the derivation in order to reduce the impact of skewed delay distribution caused by the retransmissions, as shown in Figure 10.2.

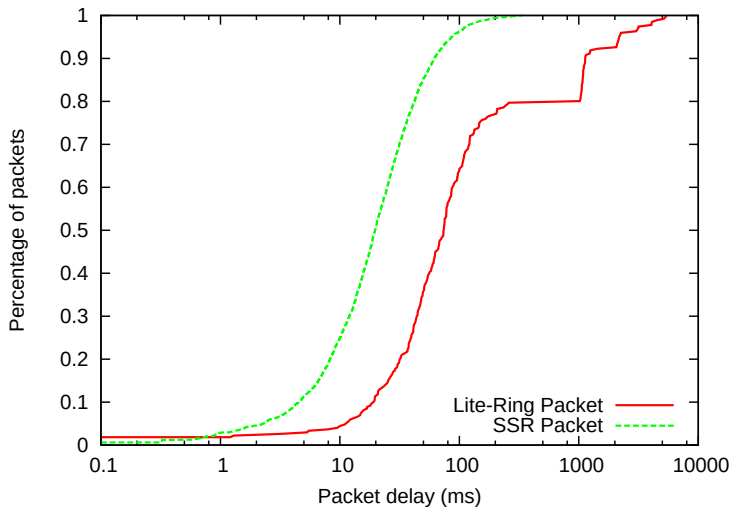


Figure 10.2: Cumulative distribution of packet delay with one application

Figure 10.2 depicts the delay distribution of the Lite-Ring packets and the SSR packets. Here, we again see that the delay of the Lite-Ring packets is larger than that of the SSR packets. Moreover, there are several "jumps" in the Lite-Ring packet delay, e. g. between 250 millisecond and 1 second. These are caused by the retransmission of the DHT `get` operations. (The retransmission interval is set to one second.)

The simulation results of this scenario show that if there is a single KBR application deployed on all the nodes in the network, the performance of Lite-Ring is worse than that of SSR, especially in terms of packet delay. However, the advantage of Lite-Ring is highlighted when there are multiple KBR applications deployed on different nodes.

10.3 Scenario with Multiple Applications

In this simulation, there are two applications deployed in the network. In particular, application A_1 is deployed on 70% of the nodes and application A_2 is deployed on 20% of the nodes. For both applications, the nodes are chosen randomly and independently. Each application instance sends packets to randomly chosen destinations (within the same application). The packet delivery rate is again one packet every 20 seconds.

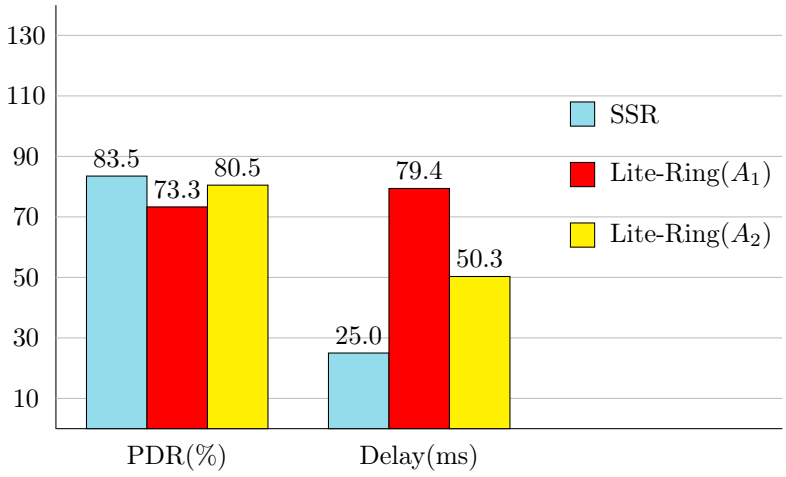


Figure 10.3: Network performance with two applications

In Figure 10.3, the PDR of application A_2 is slightly greater than that of application A_1. This is due to the fact that there are fewer application instances of A_2, so that the chance that the packet destination lies in the local cache of A_2 is higher than that of A_1.

As shown in Figure 10.3, the packet delay of application A_1 is approximately 3 times as large as that of SSR, whereas the packet delay of application A_2 is about 2 times as large as that of SSR. Similarly, the smaller packet delay of application A_2 is due to its small overlay size.

Although under certain conditions, the performance of Lite-Ring is worse than that of SSR, we note that the Lite-Ring system provides the KBR service for multiple applications simultaneously. More importantly, the Lite-Ring system doesn't need to build full-fledged structured overlays in wireless ad-hoc networks, which would lead to enormous overhead and have been demonstrated as infeasible in [31, 18].

10.4 Scenario with an Unstable DHT

Since the DHT modules provide a public service, it might be abused by some malicious or not well designed applications, e. g. limiting the keys just on small address space. The simulation in this section checks the impact of an unstable DHT on the Lite-Ring system.

In this simulation, 50 of the 100 nodes are deployed with a single KBR application, i. e. 50 nodes have the Lite-Ring modules. 0∼40% of the 100 DHT

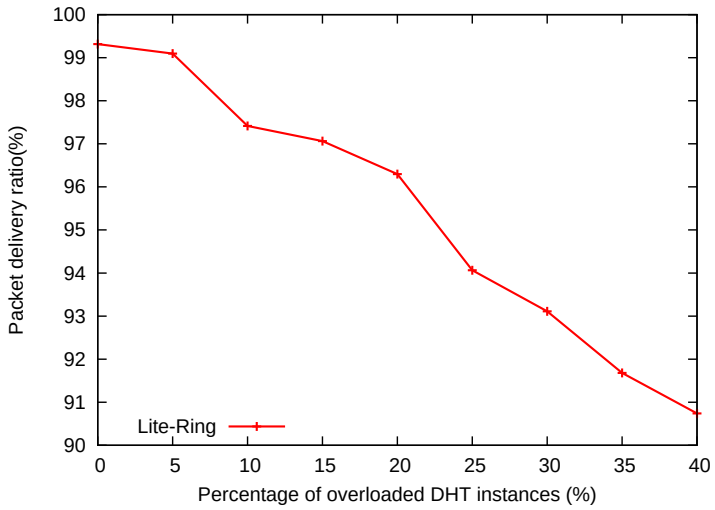


Figure 10.4: Packet delivery ratio in case of an overloaded DHT

modules are configured as overloaded, i. e. the overloaded DHT modules set their capacity to 0 and ignore the received `put` messages. In order to exclude the influence of the packet loss, a simplified mac module which connects the neighborhood nodes with reliable links is used instead of the IEEE 802.11 module.

As shown in Figure 10.4, the PDR decreases as the percentage of the overloaded DHT instances increases. Note that, the PDR remains over 90% even when 40% of the DHT instances are overloaded. This confirms the conclusion in Chapter 6.2.4, that Lite-Ring is robust and even works with a quite unstable DHT.

Figure 10.5 depicts the packet delay, in the case of an unstable DHT service. We observe that the delay increases when the percentage of the unstable DHT instances increases. This can be explained by Figure 10.6. As more and more DHT modules become overloaded, the routing overhead, i. e. the number of the DHT operations needed by the lookup process and the number of Lite-Ring overlay hops, increases.

10.5 Scenario with Increasing Load

In this scenario, 50 nodes are deployed with a single application and the application packet rate increases from 1 packet per minute to 20 packets per minute. As shown in Figure 10.7, the packet delivery ratio keeps above 80% when the application packet rate is lower than 8 packets per minute (approx. 1 KB/second). It drops fast when the message rate exceeds the threshold (8 packets per minute).

The low average node throughput (approx. 0.5 KB/second) is caused by the limited capacity of the wireless ad-hoc networks [56, 79]. As was observed and analyzed in [79], the random traffic pattern is responsible for the low network capacity. This problem will be further discussed in the next chapter in some detail.

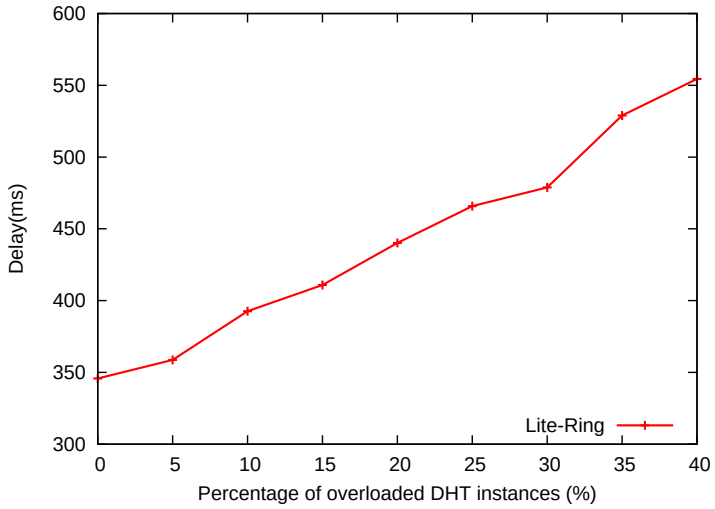


Figure 10.5: Packet delay in case of an overloaded DHT

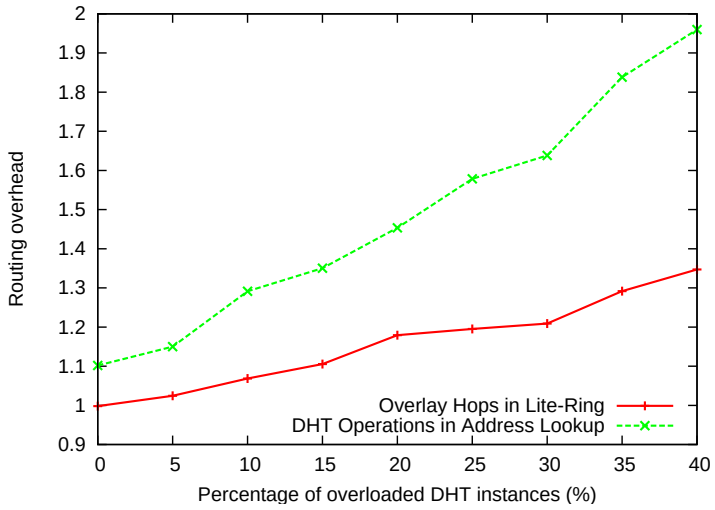


Figure 10.6: Routing overhead in case of an overloaded DHT

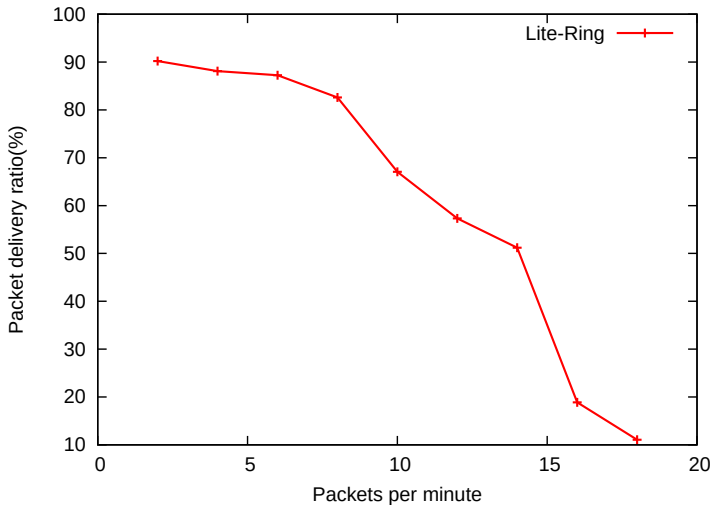


Figure 10.7: PDR with increasing payload

Chapter 11

Conclusions

11.1 Summary

In this part of the dissertation, I first introduced a short case study to demonstrate the applicability of a structured overlay in wireless ad-hoc networks. Then I described the characteristics of P2P networks and wireless ad-hoc networks, as well as the challenges of constructing a P2P overlay in a wireless ad-hoc network.

The Lite-Ring system builds multiple KBR overlays for different applications in wireless ad-hoc networks. Some initial simulations, performed in the OMNeT++ simulator, demonstrated the feasibility of the Lite-Ring system. To my best knowledge, this is the first work aiming at building multiple KBR overlays in wireless ad-hoc networks.

11.2 Discussion

There are a few open issues that need to be considered for the future development of the Lite-Ring system in wireless ad-hoc networks.

11.2.1 Reliable Transmission

As mentioned in Chapter 8.4, reliable delivery is still an open issue of many applications in wireless ad-hoc networks. One straightforward method is to add TCP to the Lite-Ring system. However, as observed in many research works, the original TCP protocol is inappropriate in a wireless environment. Since TCP is unable to distinguish between packet loss caused by congestion and non-congestion-related losses.

A few TCP variants were proposed to overcome the problem of TCP in wireless networks. Some of them, such as [87], try to detect non-congestion-related packet loss with network support, others, such as [127, 16, 49], use end-to-end approaches to accurately probe the available bandwidth. Sundaresan et al. [117] pointed out that TCP and its minor modified versions are fundamentally not appropriate for wireless ad-hoc networks. They proposed a new reliable transport layer protocol, the Ad-hoc Transport Protocol (ATP), for wireless ad-hoc networks. However, none of these new transport layer protocols have been widely

tested or implemented in the network simulators. Since it is very costly to implement a new reliable transport layer protocols inside the Lite-Ring system, the effort of doing this is beyond the scope of this thesis.

As a compromise, the current version of the Lite-Ring system includes some reliability checks in the DHT module, e. g. they re-do the DHT operations in case of timeout. This solution increases the packet delivery ratio, however it causes larger packet delay and jitter.

11.2.2 Capacity of Wireless Ad-hoc Networks

As shown in Chapter 10.5, the capacity of wireless ad-hoc networks is much lower as naively expected. For random traffic pattern, the end-to-end throughput that is available to each node is only $O(\frac{1}{\sqrt{N}})$ [56], where N is the total number of the nodes in the wireless ad-hoc network. Li et al. [79] have found that the average per-node throughput p in an ad-hoc network with 802.11 MAC agrees with the asymptotic bound with a very small coefficient, i. e. $p = 0.047\frac{1}{\sqrt{N}}$.

Li et al. [79] also pointed out that the per-node capacity scales only when the traffic pattern is exactly local. In the current design, each Lite-Ring module has to maintain a constant number of overlay links. However, these links are randomly scattered across the whole network, a fact that reduces the network scalability.

One possible way to improve Lite-Ring's scalability is to keep the maintenance traffic local. If a new coming node could find a proximal Lite-Ring instance and join the network via that instance, the maintenance of ring structure, i. e. the periodical notification to the predecessor and successor, would cause only local traffic. This is because that the proximal bootstrapping node would be the predecessor of the new coming node (rf. Chapter 5.1). This also implies that the proximal nodes would often have close addresses. Using a proximal node to bootstrap also reduces the maintenance overhead of the tree structure, because the addresses of the most parent nodes are close to the addresses of their child nodes (rf. Figure 5.2).

11.2.3 Performance Comparison with ReDiR

ReDiR [100] is also an approach of providing KBR services to multiple applications. Chapter 6.3 and Chapter 5.8 showed that Lite-Ring outperforms ReDiR with respect to route stretch and robustness under an unstable DHT service.

As analyzed in Chapter 2.6.1, a ReDiR node needs 4 `get` and 3 `put` operations per maintenance interval to refresh its entries in the DHT, while a Lite-Ring node only needs 1 `put` operation to refresh its DHT entry and also needs to maintain 6 overlay links for its ring and tree structures (rf. Chapter 5.2).

Using a conventional DHT, both the `get` and `put` operations generate approximately $\log N$ overhead in the underlay, where N represents the number of DHT nodes and overhead means the cost of an end-to-end routing. Hence the maintenance overhead of ReDiR in the underlay is $7 \log N$, whereas that of Lite-Ring is only $\log N + 6$.

According to the PRS/PNS optimization, the $\log N$ overhead factor can be reduced to approximately 1.3 [54]. Event though, Lite-Ring still generates less maintenance traffic than ReDiR does in the underlay.

11.2.4 Layered Solution vs. Cross-layer Solution

The Lite-Ring system contains several independent modules in different layers. In such a layered architecture, the modules in the upper layers are not aware of the underlay, which could limit the network performance occasionally.

Many cross-layer solutions have been proposed to improve the overall wireless network performance by utilizing the underlay information. Such approaches are applied in different layers for various purposes: Overlay protocols, such as CrossROAD [39], utilize the maintenance messages of the network layer to build their overlay structures; Routing protocols, like ExOR [12], explore the medium properties to improve the packet delivery ratio. As the trade-off of these cross-layer solutions, stack-wide layer inter-dependencies are introduced.

In the next part of this dissertation, I will introduce several cross-layer designs for the Lite-Ring system, which will significantly improve Lite-Ring's performance in wireless ad-hoc networks.

Part IV

Cross-Layer Designs for KBR

Chapter 12

Overview

In the previous part of the dissertation, I have proposed the Lite-Ring system which provides the KBR service to multiple applications in wireless ad-hoc networks. Some initial simulations also have demonstrated the feasibility of the Lite-Ring system.

As shown in Figure 9.1, the Lite-Ring system consists of one or more Lite-Ring modules, a DHT module and an SSR module. In the original design, these modules are independent from each other and can be re-used by other modules in the simulator. So far, the Lite-Ring system is constructed straightforwardly according to a layered architecture. However, since each module is independent from each other, it can not be benefited from the useful features of other modules. Hence, the performance of the Lite-Ring system is suboptimal.

12.1 Goal

In this part of the dissertation, some cross-layer optimizations are applied to the Lite-Ring system. In particular, these optimizations intend to improve the network performance in three directions: reducing Lite-Ring's maintenance overhead, accelerating DHT's lookup and optimizing SSR's route.

12.1.1 PNS in the Lite-Ring System

In the original design of the Lite-Ring system, since the Lite-Ring module has no knowledge of the underlay, its maintenance traffic is in a random pattern in the network layer. One possible solution of reducing the maintenance traffic is to apply Proximal Neighbor Selection (PNS), i. e. using proximal nodes to build the overlay structures. In Chapter 2.5, some proximity methods related to PNS are proposed. However, they are all designed for P2P applications in the Internet, thus they can not directly be used in wireless ad-hoc networks, either due to the lack of certain services, e. g. reliable landmarks, or due to the extensive probing overhead.

Recall that in the Internet, the underlay topology information has been already (partially) stored on the routers. However, in most cases the overlay nodes are not the routers but the end-hosts. Hence, lots of efforts have been paid to let the overlay nodes (end-hosts) estimate the topology and to adapt

the overlay structure to the estimated underlay topology. (see Chapter 2.5 for more details.)

In contrast to the Internet, a node in an ad-hoc network is both a router and an end-host at the same time. Thus, an overlay nodes is able to get some information about the network topology directly from the network layer routing protocol. Based on this observation, in the Lite-Ring system, by scanning the routing cache of the SSR module, the Lite-Ring module should be able to find nearby overlay nodes and build proximity-aware overlays.

12.1.2 DHT Caching

Although the Lite-Ring system can run with a primitive DHT service, obviously the performance of the Lite-Ring system can be further improved with a more reliable and efficient DHT.

In most structured overlay protocols, there exists implicitly an aggregation tree for each key. The tree root is the responsible node for the key. Messages from two different nodes to the same key are often supposed to reach the same node before reaching the tree root. Based on this property, DHT can improve its lookup efficiency by caching the data on the intermediate nodes along the path towards the tree root.

In the conventional structured overlays in the Internet, such as Chord, this DHT caching approach generates $O(\log N)$ duplications for each data. In ad-hoc networks[1], since the distance between two nodes is usually $O(\sqrt{N})$ hops, this caching approach could generate $O(\sqrt{N})$ duplications for every data.

Note that such caching is optional for the intermediate nodes, i. e. if nodes run out of memory, the caching function on these nodes can be turned off. More importantly, this caching approach doesn't increase the network traffic. Thus, by adding the intermediate caching approach to the DHT module, the performance of the Lite-Ring system can be improved.

12.1.3 Link-layer Broadcast in SSR

Generally speaking, if there is only one KBR application on each node in a wireless ad-hoc network, SSR is able to provide its KBR service very efficiently (cf. Chapter 10). If there are one or more KBR applications deployed on a part of the nodes in the wireless ad-hoc network, the Lite-Ring system is able to provide the KBR service while SSR fails or at least inefficiently (cf. Chapter 4.2).

Note that in wireless ad-hoc networks, as the connections between nodes are shared-medium, the packets that are delivered can be overheard by sender's neighbors. Moreover, since the SSR module lies on top of the link layer, the SSR performance can be also improved by use of the broadcast nature of the radio channel. Obviously, as SSR is a module of the Lite-Ring system, this may also benefit the overall performance of the Lite-Ring system.

[1] Suppose that the ad-hoc network has a unit-disk graph.

12.2 Background

12.2.1 Layered and Cross-layered Architectures

According to the well-known Open Systems Interconnection (OSI) model, the networking system can be divided into seven different and relatively independent layers. Each layer communicates only with its adjacent layers. The upper layer makes use of the services provided by its immediately underlying layer. Such a layered architecture works well in wired networks, both in the Internet and in an Intranet.

During the last decades, wireless networks have become more and more popular. Wireless communication has actually become an important part of the data networks. Many researchers found that the layered architecture is not suitable for wireless networks. Various cross-layer designs have been proposed to improve the wireless network performance. Some of them aim at solving the problems in wireless networks, that cannot be fixed with the layered architecture. Others focus on exploiting the new feature of the wireless medium, e. g. opportunistic usage of the radio channel [12]. A survey of these proposals can be found in [72]. According to [111], these proposals can be classified into four groups,

- Creation of new interfaces
- Merging adjacent layers
- Design coupling without new interfaces
- Vertical calibration.

12.2.2 Topology Information in the Network Layer

Routing protocols in the network layer are in charge of the end-to-end routing service. By exchanging its topology information with other routers, each router gets the complete or partial network topology information. In a layered architecture, this information is restricted to the network layer and is invisible to other layers. However, this information can be very useful to the higher layers, e. g. to allow PNS in P2P applications.

Cramer et al. [30] demonstrated the performance improvement of P2P applications by using PNS in wireless ad-hoc networks. They also suggested some locality-aware bootstrapping mechanisms [32]. However their approaches are based on a layered architecture and assume that the P2P applications have no access to the topology information stored in the network layer.

As stated in Chapter 3.2.2, in its routing cache, each SSR node stores the source routes to three kinds of nodes: its physical neighbors, its virtual neighbors in the ring structure and the recently contacted nodes. Since there must be intermediate nodes on the source routes to the remote nodes, there is valuable topology information of the proximal area in the routing cache. Therefore, the P2P applications on top of SSR can easily apply PNS by finding proximal nodes in SSR's routing cache.

12.2.3 Caching in DHT

There have been various caching techniques proposed to increased the fault-tolerance of DHT [73, 105, 35, 112]. According to these approaches, the data on a DHT node is duplicated to the successors of this node. This is done either by replication [105, 112] or by erasure-coding [73, 35]. The trade-off between these two techniques lies in the delay of the `get` operation and the duplication overhead [35].

Although these caching approaches make DHTs more reliable, they result in a large amount of packets required for the duplication. In wireless ad-hoc networks, the bandwidth resources are restricted, thus having lots of extra packets for data duplication is undesirable. Instead of the successors of the key's root, the intermediate nodes between the data-originator and the key's root can be used to cache the data, as discussed in Chapter 12.1.2. This caching approach generates several replicas on the intermediate nodes without any additional traffic. Moreover, the `get` message can probably obtain the data before reaching the key's root.

12.2.4 Broadcast in Routing Protocols

Routing in wireless ad-hoc networks has been a challenging research topic for more than a decade. Because these networks operate in a difficult environment. Firstly, the quality of the radio channel is usually unstable. Secondly, nodes move during operation (node mobility). At last, nodes may leave the network ungracefully (node churn). All this results in an unstable and typically unpredictable network topology.

Most ad-hoc routing protocols separate the calculation of the forwarding path from the actual forwarding. Nodes build up and maintain some kind of Forwarding Information Base (FIB). Based on this FIB, they forward a packet to the neighbor whom they consider to be best suited to deliver the packet to its destination. But due to the instability of wireless networks, the information gathered in the FIB might already has become obsolete when it is used for forwarding. Typically, the forwarder detects the resulting routing failure immediately, e. g. by a missing acknowledgment. But recovery from that failure is costly, especially in terms of delay. On the other hand, increasing the amount of proactive checking for the presence of one's neighbors is costly in terms of bandwidth. Furthermore, even if the forwarding action has been successful, the resulting paths may be sub-optimal.

So far, a few routing protocols have been proposed to explicitly exploit the broadcast nature of radio links. For example, in opportunistic routing protocols, an intermediate node and its neighbors jointly determine the next hop from a set of several possible forwarding candidates (rf. Chapter 3.1.4). Thereby, the routing performance can be significantly improved.

Chapter 13

PNS and Caching in Lite-Ring System

In this chapter, I present two extensions in the Lite-Ring system: the Proximal Neighbor Selection (PNS) in the Lite-Ring module and the intermediate caching in the DHT module.

13.1 PNS Design

Gummadi et al. [54] have pointed out that applying PNS in structured overlays can improve the network performance significantly. Cramer et al. [30] have demonstrated further that PNS plays also an important role for structured overlays in wireless ad-hoc networks. In the present work, instead of using complicated proximity methods designed for P2P applications in the Internet, such as Vivaldi [34] and Meridian [126], I propose a simple approach to realize PNS for the Lite-Ring module.

The routing cache in SSR has a tree structure, whose root is the node's own address. In this tree structure, three kinds of source routes are stored: the routes to the node's physical neighbors, those to its virtual neighbors and those to the nodes that have been recently communicated with. An SSR node regularly sends `hello` messages to its physical neighbors. The `hello` message contains the neighbor list of the sender. By comparing the own neighbor list with the lists received from the neighbors, the SSR node can detect unidirectional links and exclude these links from routing.

In this PNS extension, the `hello` message piggybacks the application IDs that run on the node. The neighbor list also contains the application IDs of the neighbors. Upon receiving the `hello` message, the application IDs of the sender and those of the sender's neighbors are stored in the routing cache. With this approach, each node knows the distribution of the application instances in its two-hop neighborhood. Moreover, the SSR node could piggyback its application IDs to the by-passing messages (other than the `hello` messages). In particular, the chance of doing this is inversely proportional to the distance between the intermediate node and the destination node. This allows the SSR node to know more information about the application deployment in its proximity.

During initialization, the Lite-Ring module asks the SSR module for the list of nodes that run the same application in its neighborhood as well as the distance (in hops) to these nodes. If the list is nonempty, the Lite-Ring module sends a bootstrapping message to one of these nodes. Otherwise, the Lite-Ring module bootstraps regularly by querying the DHT, e. g. with the "address picking and probing" method (cf. Chapter 5.3.3).

If more than one node run the same application in the neighborhood, the physical distance could be used as one simple metric to select the bootstrapping node. However, such a metric will generate an unbalanced address tree in the Lite-Ring module. The balancing maneuver might then introduce undesirable overlay links with long physical distance. Hence, besides the physical distances of the bootstrapping nodes, it is also relevant to concern the balance state of the address tree, which is an important factor of the Lite-Ring protocol (cf. Chapter 5.5).

An alternative metric is thus to use the balance state of the address tree. In particular, the SSR module stores not only the IDs of the application instances in its neighborhood, but also the IDs of the Lite-Ring modules as well as the IDs that the Lite-Ring modules will allocate. Upon initialization of the Lite-Ring module, the SSR module returns not only the SSR address of the proximal nodes and its distance to them, but also the Lite-Ring IDs of the proximal nodes and the Lite-Ring IDs that will be allocated by them. Among these proximal nodes, the node, who is not far away (< 3 hops) and will allocate the Lite-Ring ID at the highest level of the address tree, is preferred as the bootstrapping node.

13.2 DHT Caching Design

Caching techniques have been widely applied in DHT applications in the Internet to increase their reliability. As a conventional method, the DHT node stores replica data on its successor nodes. Obviously, such method generates a large amount of traffic. Considering the restricted bandwidth in wireless ad-hoc networks, it is not suitable for the DHT module in the Lite-Ring system to apply this kind of replication.

As stated in Chapter 12.2.3, caching on the intermediate nodes is a specific caching technique, that doesn't generate extra traffic. Based on this observation, this caching mechanism is built into the DHT modules of the Lite-Ring system.

In particular, when a DHT module sends a `put` message to a certain destination node, using the "upcall" feature of the SSR module, the DHT modules on the intermediate nodes inspect this `put` message and cache the stored `key-data` pair. When an intermediate node receives a `get` message with a certain `key`, it can directly reply with the stored `data`, if there is a local entry that matches the `key`.

Caching on intermediate nodes can introduce inconsistencies into the DHT. For example, if there are two nodes that `put` different data under the same key into the DHT, different `key-data` pairs will be stored on the intermediate nodes that lie between the data originators and the root node of the key. Only the `key`'s root is guaranteed to have the complete data set. Therefore, a new DHT `get` interface has to be constructed in the DHT module for retrieving the complete data set: `get-from-root(key)`.

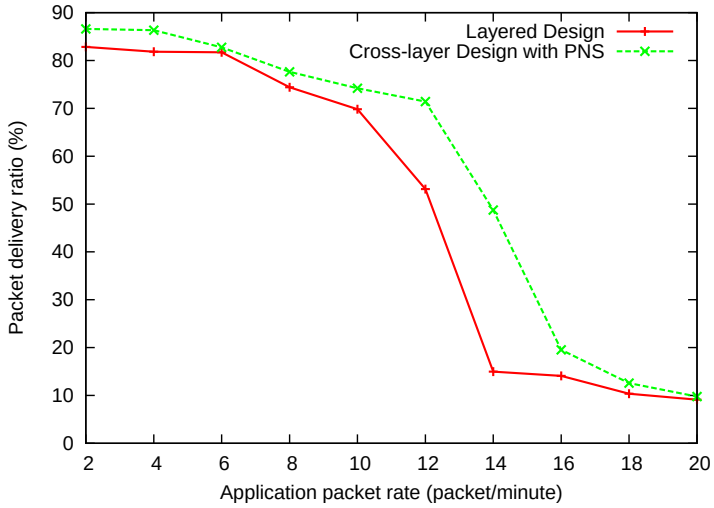


Figure 13.1: Packet delivery ratio of Lite-Ring with and w/o PNS

13.3 Performance Evaluation

In this section, some simulations are performed to demonstrate the benefits of the two cross-layer optimizations in the Lite-Ring system. The simulation settings can be found in Chapter 10.1.

13.3.1 Simulation with PNS

In the first simulation, PNS is enabled in the Lite-Ring system. There are 100 nodes in the network and 50 of them run one unique application.

Figure 13.1 depicts the PDR of the application packets, where we observe that the PDR with PNS is approximately 5% larger than that without PNS until the network becomes saturated[1]. Moreover, we can see that the throughput is increased by 20% (from 10 packet/min to 12 packet/min). Figure 13.3 shows that the application packet delay with PNS is slightly lower than that of the original Lite-Ring system with a layered design.

Figure 13.2 shows the average physical hop count of the Lite-Ring overlay links. After applying PNS, the physical lengths of the overlay links are almost halved. Obviously, the Lite-Ring module with PNS generates less traffic in the underlay, that further results in less interference, thus a higher PDR and a lower packet delay.

As shown in Figure 13.1, when the traffic load exceeds the saturation threshold, the network congests strongly and the PDR of both versions is reduced to a very low level. This is because the application traffic occupies the most part of the network traffic, so that the benefit gained by the reduced maintenance traffic becomes negligible.

Note that even with PNS, the overlay link might also be established between two remote nodes. One reason is that there might be no proximal node running the same application at all. The other possible reason is that the balancing

[1]We define the PDR of 60% as the saturation threshold in this dissertation.

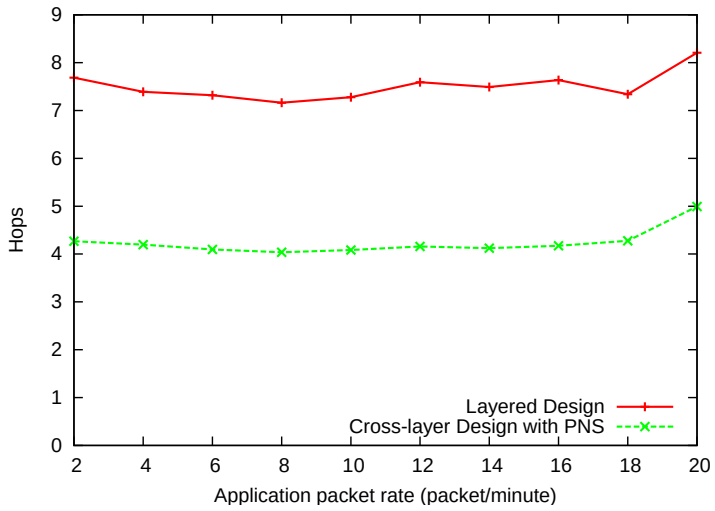


Figure 13.2: Hop count of the overlay links of Lite-Ring with and w/o PNS

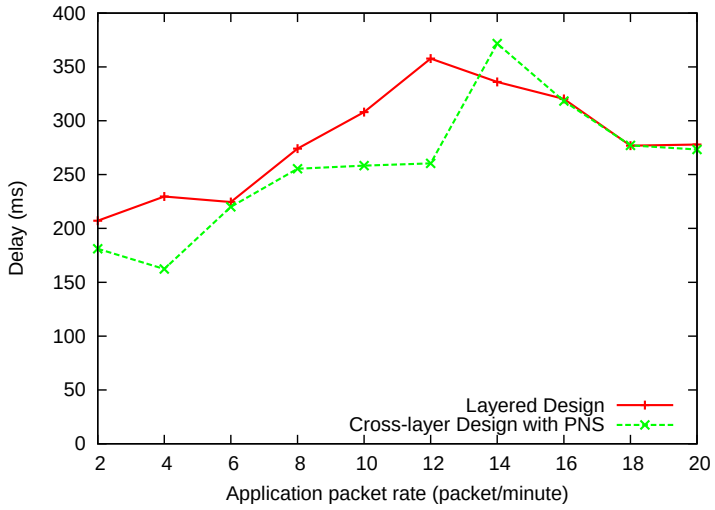


Figure 13.3: Packet delay of Lite-Ring with and w/o PNS

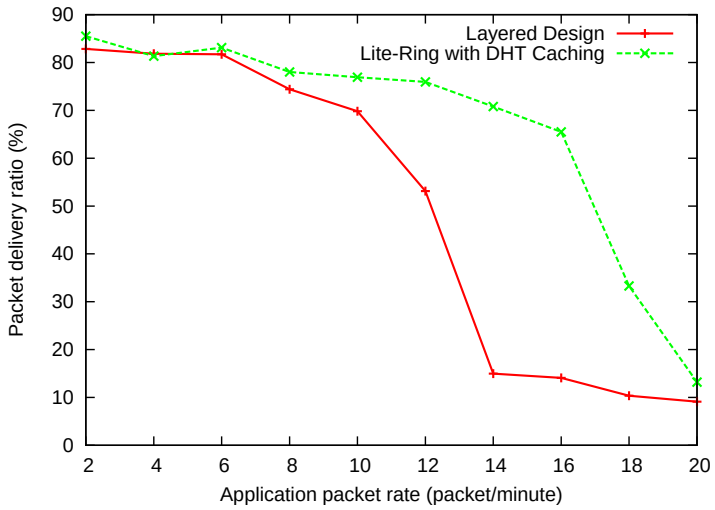


Figure 13.4: Packet delivery ratio of Lite-Ring with and w/o DHT caching

mechanism can replace the proximal links with non-proximal links. Although turning off the balancing mechanism could keep those proximal overlay links. the lookup time would greatly increase, due to the imbalance of the address tree. Thus, this alternative would not be considered further.

13.3.2 Simulation with DHT Caching

In this simulation, the DHT caching mechanism is turned on, i. e. , the DHT modules cache the key-data pair of the by-passing `put` message. Again, there are 100 nodes in the network and 50 of them run the same application.

As shown in Figure 13.4, the throughput of the Lite-Ring system with the DHT caching mechanism has increased by 60% (from 10 packet/min to 16 packet/min). In Figure 13.5, the packet delay of the Lite-Ring system drops dramatically when the intermediate caching is applied.

Owing to the intermediate caching, the inquirer may be able to get the data even before the `get` message reaches the key's root. This raises the efficiency of the lookup process of Lite-Ring. The increase of the PDR and the drop of the packet delay are results of two facts. On the one hand, the reduction of the hop count of the `get` messages and the `result` messages, increases the success ratio of a DHT lookup in an unreliable wireless environment. On the other hand, the reduction of the hop count reduces the network traffic, which further reduces the interference probability on the radio link.

Although no extra network traffic is introduced by the caching mechanism, the amount of data entries stored on the nodes is increased. As illustrated in Figure 13.6, the amount of data entries is increased more than 10 times. This might be inappropriate to the node with very small memory, e. g. a sensor node. However, This caching mechanism is an optional feature and each host can switch it off according to its available capacity.

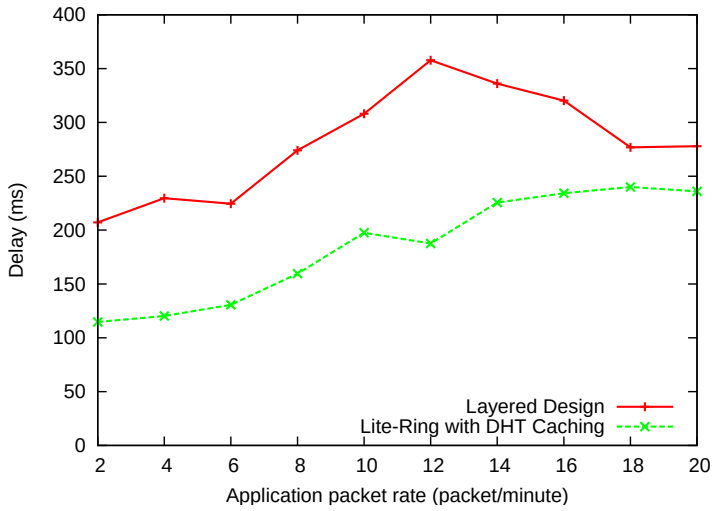


Figure 13.5: Packet delay of Lite-Ring with and w/o DHT caching

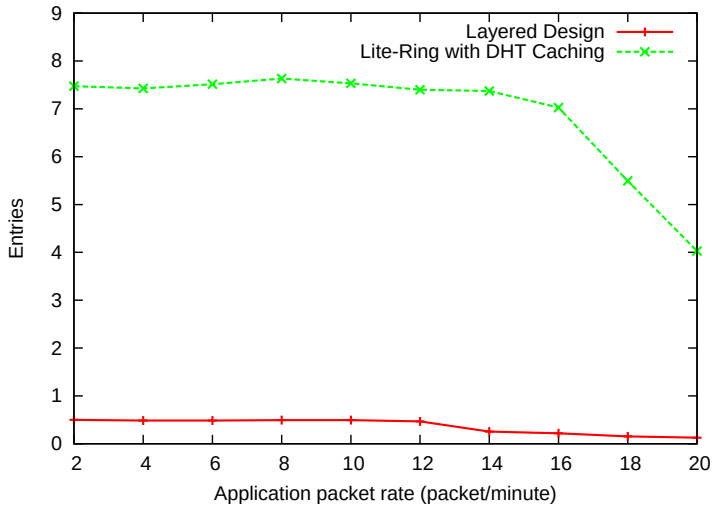


Figure 13.6: Entries stored in the DHT modules with and w/o caching

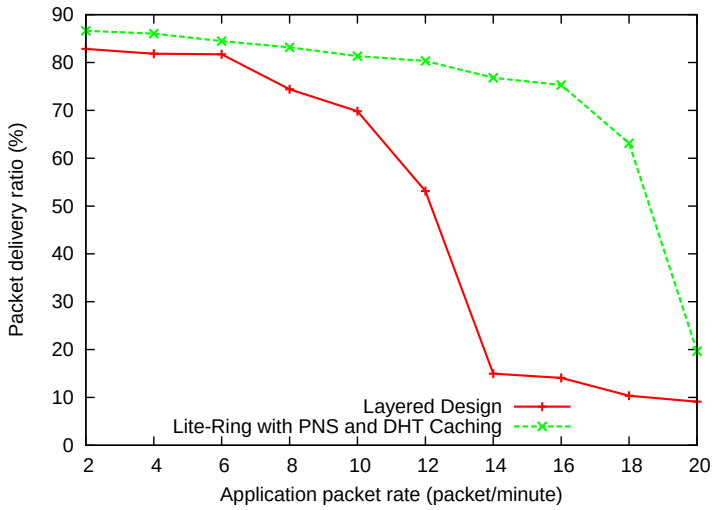


Figure 13.7: Packet delivery ratio of Lite-Ring with and w/o both extensions (PNS and DHT caching)

13.3.3 Simulation with Combined Extensions

In this simulation, we enable the both extensions in the Lite-Ring system. As shown in Figure 13.7, the throughput of the extended Lite-Ring system is increased by about 80% (from 10 packets per minute to 18 packet per minute), whereas the packet delay of the extended version drops by even more than 50%, assuming an intermediate application packet rate, e. g. 12 packets per minute (see Figure 13.8).

In Figure 13.8, we observe that the packet delay of the original Lite-Ring system decreases, when the network is saturated. This is caused by its low packet delivery ratio (see Figure 13.7), and most successfully delivered packets have a small hop count,

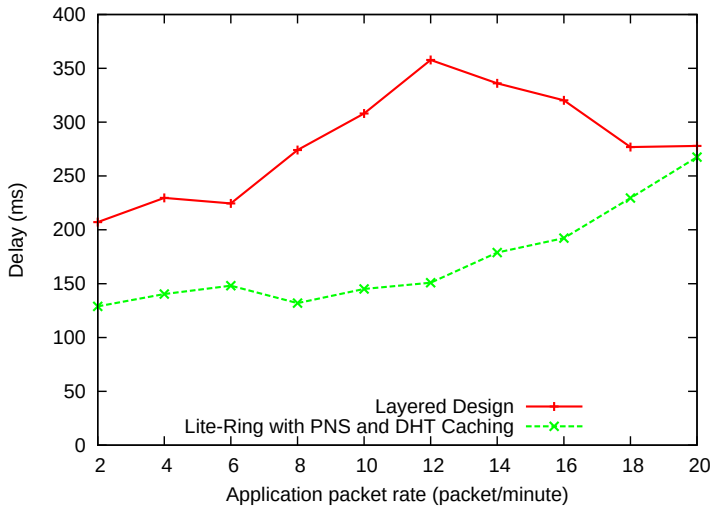


Figure 13.8: Packet delay of Lite-Ring with and w/o both extensions (PNS and DHT caching)

Chapter 14

Optimizing SSR with Link-layer Broadcast

By embedding a structured overlay into the network layer, SSR achieves a good route performance with only small routing state and little maintenance overhead. Since SSR is a routing protocol that is independent of the underlying network type, it can be applied in the networks with different link-layer protocols.

In wireless ad-hoc networks, nodes are connected via a shared medium, which means that packets are broadcast and can be overhead by the neighbors. As it will be shown in this chapter, this characteristic of wireless links can be utilized to improve the performance of routing protocols. In the following, I will discuss an enhancement of SSR that uses the broadcast feature of wireless links. This chapter is extended from an earlier work: "Using Link-Layer Broadcast to Improve Scalable Source Routing"[42].

14.1 Broadcast in Routing Protocols

The enhancement of SSR by link-layer broadcast was inspired by geographic routing and opportunistic routing.

Using link-layer broadcast to improve routing performance was originally proposed in geographic routing. Beacon-Less Routing (BLR) [57] uses broadcast to select the next hop in a distributed manner: The node which is the geographically closest to the destination acknowledges the packet first, and it also keeps the other nodes from unnecessarily forwarding the packet by defining a forwarding area, where every node can hear each other. Consequently, all the nodes along the path jointly determine the forwarding path. Chawla et al. [24] suggested an alternative delay function for BLR. Here, nodes closer to the forwarding node reply earlier. As a result, more nodes are likely to hear the first response, and the suppression of duplicate transmissions is more effective. BOSS [106] is also a variant of BLR. It lets the sender select the next hop and thus suppress the packet duplication entirely.

In opportunistic routing, e. g. ExOR [12], the sender assembles a prioritized forwarding candidate list in the packet header before sending the data packet. Receiving nodes can switch off their receivers when they are not in the candidate

list. Depending on the radio details, this potentially saves energy. The nodes in the candidate list, that have successfully received the data packet, acknowledge the receipt in the sequence according to their priority. The Acknowledgment (ACK) of the node with higher priority suppresses the ACKs of the nodes with lower priorities. The receiver, who did not receive an acknowledgment with a higher priority, forwards the packet.

The idea behind the data-first scheme in ExOR is to avoid the retransmission of the data packet on error-prone links. The more candidates hear the data packet, the higher is the probability that at least one of them will receive the packet correctly. Thus the loss probability is lower than that in the case of only one pre-determined next hop.

14.2 Enhancement of SSR

In this section, we describe an enhancement of SSR, that combines the joint forwarding decision of the aforementioned protocols to SSR. Similar to ExOR [12] and BOSS [106], the sender adds a set of candidates to the data packet. Upon receiving the data packet, the candidates will sequentially acknowledge the receipt of the packet and their routing ability of the packet. The sender usually selects the one who first acknowledges to forward the packet. Thereby, it suppresses the acknowledgments from the other candidates. However, if one of the overhearing nodes detects that the next hop was chosen sub-optimally, it will notify the sender. It can then become the forwarder for that path in future. This allows the protocol to optimize the source route iteratively.

14.2.1 Sending DATA

Before each DATA[1] packet a candidate list is sent using a so-called CTS packet. In fact, the CTS can be piggybacked on the DATA packet such that they share one link-layer broadcast frame (cf. Figure 14.1). The node sequence in the candidate list specifies the acknowledgment sequence of the candidates: When the n^{th} node from the candidate list wants to acknowledge, it will do so only after the candidates 1 to $n-1$ had their chance to do it.

Note that the original SSR proposal suggested a tree structure for the nodes' routing cache [50]. This means that there is only one unique forwarding path for each destination. In this enhancement, the respective next hop node is inserted into the first slot of the candidate list, and this node is called *default candidate*. Later SSR versions [51] equipped the routing cache with auxiliary links. The respective nodes from these auxiliary links can also be inserted into the remaining slots according to their routing ability of the packet. (For a discussion of the ability calculation we refer to Chapter 14.2.4.)

After receiving the DATA packet, each of the candidates calculates its routing ability for this packet. In the best case, the *default candidate* acknowledges the receipt of the DATA packet by immediately sending an ACK. The other nodes in the candidates list will respond to the receipt of the DATA packet after $T = \delta t * n$, where n is the position of the node in the candidate list. Note

[1]Note that – unless stated otherwise – throughout this chapter, the terms DATA, ACK, RTS and CTS are used to name the link-layer broadcast packets of the enhancement protocol. (These terms are in analogy to the well-known IEEE 802.11 frames.)

that applying a discrete delay for ACKs avoids packet collisions. The value of δt depends on the respective values of the MAC protocol. (In this thesis, IEEE 802.11 MAC layer is used.)

14.2.2 Receiving ACK

Each ACK includes the candidate's ID, the packet ID and the acknowledging node's routing ability for that packet. The nodes overhearing the ACKs can determine how to deal with DATA packet depending on the ability value overheard: If the received value is larger than the local value, the cached DATA packet will be dropped and the scheduled ACK will be canceled; otherwise, the ACK will be sent as scheduled and the DATA packet will be further cached.

Upon receiving the ACK from the default candidate, the sender broadcasts a RTS message. This RTS message includes the default candidate's address and its routing ability.

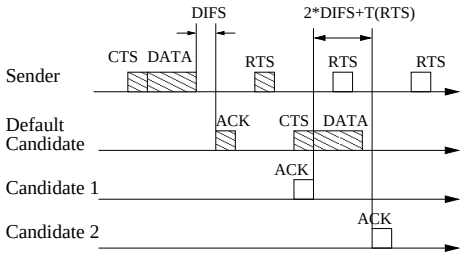


Figure 14.1: Packet delivery sequence: The sender always waits for the first ACK. If the default candidate responds, candidate 1 and 2 will cancel their ACKs upon receiving the RTS; otherwise, candidate 1 and 2 will send their ACKs in the predetermined sequence unless they receive an RTS.

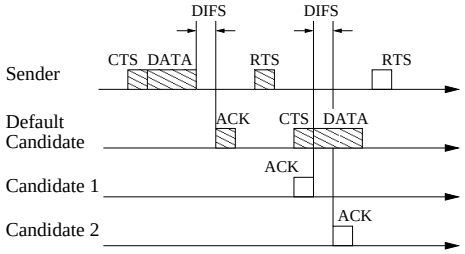


Figure 14.2: Packet delivery sequence: If the default candidate responds, candidate 1 and 2 will cancel their ACKs upon receiving the RTS; otherwise the sender waits for the ACKs of all candidates before it selects the forwarder.

If the first received ACK is not from the *default candidate*, the sender can either immediately trigger the forwarding of the DATA packet by broadcasting a respective RTS, or it can wait for further ACKs before deciding which node shall forward the packet. In the first case, the pause between the ACKs must be sufficiently large in order to wait for a potential RTS (cf. Figure 14.1). In the second case, a smaller pause between the ACKs is allowed (cf. Figure 14.2).

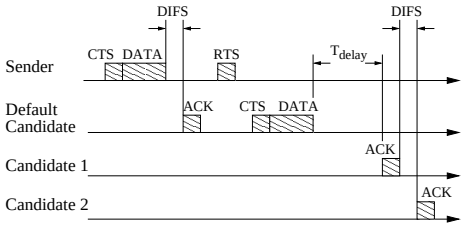


Figure 14.3: Packet delivery sequence: If the default candidate succeeds to respond, other candidates may nevertheless acknowledge to improve future forwarding actions.

Each candidate node caches the DATA packet until it receives the RTS from the sender (or a timeout occurs in case the RTS was lost). The RTS determines which candidate shall forward the DATA packet. In case that none of the candidates sends an ACK, the sender has to retransmit the DATA packet. Thus, the sender has to cache the DATA packet, too.

14.2.3 Route Optimization

Obviously, if the default candidate acknowledges the DATA packet, no further selection for the next hop is necessary. (Remember that only the ACKs from the non-default candidates will lead to an improvement of the forwarding path.) However, if a candidate detects that it would have been better suited to forward the packet than the default candidate, it will send an additional ACK after the forwarder has sent the DATA packet (cf. Figure 14.3). This enables the sender to optimize its candidate list for future DATA packets. The delay T_{delay} of this ACK must be waited so that this ACK does not collide with the ACKs and the RTS of the next forwarding hop.

Nevertheless, if the default candidate is temporarily unavailable, other candidates can acknowledge the DATA packet according to the sequence specified in the candidate list. Thus, in situations with unstable network conditions where it is likely that the default candidate fails to acknowledge the DATA packet, other candidates have their chance to quickly stand in for that packet.

14.2.4 Ability Calculation

The original SSR protocol chooses its next hop based on the virtual distance (D_{vir}) and the physical distance (D_{phy}) from the nodes in local routing cache to the destination (see Chapter 3.2.2). In this enhancement, the nodes in the neighbors' routing cache are also taken into account. Moreover, another parameter, the node's liveliness, is proposed to extend the choice of forwarding candidates. In detail, the *routing ability* Q of a candidate with respect to a given destination is defined as follows:

$$Q = \begin{cases} D_{phy} + T_{nextHop} \\ \qquad \text{if the candidate can provide a complete path} \\ D_{phy} + PENALTY + D_{vir} \qquad \text{otherwise,} \end{cases}$$

where D_{phy} is the physical distance to the destination (or the next intermediate node) measured in hops, D_{vir} is the virtual distance between the next intermediate node and the final destination normalized to the unit interval $[0, 1)$, $T_{nextHop}$ is the liveliness of the next hop measured as the ratio of the time since the respective node last sent a packet and the length of the SSR hello interval (1 second in the implementation), and $PENALTY$ is the penalty value for the incomplete paths ($PENALTY = 50$ in the implementation).

Considering the effect of this formula, we see that a complete path is weighted much more than the path to an intermediate. (Note that a smaller Q means a better routing ability.) The liveliness $T_{nextHop}$ can exclude stale nodes from routing packet and is only taken into consideration when choosing between several complete paths of the same length. If none of the candidates can provide a complete route to the destination, the shorter incomplete path towards the destination will be preferred. To this end, the ability calculation reflects the respective SSR requirements.

The following example illustrates the ability calculation and its effect: Assume node 1 wants to route a packet to node 20. Node 1 can produce the source route 1-4-17 from its routing cache. Thus, node 4 becomes the default candidate.

Node 1 also knows that it has two more physical neighbors besides node 4: node 2 and node 3. Both become further candidates in the candidate list. Node 2 has the source route 2-6-9-20 in its cache. It has last heard of node 6 half a second ago. Thus $Q_2 = 3 + 0.5 = 3.5$. Node 3 has the source route 3-8-19 in its cache. Assuming an address space $0 \ldots 99$ for this example, we get $Q_3 = 2 + 50 + 0.1 = 52.1$. Node 4 has the source route 4-18 in its cache. Thus $Q_4 = 1 + 50 + 0.2 = 51.2$.

As the default candidate, node 4 acknowledges the DATA packet, receives the RTS and forwards the DATA packet. Node 3 cancels its ACK because $Q_4 < Q_3$. At least T_{delay} after node 4 has forwarded the DATA packet, node 2 sends its ACK to node 1 including the route 2-6-9-20. The reason is that $Q_2 < Q_4$. Thus in future, node 1 will choose node 2 as default candidate.

Note that $T_{nextHop}$ is important in scenarios with mobility or churn. It enables the nodes to quickly learn the nodes that can provide a path to the destination. In the original SSR proposal, broken links were only detected after the timeout of a regular `hello` message, which is 3 seconds in the implementation. This is too long for networks with node churn or node mobility. Note that it is possible to detect a broken link by the timeout of a link-layer unicast packet from the 802.11 link-layer, but this would also cause large delay due to the exponential back-off of the retransmission. Moreover some MAC implementations might not support such delivery feedback.

Unlike the original SSR proposal, in this proposal not only the sender and the intermediate nodes determine the routing paths, but potentially all their physical neighbors can contribute. As it will be shown in Chapter 14.4, this allows efficient SSR operation with a significantly smaller routing cache.

14.3 Design Rationales

14.3.1 Subsequent Route Optimization

At first glance, this enhancement might seem to be sub-optimal: The respective sender of a packet determines the sequence of the forwarding candidates. Unless the default candidate fails to acknowledge the packet, there is no freedom left to improve the path. However, as mentioned already, an unsuccessful candidate may still send its ACK at some later time. This additional information improves the path for future packets that are sent to the same destination.

14.3.2 Sender-based Forwarder Selection

In BLR [57], the acknowledgment sequence and the next hop are determined by the candidates locally. This means the best candidate has to find some non-trivial way to suppress the other candidates from forwarding. Moreover, when the candidates have similar routing ability, the collision probability of their acknowledgments will increase.

In ExOR [12], the sender determines the acknowledgment sequence, but the forwarding decision is made in a distributed manner by the candidates. This saves the RTS/CTS overhead, but it can result in DATA packet duplication.

In this extension of SSR, the next hop selection is applied in a centralized manner, i. e. both the acknowledgment sequence and the actual forwarder are determined by the sender. This centralized approach avoids the collision of the acknowledgments as well as the potential duplication of the DATA packet. Furthermore, it does not introduce significant additional packet delay as compared to the standard RTS/CTS handshake in IEEE 802.11.

14.3.3 DATA-first Order

Using DATA-first order can increase the probability of collisions on the link-layer, because a potential forwarder cannot inform the sender that it is busy receiving another packet. However, if there are several potential forwarders and the link is error-prone anyway, the effect of the additional packet errors caused by collisions is negligible. Furthermore, as we have discussed above, the proposed scheme withstands a temporarily unavailable default forwarder, because this allows for more diversity in forwarding packets.

14.3.4 Hidden Terminal Problem

If packet errors were mostly caused by hidden terminals, this proposal could easily be adapted to that situation, too. In that case, the DATA packet would not be sent until the next hop forwarder has sent its CTS. This means, by exchanging the RTS and CTS packets, which include the Network Allocation Vector (NAV), the medium is allocated for the next DATA. In short, the determination of the forwarding path would happen one hop in advance.

It is worth noting that the NAV can only be included in the RTS and CTS packets. The reason is that only after exchanging RTS and CTS packets, the concrete DATA transmission time can be determined.

14.4 Performance Evaluation

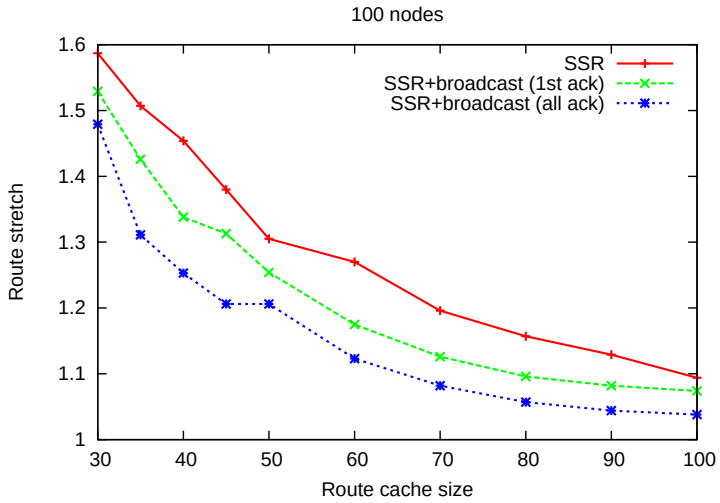


Figure 14.4: Static scenarios (100 nodes)

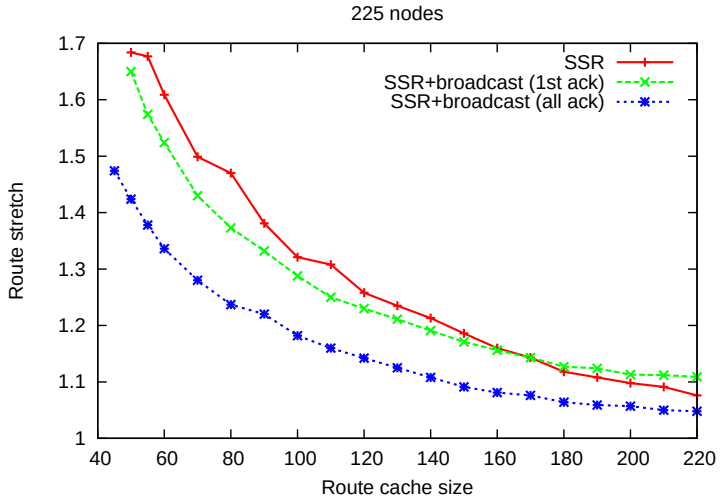


Figure 14.5: Static scenarios (225 nodes)

In this section, the proposal is evaluated by simulations with three kinds of scenarios: static scenarios, scenarios with node mobility and scenarios with node churn. The three scenarios are evaluated with 100 nodes and/or 225 nodes. The static nodes are arranged in a lattice structure with a distance of 50 meters in between. The transmission range of the node is chosen such that each node has eight neighbors, except for the nodes in the corner and on the margin, which have three and five neighbors respectively.

Again, the OMNeT++ simulator [121] is used and the IEEE 802.11 MAC

module is derived from the INET framework. Moreover, except for the radio range, which is set to 75 meters, default parameters of the MAC module are used[2]. The application packet size is set to 120 bytes. Before the application begins to send messages, the system is simulated for 3 minutes. During this time, SSR bootstraps and fills up the nodes' routing caches. After the initialization period, the simulation runs for another 3 minutes.

In order to focus on the performance improvement of SSR when link-layer broadcast is applied, the Lite-Ring module and the DHT module are excluded from the Lite-Ring system. One same application module is directly deployed on top of each SSR module.

14.4.1 Static Scenarios

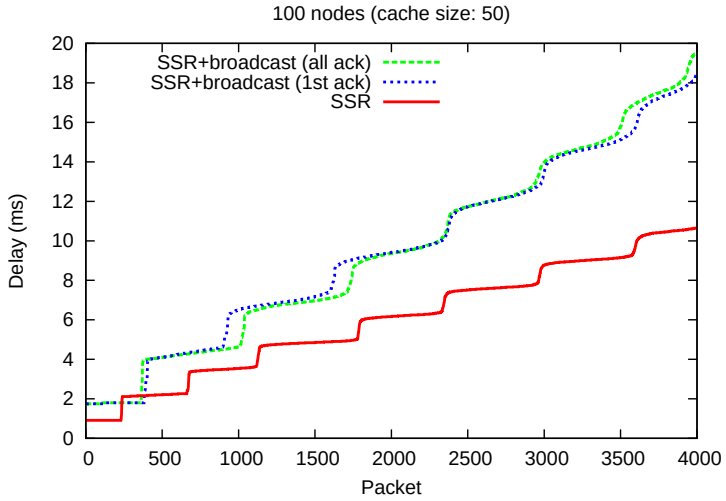


Figure 14.6: Packet delay in static scenarios (sorted packet delay)

In this scenario, the enhanced SSR is compared with the original SSR protocol in two static scenarios with 100 and 225 nodes, respectively. Each node generates 5 consecutive packets to one randomly chosen destination every 30 seconds. The simulation runs several times with a routing cache size ranging from a minimal size of 30 (45) nodes to a maximal size of 100 (225) nodes.

In Figure 14.4, we can see that the enhanced SSR protocol achieves a shorter path length than the original SSR protocol. The improvement is especially significant for small routing cache sizes. Conversely, we find that with this enhancement the routing cache can be reduced by about 30% and still achieve the same route stretch as the original SSR protocol.

However, due to the extra handshaking overhead in the network layer, the delay increased slightly despite of reduced path lengths (see Figure 14.6). Although the handshaking packets are named as DATA/ACK/RTS/CTS packets, in analogy to the respective link-layer frames, these packets are broadcast in the IEEE 802.11 link-layer (with DATA frame). Thus these packets create more overhead than the link-layer handshaking in the original SSR protocol.

[2]These parameters are derived from the ad-hoc example scenario of the INET framework.

Delivered bits	Original SSR	SSR+Broadcast(1st ack)	SSR+Broadcast(all ack)
Application layer	557760 (1.0)	579840 (1.0)	590400 (1.0)
Network layer	9942584 (17.8)	9179144 (15.83)	9593704 (16.25)
Physical layer	36880408 (66.1)	39526768 (68.1)	40544024 (68.6)

Table 14.1: Traffic in the network stack (static scenario with 100 nodes, routing cache size 50, 1 minute)

Table 14.1 shows the entire traffic that is generated at the different layers. Herein, we observe that the enhanced SSR generates less overhead in the network layer than the original SSR protocol. This reduction is caused by the reduced path length. However, due to the fact that broadcast packets are used for the handshaking, the overhead in physical layer increases in the enhanced SSR version. This is because that the DATA frame is the basic frame used for each handshaking packets in the enhanced SSR. For example, a link-layer ACK frame according to IEEE 802.11 consists of 14 bytes in the OMNeT++ simulator, whereas, the minimum length of a DATA frame contains 34 bytes.

The increase of the packet delay and the traffic overhead at the physical layer can be ascribed to using link-layer (IEEE 802.11) broadcasts to realize handshaking. Apparently, if another MAC protocol is used or the handshaking part of the enhanced SSR is moved onto the IEEE 802.11 MAC-layer, these effects would vanish.

14.4.2 Scenario with Mobility

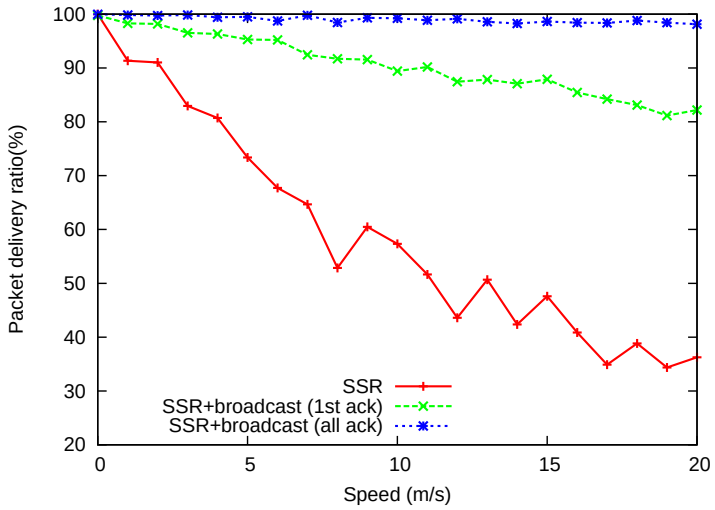


Figure 14.7: Scenarios with mobility

In this scenario, one node moves randomly (according to the random way point model) in the network. Some randomly selected static nodes send messages to this moving node. In order to eliminate the potential influence of insufficient routing cache, the routing cache size is set large enough to store all

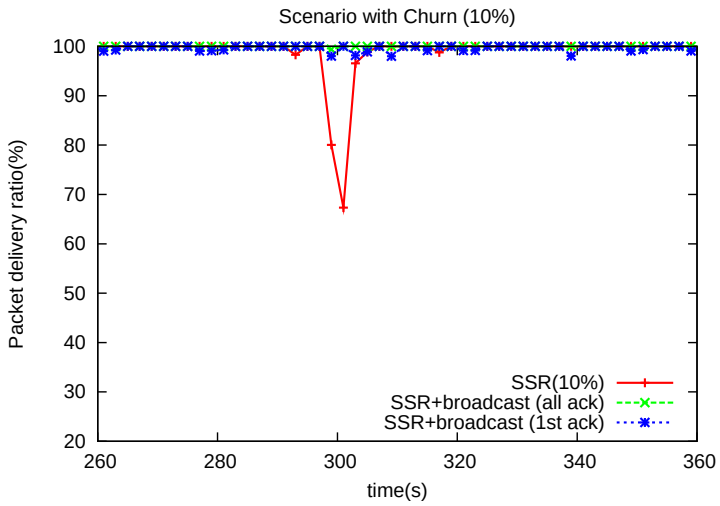


Figure 14.8: Packet delivery ratio in churn scenarios (churn rate 10%)

the nodes in the network.

In Figure 14.7, we observe that the packet delivery ratio of the original SSR decreases rapidly with the increase of node speed. This is due to the fact that the plain SSR cannot distinguish a broken link from a link with packet loss. Thus, the plain SSR might keep on sending packets to a broken link until the breakage is detected via a timeout of a `hello` message. Obviously, this would cause a large packet loss.

According to the enhanced SSR, a broken link can be quickly replaced by another available link. As shown in the simulation, both variants of the enhanced SSR achieve a delivery ratio of more than 80% even when the node moves at a speed of 20m/s. In particular, the variant of waiting for all the ACKs before determining the next hop achieves an even higher delivery ratio of 97% at that node speed.

14.4.3 Scenario with Churn

In this scenario, 10 (and 50) of the 100 nodes are inactive from the 300th second to the 330th second. Note that the inactive nodes are chosen in such a way that the remaining network of the active nodes is still connected and the traffic during this period is only generated between the active nodes.

In Figure 14.8, we observe that with respect to a moderate churn rate of 10%, the packet delivery ratio of the original SSR drops dramatically to 65%, while the packet delivery ratio of the enhanced SSR is almost uninfluenced. The performance drop of the original SSR can be ascribed to the deterministic forwarding mechanism based on the potentially outdated neighbor information. In contrast, in the enhanced SSR, the opportunistic forwarding mechanism using timely neighbor information is applied.

In Figure 14.9 we observe that the PDRs of both the original SSR and the enhanced SSR are affected during the churn period. The PDR of the original SSR drops to 20% and then slowly recovers to 100%. However, the PDR of the

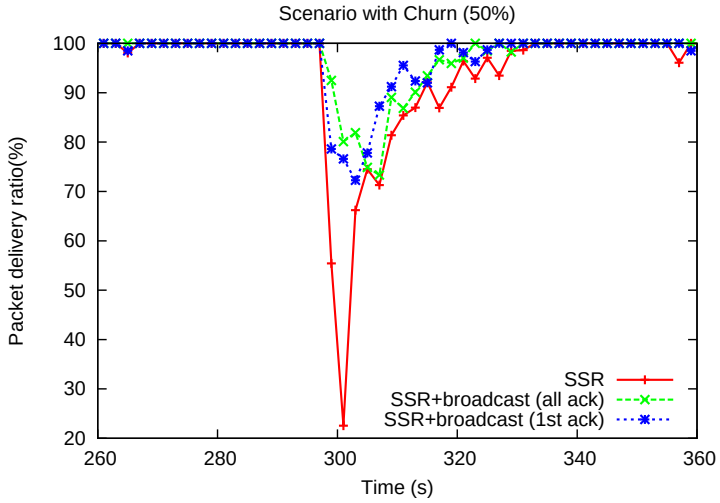


Figure 14.9: Packet delivery ratio in churn scenarios (churn rate 50%)

two variants of the SSR enhancement stays above 70%.

Figure 14.10 shows the path lengths of the original SSR and the enhanced SSR with respect to an extreme churn rate of 50%. We observe that at the beginning of the churn period the path length of the original SSR drops severely while both variants of the enhanced SSR protocol are only slightly affected. This can be explained by the fact that when churn begins, most of the present multi-hop paths will break. Thus, the original SSR protocol can only route packets to the destinations that happen to be close to the source node, which implies very short path lengths. As for the enhanced SSR, timely neighbor information can be used to repair some of the broken paths, so that the overall path lengths are barely affected.

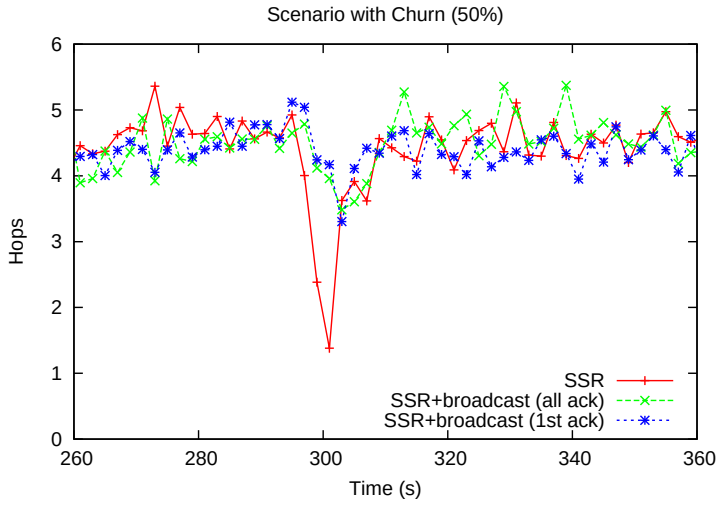


Figure 14.10: Packet hops in churn scenarios (churn rate 50%)

Chapter 15

Conclusions

15.1 Summary

In this part of the dissertation, I have presented three cross-layer designs to improve the performance of the Lite-Ring system. They are Proximal Neighbor Selection (PNS) in the Lite-Ring module, intermediate caching in the DHT module, and link-layer broadcast in the SSR module.

- PNS in the Lite-Ring module is done with the interaction between Lite-Ring and SSR. Each node in ad-hoc networks is both an end-host and a router. Thus, each node that runs the SSR protocol has partial topology information. Lite-Ring can utilize SSR's topology information to perform PNS.
- Intermediate caching in the DHT module is achieved by the interaction between DHT and SSR. With the support of the upcall feature of SSR, the DHT module can cache the data of the by-passing `put` messages. This significantly reduces the lookup time and the network traffic.
- Link-layer broadcast in the SSR module is actually interacting SSR with MAC. In wireless networks, the packets are delivered to a shared medium and can thus be overheard by the sender's neighbors. SSR can utilize this feature to allow the neighbors to co-determine the route.

The evaluations presented in Chapter 13.3 show that with PNS and intermediate caching, the Lite-Ring system achieves a higher PDR as well as a lower packet delay. The simulations in Chapter 14.4 show that by the use of link-layer broadcast, the new proposed SSR achieves comparable routing efficiency as the original SSR but with a significantly smaller per-node state. Moreover, the proposed SSR enhancement is able to greatly improve SSR's performance in scenarios with high node mobility and node churn.

15.2 Discussion

Providing an efficient KBR service to multiple applications in wireless ad-hoc networks is a relatively new research topic. The Lite-Ring system together with some cross-layer designs offers a feasible solution. Nevertheless, there are still

some open issues concerning the performance of the Lite-Ring system for the future development.

15.2.1 PNS in the Balancing Mechanism of Lite-Ring

With topology information from the network layer, the Lite-Ring module could find some proximal nodes for bootstrapping. This reduces the physical distance of the overlay links. According to the balancing mechanism in the Lite-Ring protocol (cf. Chapter 5.5), the leaf nodes on the overweight side of the address tree will be moved to the underweight side. The balancing mechanism increases the hit rate of a correct estimation of the destination node ID for a given key. However, it might break the proximal overlay links and generate overlay links that cross the entire network. Thus, a proximity-aware balancing mechanism is desirable: when a node has to be shifted into a certain address range, it can ask the SSR module whether there are already some proximal nodes in this address range. If yes, it can rejoin the overlay via one of these nodes. Otherwise, the original balancing mechanism can be applied.

15.2.2 Caching the Data from DHT Result Messages

As demonstrated before, the DHT lookup performance is significantly improved when the intermediate nodes cache the data from the passing by `put` messages. The `get` messages could then retrieve the cached data before reaching the key's root. Obviously, the trade-off is the local state. When the intermediate nodes have enough cache space, they can also cache the data from the passing by `get-result` messages. This would further reduce the lookup time and network traffic. However, the space consumption would further increase.

15.2.3 Including MAC into Lite-Ring System

Although the SSR performance can be improved by using link-layer broadcast, the achieved performance gain is sub-optimal due to the additional handshaking overhead in the network layer. To avoid such overhead, modifications in the MAC protocols (for heterogeneous underlying layers) are necessary but very costly. Hence, in the general version of the Lite-Ring system, the cross-layer design between SSR and the MAC layer is not included. However, this cross-layer optimization will make sense when deploying the Lite-Ring system concretely in a specific wireless network, e. g. in an IEEE 802.11 network.

15.2.4 Asymmetric Links

According to some empirical studies [37, 52, 132], there frequently exist unidirectional links in wireless networks. Theoretically, the network performance could be improved by utilizing these links. However, only a few routing protocols support unidirectional links, e. g. DSR [63], some variants of AODV and DSDV [92].

Similar to most routing protocols, SSR works only with bidirectional links, i. e. unidirectional links are detected and then excluded in routing process. Birnstill et al. [11] proposed several algorithms for SSR, which enable SSR to use the

unidirectional links. They also demonstrated that the SSR performance can be improved by about 20% by utilizing these unidirectional links.

15.2.5 MAC Protocols

For sake of a brief illustration, only the IEEE 802.11 MAC layer has been simulated for this work. As mentioned in Chapter 14.2, some handshaking functions of the MAC layer are implemented in the enhanced SSR. This in turn increases packet delay and network traffic, as shown in the simulations in Chapter 14.4.1. As mentioned, modifications in the different MAC protocols are needed to avoid such overhead.

Note that the IEEE 802.11 MAC protocol used in this work is more relevant to the MANETs than the low-power WSNs. Since the energy efficiency is one of the most important factors in the WSNs, some specific low-power MAC protocols are developed, e. g. IEEE 802.15.4 MAC [61], S-MAC [128] and B-MAC [91]. Thus, if the SSR enhancement would be implemented for a WSN, a low-power MAC protocol should be used.

Part V

The End

Chapter 16

Conclusions

In this dissertation, I have presented the Lite-Ring overlay network protocol and the Lite-Ring system that implements the Lite-Ring protocol in wireless ad-hoc networks. Moreover, I have proposed some cross-layer optimizations for the Lite-Ring system.

Contributions

In recent decades, the application of structured overlays has become more and more popular, not only in conventional file-sharing programs, but also in many other areas, such as in distributed file systems, in distributed databases, etc.. One of the main features of a structured overlay is the Key-Based- Routing (KBR) service that it provides. KBR enables each overlay node to be cooperatively responsible for a certain range of the address space and to route the messages to their responsible nodes with a limited amount of overlay hops, typically $O(\log N)$) hops. In addition, the node state of a structured overlay is usually limited within $O(\log N)$. The small routing overhead and the small node state enable the applications to scale well.

Although the node state of a structured overlay is typically $O(\log N)$, the node state could be relatively large in practice due to the extra state introduced for performance improvement. Therefore, if multiple applications are installed locally and need KBR services at the same time, the overall node state could overload the node and/or its bandwidth.

Lite-Ring provides a new way to build structured overlay networks for various applications that need KBR services in a more efficient manner. The node in Lite-Ring overlay maintains only $O(1)$ state. However, it is able to provide the KBR service to an application just as a conventional full-fledged structured overlay does.

Lite-Ring applies a predicable addressing scheme, which uniformly distributes the node addresses in the address space. This, in turn, allows each node to estimate the overlay size based on the addresses of its overlay neighbors. Subsequently, the node is able to estimate the addresses of all the other nodes and thus calculate the address of the destination node for an arbitrary key. In short, the address of the destination node is calculated by local calculation instead of the recursive overlay routing procedure as applied by conventional structured overlays.

With the help of a public DHT service, where each node stores its transport layer address under its overlay address, the transport layer address of the destination node can be solved by one DHT request on the calculated overlay address, and the message can be delivered as long as the DHT request succeeds.

As the address estimation is rough, the calculation of the destination address can sometimes yield an incorrect result. Nevertheless, the delivery to an incorrect destination can be corrected either by the sender with new address calculations or by the receiver with further routing steps. As shown in the early chapters, if the address tree is balanced, the probability of an incorrect address calculation of the destination node is only approximately 17%, i. e. the address calculation yields the correct destination node address. In another words, the routing overhead of Lite-Ring is relatively small, i. e. only 0.17.

As previously stated, for a balanced address tree, the routing overhead is very small. To this end, different approaches have been proposed in the thesis in order to keep the address tree balanced.

In short, with the predictable address assignment schema, Lite-Ring achieves a small ($O(\log N)$) routing overhead despite a small ($O(1)$) node state. The message delivery in Lite-Ring requires in most cases only one DHT lookup to solve the transport layer address of the final destination, regardless of the application overlay size and the number of applications. Thus, Lite-Ring is able to provide a more efficient KBR service than other approaches, e. g. ReDiR and Diminished Chord.

The KBR service is not only required in many distributed applications in the Internet, but also in some distributed applications in wireless ad-hoc networks. However, there is still no well-known solution to provide efficient KBR services for multiple applications in wireless ad-hoc networks. In this thesis, I have also presented the Lite-Ring system to solve this problem. To the best of my knowledge, the Lite-Ring system is the first exploratory study aiming at providing KBR services to multiple applications in a wireless ad-hoc network.

The Lite-Ring system consists of three components: a Lite-Ring module that implements the Lite-Ring protocol and provides the KBR service to the application, a DHT module that provides a basis DHT service needed by the Lite- Ring protocol, and a SSR module that provides the routing service in the network layer. Some exploratory simulations have demonstrated the feasibility of the Lite-Ring system to provide KBR services to multiple applications.

In the later chapters, I have also introduced three cross-layer optimizations designed for the Lite-Ring system. They are the Proximal Neighbor Selection (PNS) in the Lite-Ring module, the use of data caching in the DHT module, and the link-layer broadcast in the SSR module. The simulation results have shown significant performance improvement by applying these optimizations.

Outlook

This work has proposed Lite-Ring overlay network protocol, a novel solution to build structured overlays and provide efficient KBR services for multiple applications. However, there are still some open topics that need to be investigated before it can be deployed in practice. For example, as a public DHT service is required by Lite-Ring and its reliability and performance have direct impact

on the quality of the KBR service that Lite-Ring provides, some survey of current public DHT services would be necessary. Moreover, the common security problems of P2P applications apply also to the Lite-Ring overlay network. although they are not in the scope of this thesis. Thus, the attack resistance of the Lite-Ring overlay network is also an important research topic in the future.

Chapter 17

Acknowledgments

The work presented here has been carried out during the years 2005-2009 at the Department of Informatics, Karlsruhe Institute of Technology as well as at the Department of Informatics, Technical University Munich, Germany.

First of all, I would like to express my sincere appreciation to my doctoral supervisor Dr. Thomas Fuhrmann for providing me the opportunity to work in the research area of informatics, for his expert guidance and mentorship, for his motivation and help at all levels.

Moreover, I would like to thank professor Dr. Georg Carle and professor Dr. Michael Gerndt for accepting to be the co-referees of this thesis.

Also, I would like to thank all my colleagues for the good and encouraging collaboration, for their friendship and for many enjoyable times. Especially I would like to mention Dr. Kendy Kutzner, Carola Kutzner, Dr. Curt Cramer, Stefan Denk, Dr. Sean O'Donoghue, Yaser Houri, Dr. Björn Saballus, Johannes Eickhold, Sven Schlender, Pascal Birnstill, Dr. Benedikt Elser and Dr. Bernhard Amann. Thank you!

Many special thanks to my parents for their love and for their warm encouragement to let me be able to finish this thesis.

Appendix A

Balancing Cost with Node Count

In this appendix I briefly describe the derivation of the balancing cost in the case that the nodes use the node count information (cf. Chapter 5.5.2).

Let's define the symbol $P_{a,n}^{p,q}$ as follows: Assume that there are a positions in the lowest level in total, of which n are occupied and $(m = a - n)$ vacant. $P_{a,n}^{p,q}$ is the probability that when choosing p positions from all the a positions q positions among the p chosen positions are already occupied. Simple combinatorics yields:

$$P_{a,n}^{p,q} = (\prod_{i=0}^{q-1} \frac{n-i}{a-i})(\prod_{i=0}^{p-q-1} \frac{a-n-i}{a-q-i})(\prod_{i=0}^{q-1} \frac{a-i}{i+1}) \tag{A.1}$$

When a new node joins the overlay, it might choose a position that makes the tree become unbalanced. In that case, one of its parent nodes will detect the imbalance. It thus recommends the node to move to the subtree on the other side of said parent node. Figure A.1(c) illustrates an example: The new node joins below node 1 and thus causes the tree to become unbalanced. Node 4 detects the imbalance and recommends the newly joined node to move to another position.

Let $P(d)$ be the probability for an unbalanced tree with depth d. Then:

$$P(d) = P_{a,n}^{2^d,2^d} - P_{a,n}^{2^{d+1},2^{d+1}}$$

In the example of Figure A.1(c), where $a = 8$ and $n = 2$, we can get the probability for this case as follows:

$$P(2) = P_{8,2}^{2,2} - P_{8,2}^{4,4} = \frac{2}{8} \cdot \frac{1}{7} - 0 = \frac{1}{28}$$

Let $J(a, n)$ be the expected number of node shifts of the newly joining node, where there exist in total a positions in the lowest level of the address tree and n of them are occupied. Thus, for an address tree with 10 nodes, the expected number of node shifts of the joining node, $J(8, 2)$, is:

$$
\begin{aligned}
J(8,2) &= 0 \cdot P(0) + 1 \cdot P(1) + 1 \cdot P(2) + 1 \cdot P(3) \\
&= 0 \cdot P_{8,2}^{1,0} + 1 \cdot (P_{8,2}^{1,1} - P_{8,2}^{2,2}) + 1 \cdot (P_{8,2}^{2,2} - P_{8,2}^{4,4}) + 1 \cdot P_{8,2}^{4,4} \\
&= 0.25
\end{aligned}
$$

This formula corresponds the three possible scenarios in Figure A.1. The coefficients in the formula are the expected number of node shifts in each scenario and they can be achieved from Equation A.2. (Note that in this case, it is not possible that the imbalance happens at a depth greater than 2, i. e. $P(3) = 0$.)

In general, the $(n+1)^{th}$ node in the lowest level needs to shift its position $J(a,n)$ times to balance the tree, where

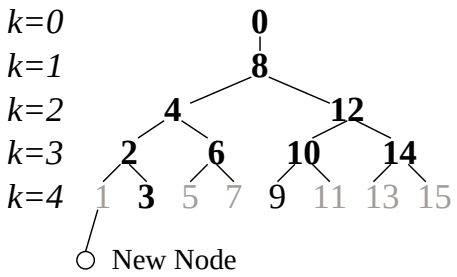
$$
\begin{cases}
J(1,0) = 0 \\
J(a,n) = \sum_{i=1}^{\log_2 a} P(i) \cdot (\sum_{j=0}^{i-1} P_{a-2^i, n-2^i}^{j,2^i} \cdot J(2^i, j))
\end{cases}
\tag{A.2}
$$

where $P_{a-2^i,n-2^i}^{j,2^i}$ is the probability for the various node distributions in the other side of the subtree of the parent node that detects the imbalance and $J(2^i, j)$ is the expected node shifts further within the subtree.

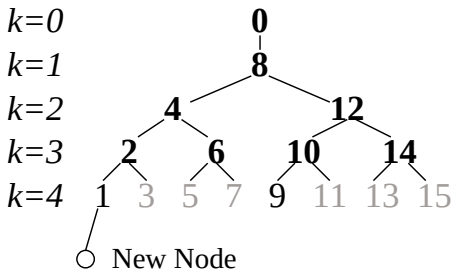
So, let $J'(n)$ be the expected number of address shifts for the n^{th} node of the entire overlay. We can get:

$$
J'(n) = J(2^l, n - 2^l)
$$

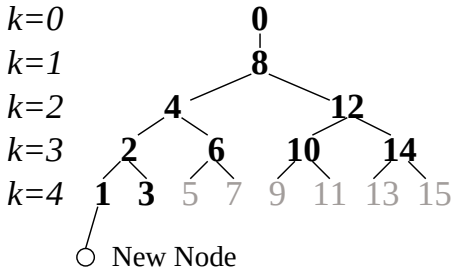
where $l = \lfloor log_2(n-1) \rfloor$ is the depth of the existing completely filled part of the binary tree.



(a) Balanced tree with probability $P(0)$



(b) Unbalanced tree with depth 1. The probability is $P(1)$



(c) Unbalanced tree with depth 2. The probability is $P(2)$

Figure A.1: Different scenarios when a node joins

Appendix B

Curriculum Vitae

Pengfei Di

Address Schlippehof 3
79110 Freiburg
Germany

E-Mail pengfei.di@live.de

Date and place of birth

1978-Oct-06 in Jiangsu, China

Education

01/2009 - 12/2009 **Ph. D Candidate**, Department of Informatics, Technical University Munich, Germany. Dissertation Title "Providing Efficient Key Based Routing for Multiple Applications".

04/2005 - 12/2008 **Ph. D Candidate**, Department of Informatics, Karlsruhe Institute of Technology, Germany.

09/2002 - 02/2005 **Master of Science in Computational Science in Engineering**, Technical University Braunschweig, Germany. Master Thesis "Simulation and Evaluation of DCCP-lite in OPNET Modeler".

09/1997 - 07/2002 **Bachelor in Mechanical Engineering**, Tongji University, Shanghai, China.

09/1994 - 07/1997 High School of Liyang, Jiangsu, China

Professional Experience

03/2013 - **Senior Software Engineer**, Infor Global Solutions, Breisach, Germany

10/2010 - 06/2012	**Software Developer**, match2blue Software Development GmbH, Jena, Germany
01/2009 - 12/2009	**Research Assistant**, Technical University Munich, Germany
04/2005 - 12/2009	**Research Assistant**, Karlsruhe Institute of Technology, Germany

Professional Affiliations

2009 – 2011	TPC Member of the IEEE Consumer Communications and Networking Conference (CCNC)
2009	IEEE Student Member

Knowledge of Languages

Chinese	Native
English	Fluent
German	Fluent

Miscellaneous

2001	Award of "Excellent Graduate of Shanghai 2001", Shanghai, China
1998-2001	University Scholar, Tongji University, Shanghai, China
01/2010 - 09/2010	Parental leave

Freiburg im Breisgau, Germany, August 2014

Appendix C

List of Publications

- Pengfei Di, Matthias Wählisch, Georg Wittenburg, "Modeling the Network Layer and Routing Protocols". In Modeling and Tools for Network Simulation (Klaus Wehrle, Mesut Günes, James Gross Ed.), Pages 359-384, Springer, 2010, ISBN: 978-3-642-12330-6

- Pascal Birnstill, Pengfei Di and Thomas Fuhrmann, "Using Asymmetric Links to Improve SSR's Routing Performance". In Proceedings of the 9th IFIP Annual Mediterranean Ad Hoc Networking Workshop (MedHocNet'10), Juan-Les-Pins, France, June 23-25, 2010

- Pengfei Di and Thomas Fuhrmann, "Scalable Landmark Flooding - A Scalable Routing Protocol for WSNs". In Proceedings of the CoNEXT Student Workshop, Rome, Italy, December 1, 2009

- Pengfei Di and Thomas Fuhrmann, "Using Link-Layer Broadcast to Improve Scalable Source Routing". In Proceedings of the 5th International Conference on Wireless Communications and Mobile Computing (IWCMC'09), Leipzig, Germany, June 21-24, 2009

- Pengfei Di, Yaser Houri, Kendy Kutzner and Thomas Fuhrmann, "Towards Comparable Network Simulations". Technical Report 2008-9, ISSN 1432-7864, Department of Informatics, Universität Karlsruhe (TH), Germany, August 2008

- Pengfei Di, Johannes Eickhold, and Thomas Fuhrmann, "Linyphi: Creating IPv6 Mesh Networks with SSR". Concurrency and Computation: Practice and Experience, Volume 20, Issue 6, Pages 675-691, April 2008

- Pengfei Di, Kendy Kutzner and Thomas Fuhrmann, "Providing KBR Service for Multiple Applications". In Proceedings of the 7th International Workshop on Peer-to-Peer Systems (IPTPS'08), Tampa Bay, Florida, USA, February 25-26, 2008

- Pengfei Di, M. Yaser Houri, Qing Wie, Joerg Widmer, and Thomas Fuhrmann, "Application of DHT-Inspired Routing for Object Tracking". In Proceedings of the 4th IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS'07), Pisa, Italy, October 8-11, 2007

- Thomas Fuhrmann, Pengfei Di, Kendy Kutzner, and Curt Cramer, "Pushing Chord into the Underlay: Scalable Routing for Hybrid MANETs". Technical Report 2006-12, Department of Informatics, Universität Karlsruhe (TH), Germany, June 2006

- Pengfei Di, Massimiliano Marcon, and Thomas Fuhrmann, "Linyphi: An IPv6-Compatible Implementation of SSR". In Proceedings of the 3rd International Workshop on Hot Topics in Peer-to-Peer Systems (HotP2P'06), Rhodes Island, Greece, April 25-29, 2006

- Xiaoyuan Gu, Pengfei Di, and Lars Wolf, "Performance Evaluation of DCCP: A Focus on Smoothness and TCP-friendliness". Annals of Telecommunications Journal, Special Issue on Transport Protocols for Next Generation Networks, Volume 61, No. 1, Pages 191-216, January 2006

- Pengfei Di, "Simulation and Evaluation of DCCP-lite in OPNET Modeler". Master Thesis, Technical University Braunschweig, Germany, 2005

List of Figures

List of Tables

Acronyms

ACK	Acknowledgment
ACL	Access Control List
ALM	Application Layer Multicast
AODV	Ad-hoc On-demand Distance Vector
API	Application Programming Interface
AS	Autonomous System
ATP	Ad-hoc Transport Protocol
BGP	Border Gateway Protocol
BLR	Beacon-Less Routing
CAN	Content Addressable Network
CDN	Content Distribution Network
CTS	Clear to Send
DCF	Distributed Coordination Function
DCS	Data-Centric Storage,
DHT	Distributed Hash Table
DNS	Domain Name System+
DOLR	Decentralized Object Location and Routing
DSL	Digital Subscriber Line
DSR	Dynamic Source Routing
DSDV	Destination-Sequenced Distance Vector
ExOR	Extremely Opportunistic Routing
FIB	Forwarding Information Base
GHT	Geographic Hash Table
GNP	Global Network Positioning
GPS	Global Positioning System
GPSR	Greedy Perimeter Stateless Routing
HTTP	Hypertext Transfer Protocol
i3	Internet Indirection Infrastructure
ICE	Interactive Connectivity Establishment
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IGF	Implicit Geographic Forwarding
IP	Internet Protocol
IRC	Internet Relay Chat
ISP	Internet Service Provider

KBR	Key Based Routing
LER	Last Encounter Routing
LRU	Least Recently Used
LS	Link State
LSU	Link State Update
MAC	Medium Access Control
MANET	Mobile Ad-hoc Network
MORE	MAC-independent Opportunistic Routing and Encoding
MPR	Multipoint Relay
NAT	Network Address Translation
NAV	Network Allocation Vector
NP	Nondeterministic Polynomial
OLSR	Optimized Link State Routing
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
P2P	Peer-to-Peer
PDR	Packet Delivery Ratio
PIS	Proximal Identifier Selection
PNS	Proximal Neighbor Selection
PPP	Point-to-Point Protocol
PRS	Proximal Route Selection
ReDiR	Recursive Distributed Rendezvous scheme
RFID	Radio Frequency Identification
RIP	Routing Information Protocol
RPC	Remote Procedure Call
RREQ	Route Request
RSVP	Resource Reservation Protocol
RTS	Ready to Send
RTT	Round-Trip Time
SHA	Secure Hash Algorithm
SSR	Scalable Source Routing
STUN	Session Traversal Utilities for NAT
TC	Topology Control
TCP	Transmission Control Protocol
TIV	Triangle Inequality Violation
TTL	Time To Live
UDP	User Datagram Protocol
URL	Universal Resource Locator
VRR	Virtual Ring Routing
WLAN	Wireless Local Area Network
WMN	Wireless Mesh Network
WSN	Wireless Sensor Network
XOR	Exclusive OR
XML	Extensible Markup Language

Bibliography

[1] A C++ Implementation of ReDiR. [Online]. Available: `http://opendht.org/redir++-0.3.tar.gz` Accessed 30-Sep-2010.

[2] The aMule Project. [Online]. Available: `http://www.amule.org/` Accessed 30-Sep-2010.

[3] The eMule Project. [Online]. Available: `http://www.emule-project.net/` Accessed 30-Sep-2010.

[4] Vuze - A BitTorrent Client. [Online]. Available: `http://www.vuze.com/` Accessed 30-Sep-2010.

[5] Skype Fast Facts Q4 2008. eBay Ink, 2009. [Online]. Available: `http://ebayinkblog.com/wp-content/uploads/2009/01/skype-fast-facts-q4-08.pdf` Accessed 30-Sep-2010.

[6] Cisco Visual Networking Index: Forecast and Methodology, 2009-2014. White Paper, Cisco, June 2010. [Online]. Available: `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf` Accessed 29-Sep-2010.

[7] Vinay Aggarwal, Anja Feldmann, and Christian Scheideler. Can ISPs and P2P Users Cooperate for Improved Performance? *ACM SIGCOMM Computer Communication Review*, 37(3):29–40, 2007.

[8] Marc Sánchez Artigas, Pedro García López, Jordi Pujol Ahulló, and Antonio F. Gómez-Skarmeta. Cyclone: A Novel Design Schema for Hierarchical DHTs. In *Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 49–56, Konstanz, Germany, August 31–September 2 2005.

[9] Richard Ernest Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[10] Ruchir Bindal, Pei Cao, William Chan, Jan Medved, George Suwala, Tony Bates, and Amy Zhang. Improving Traffic Locality in BitTorrent via Biased Neighbor Selection. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, page 66, Washington, DC, USA, 2006.

[11] Pascal Birnstill. Efficient Detection and Utilization of Asymmetric Links in Scalable Source Routing (SSR). Diploma thesis, System Architecture Group, University of Karlsruhe, Germany, May 2009.

[12] Sanjit Biswas and Robert Morris. Opportunistic Routing in Multi-Hop Wireless Networks. In *Proceedings of ACM SIGCOMM'05*, Philadelphia, PA, USA, August 2005.

[13] Brain M. Blum, Tian He, Sang Son, and John Stankovic. IGF: A State-Free Robust Communication Protocol for Wireless Sensor Networks. Technical report, Department of Computer Science, University of Virginia, USA, 2003.

[14] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) — Version 1 Functional Specification. RFC 2205, September 1997.

[15] Matthew Caesar, Miguel Castro, Edmund B. Nightingale, Greg O'Shea, and Antony Rowstron. Virtual Ring Routing: Network Routing Inspired by DHTs. In *Proceedings of ACM SIGCOMM'06*, Pisa, Italy, September 2006.

[16] Claudio Casetti, Mario Gerla, Saverio Mascolo, M. Y. Sanadidi, and Ren Wang. TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks. *Wireless Networks*, 8(5):467–479, 2002.

[17] Marcel C. Castro, Andreas J. Kassler, Carla-Fabiana Chiasserini, Claudio Casetti, and Ibrahim Korpeoglu. *Handbook of Peer-to-Peer Networking*, chapter Peer-to-Peer Overlay in Mobile Ad-hoc Networks. Springer, July 2009.

[18] Marcel C. Castro, Eva Villanueva, Iraide Ruiz, Susana Sargento, and Andreas J. Kassler. Performance Evaluation of Structured P2P over Wireless Multi-hop Networks. In *Proceedings of the 2nd International Conference on Sensor Technologies and Applications (SENSORCOMM'08)*, pages 796–801, Washington, DC, USA, 2008.

[19] Miguel Castro, Manuel Costa, and Antony Rowstron. Should we build Gnutella on a structured overlay? *ACM SIGCOMM Computer Communications Review*, 34(1):131–136, 2004.

[20] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, pages 299–314, Boston, MA, USA, 2002.

[21] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. One Ring to Rule them All: Service Discovery and Binding in Structured Peer-to-Peer Overlay Networks. In *Proceedings of the 10th ACM SIGOPS European Workshop*, Saint-Emilion, France, September 2002.

[22] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[23] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. Trading Structure for Randomness in Wireless Opportunistic Routing. *ACM SIGCOMM Computer Communication Review*, 37:169–180, October 2007.

[24] Mohit Chawla, Nishith Goel, Kalai Kalaichelvan, Amiya Nayak, and Ivan StojmenovicIvan. Beaconless Position Based Routing with Guaranteed Delivery for Wireless Ad-Hoc and Sensor Networks. In *Proceedings of the 19th IFIP World Computer Congress*, Santiago de Chile, Chile, August 2006.

[25] David R. Choffnes and Fabián E. Bustamante. Taming the Torrent: A Practical Approach to Reducing Cross-ISP Traffic in Peer-to-Peer Systems. *ACM SIGCOMM Computer Communication Review*, 38(4):363–374, 2008.

[26] T. Clausen and P. Jacquet. Optimized Link State Routing protocol (OLSR). RFC 3626, October 2003.

[27] Bram Cohen. Incentives Build Robustness in BitTorrent. In *Proceeding of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, May 2003.

[28] Edith Cohen and Scott Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *Proceedings of ACM SIGCOMM'02*, pages 177–190. 2002.

[29] Curt Cramer and Thomas Fuhrmann. ISPRP: A Message-Efficient Protocol for Initializing Structured P2P Networks. In *Proceedings of the 24th IEEE International Performance, Computing, and Communications Conference (IPCCC'05)*, pages 365–370, Phoenix, AZ, USA, April 2005.

[30] Curt Cramer and Thomas Fuhrmann. Proximity Neighbor Selection for a DHT in Wireless Multi-Hop Networks. In *Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 3–10, Konstanz, Germany, August 31–September 2 2005.

[31] Curt Cramer and Thomas Fuhrmann. Performance Evaluation of Chord in Mobile Ad Hoc Networks. In *Proceedings of the 1st International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking (ACM MobiShare)*, Los Angeles, CA, USA, September 2006.

[32] Curt Cramer, Kendy Kutzner, and Thomas Fuhrmann. Bootstrapping Locality-Aware P2P Networks. In *Proceedings of the IEEE International Conference on Networks (ICON'04)*, volume 1, pages 357–361, Singapore, November 2004.

[33] Curt Cramer, Kendy Kutzner, and Thomas Fuhrmann. Distributed Job Scheduling in a Peer-to-Peer Video Recording System. In *Proceedings of the Workshop on Algorithms and Protocols for Efficient Peer-to-Peer Applications (PEPPA)*, pages 234–238, Ulm, Germany, September 23 2004.

[34] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proceedings of ACM SIGCOMM'04*, Portland, OR, USA, August 2004.

[35] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris. Designing a DHT for low latency and high throughput. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI'04)*, San Francisco, CA, USA, March 2004.

[36] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, USA, February 2003.

[37] Douglas S. J. De Couto, Daniel Aguayo, Benjamin A. Chambers, and Robert Morris. Performance of Multihop Wireless Networks: Shortest Path is Not Enough. *ACM SIGCOMM Computer Communication Review*, 33(1):83–88, 2003.

[38] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's Highly Available Key-value Store. *ACM SIGOPS Operating Systems Review*, 41:205–220, 2007.

[39] Franca Delmastro. From Pastry to CrossROAD: CROSS-Layer Ring Overlay for AD Hoc Networks. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications Workshops(PERCOMW'05)*, pages 60–64, Washington, CA, USA, 2005.

[40] Luc Devroye. A Note on the Height of Binary Search Trees. *Journal of the ACM*, 33(3):489–498, 1986.

[41] Pengfei Di and Thomas Fuhrmann. Scalable Landmark Flooding - A Scalable Routing Protocol for WSNs. In *Proceedings of the 5th ACM CoNEXT Student Workshop*, Rome, Italy, December 2009.

[42] Pengfei Di and Thomas Fuhrmann. Using Link-Layer Broadcast to Improve Scalable Source Routing. In *Proceedings of the 5th International Conference on Wireless Communications and Mobile Computing (IWCMC'09)*, Leipzig, Germany, June 2009.

[43] Pengfei Di, M. Yaser Houri, Qing Wei, Jorg Widmer, and Thomas Fuhrmann. Application of DHT-Inspired Routing for Object Tracking. In *Proceedings of the 4th IEEE International Conference on Mobile Adhoc and Sensor Systems Conference (MASS'07)*, pages 1–9, Pisa, Italy, October 2007.

[44] Pengfei Di, Kendy Kutzner, and Thomas Fuhrmann. Providing KBR Service for Multiple Applications. In *Proceedings of the 7th International Workshop on Peer-to-Peer Systems (IPTPS'08)*, St. Petersburg, FL, USA, February 2008.

[45] Edsger Dijkstra. A Note on Two Problems in Connection with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[46] Peter Druschel and Antony Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, pages 75–80, May 2001.

[47] Jakob Eriksson, Michalis Faloutsos, and Srikanth Krishnamurthy. DART: Dynamic Address RouTing for Scalable Ad Hoc and Mesh Networks. *IEEE/ACM Transactions on Networking*, 15:119–132, 2007.

[48] Paul Francis, Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. IDMaps: A Global Internet Host Distance Estimation Service. *IEEE/ACM Transactions on Networking*, 9(5):525–540, 2001.

[49] Cheng Peng Fu and Soung C. Liew. TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks. *IEEE Journal on Selected Areas in Communications*, 21:216–228, 2003.

[50] Thomas Fuhrmann. Scalable Routing for Networked Sensors and Actuators. In *Proceedings of the 2nd Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'05)*, Santa Clara, CA, USA, September 2005.

[51] Thomas Fuhrmann. Performance of Scalable Source Routing in Hybrid MANETs. In *Proceedings of the 4th Annual Conference on Wireless On demand Network Systems and Services (WONS'07)*, pages 122–129, Obergurgl, Austria, January 24–26 2007.

[52] Deepak Ganesan, Bhaskar Krishnamachari, Alec Woo, David Culler, Deborah Estrin, and Stephen Wicker. An Empirical Study of Epidemic Algorithms in Large Scale Multihop Wireless Networks. Technical report, Intel Research, 2002.

[53] Matthias Grossglauser and Martin Vetterli. Locating Mobile Nodes With EASE: Learning Efficient Routes From Encounter Histories Alone. *IEEE/ACM Transaction on Networking*, 14(3):457–469, 2006.

[54] Krishna Gummadi, Ramakrishna Gummadi, Steve Gribble, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proceedings of ACM SIGCOMM'03*, pages 381–394, Karlsruhe, Germany, 2003.

[55] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement (IMW'02)*, pages 5–18, Marseille, France, November 2002.

[56] Piyush Gupta and P. R. Kumar. The Capacity of Wireless Networks. *IEEE Transactions on Information Theory*, 46(2):388–404, March 2000.

[57] Marc Heissenbüttel, Torsten Braun, Thomas Bernoulli, and Markus Wächtli. BLR: Beacon-Less Routing Algorithm for Mobile Ad-Hoc Network. *Elsevier's Computer Communications Journal (ECC)*, 27(11):1076–1086, 2004.

[58] Pai-Hsiang Hsiao. Geographical region summary service for geographical routing. *SIGMOBILE Mobile Computing and Communications Review*, 5(4):25–39, 2001.

[59] IEEE 802.11 Working Group. IEEE Standard 802.11-1999: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications., 1999.

[60] IEEE 802.15 Working Group. Specification of the Bluetooth System, 2002.

[61] IEEE 802.15 Working Group. IEEE Standard for Information Technology-Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Low Rate Wireless Personal Area Networks (LR-WPANS), 2006.

[62] Eu jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. Sirius: Securing Remote Untrusted Storage. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium (NDSS'03)*, pages 131–145, San Diego, CA, USA, February 2003.

[63] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. *Mobile Computing*, 353:153–181, February 1996.

[64] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer System (IPTPS'03)*, Berkeley, CA, USA, February 2003.

[65] Gene Kan. Gnutella. In *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, pages 94–122. O'Reilly, Sebastopol, CA, USA, 2001.

[66] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proceedings of the 29th annual ACM Symposium on Theory of Computing*, pages 654–663. 1997.

[67] David Karger, Alex Sherman, Andy Berkheimer, Bill Bogstad, Rizwan Dhanidina, Ken Iwamoto, Brian Kim, Luke Matkins, and Yoav Yerushalmi. Web caching with consistent hashing. In *Proceedings of the 8th International Conference on World Wide Web (WWW'99)*, pages 1203–1213, Toronto, Canada, 1999.

[68] David R. Karger and Matthias Ruhl. Diminished Chord: A Protocol for Heterogeneous Subgroup Formation in Peer-to-Peer Networks. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*, pages 288–297, San Diego, CA, USA, 2004.

[69] David R. Karger and Matthias Ruhl. Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems. In *Proceedings of the 3th International Workshop on Peer-to-Peer Systems (IPTPS'04)*, San Diego, CA, USA, 2004.

[70] Brad Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom'00)*, pages 243–254, Boston, MA, USA, August 2000.

[71] D. Katz. IP Router Alert Option. RFC 2113, February 1997.

[72] Vikas Kawadia and P.R. Kumar. A Cautionary Perspective on Cross Layer Design. *IEEE Wireless Communication*, 12(1):3–11, February 2005.

[73] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 190–201. 2000.

[74] Kendy Kutzner and Thomas Fuhrmann. Measuring Large Overlay Networks - The Overnet Example. In *Konferenzband der 14. Fachtagung Kommunikation in Verteilten Systemen (KiVS'05)*, Kaiserslautern, Germany, 2005.

[75] Kendy Kutzner and Thomas Fuhrmann. The IGOR File System for Efficient Data Distribution in the GRID. In *Proceedings of the Cracow Grid Workshop (CGW'06)*, Cracow, Poland, October 2006.

[76] Kendy Kutzner and Thomas Fuhrmann. Using Linearization for Global Consistency in SSR. In *Proceedings of the 4th International IEEE Workshop on Hot Topics in P2P Systems (HotP2P'07)*, Long Beach, CA, USA, March 2007.

[77] Kendy Kutzner, Christian Wallenta, and Thomas Fuhrmann. Securing the Scalable Source Routing Protocol. In *Proceedings of the World Telecommunications Congress 2006*, Budapest, Hungary, April 30–May 3 2006.

[78] Sanghwan Lee, Zhi-Li Zhang, Sambit Sahu, and Debanjan Saha. On Suitability of Euclidean Embedding of Internet Hosts. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/Performance'06)*, pages 157–168, Saint Malo, France, 2006.

[79] Jinyang Li, Charles Blake, Douglas S.J. De Couto, Hu Imm Lee, and Robert Morris. Capacity of Ad Hoc Wireless Networks. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom'01)*, pages 61–69, Rome, Italy, June 2001.

[80] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A Scalable Location Service for Geographic Ad Hoc Routing. In *Proceedings of the 6th Annual International Conference on Mobile*

Computing and Networking (MobiCom'00), pages 120–130, Boston, MA, USA, 2000.

[81] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of the 16th ACM International Conference on Supercomputing (ICS'02)*, pages 258–259, New York, NY, USA, June 2002.

[82] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, 2002.

[83] Petar Maymounkov and David Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, pages 53–65. 2002.

[84] National Institute of Standards and Technology. US Secure Hash Standard. Federal Information Processing Standards Publication 180-1, April 1995.

[85] T. S. Eugene Ng and Hui Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*, volume 1, pages 170–179, 2002.

[86] Lionel M. Ni, Yunhao Liu, Yiu Cho Lau, and Abhishek P. Patil. LANDMARC: Indoor Location Sensing Using Active RFID. *Wireless Networks*, 10(6):701–710, 2004.

[87] Christina Parsa and J. J. Garcia-Luna-Aceves. Improving TCP Performance over Wireless Networks at the Link Layer. *Mobile Networks and Applications*, 5(1):57–71, 2000.

[88] Charles E. Perkins and Elizabeth M. Royer. Ad hoc On-Demand Distance Vector Routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, pages 90–100, New Orleans, LA, USA, February 1999.

[89] Charles E. Perkins and Kuang-Yeh Wang. Optimized Smooth Handoffs in Mobile IP. In *Proceedings of the IEEE Symposium on Computers and Communications (ISCC'99)*, page 340, Los Alamitos, CA, USA, 1999.

[90] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. *ACM SIGCOMM Computer Communication Review*, 33(1):59–64, 2003.

[91] Joseph Polastre, Jason Hill, and David Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*, pages 95–107, Baltimore, MD, USA, 2004.

[92] Ravi Prakash. A Routing Algorithm for Wireless Ad Hoc Networks with Unidirectional Links. *Wireless Networks*, 7:617 – 625, November 2001.

[93] Amir Qayyum. *Analysis and Evaluation of Channel Access Schemes and Routing Protocols for Wireless Networks*. PhD thesis, University of Paris Sud, Orsay, France, 2000.

[94] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM'01*, pages 161–172, San Diego, CA, USA, August 2001.

[95] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Topologically-Aware Overlay Construction and Server Selection. In *Proceedings of the 21th IEEE Conference on Computer Communications (INFOCOM'02)*, New York, NY, USA, June 2002.

[96] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Application-Level Multicast Using Content-Addressable Networks. In *Proceedings of the 3rd International COST264 Workshop on Networked Group Communication (NGC'01)*, pages 14–29, London, UK, 2001.

[97] Sylvia Ratnasamy, Brad Karp, Li Yin, Fang Yu, Deborah Estrin, Ramesh Govindan, and Scott Shenker. GHT: A Geographic Hash Table for Data-Centric Storage. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, USA, September 2002.

[98] Sean Rhea, Byung-Gon Chun, John Kubiatowicz, and Scott Shenker. Fixing the Embarrassing Slowness of OpenDHT on PlanetLab. In *Proceedings of the 2nd Workshop on Real, Large, Distributed Systems (WORLDS'05)*, San Francisco, CA, USA, December 2005.

[99] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling Churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*, Boston, MA, USA, June 2004.

[100] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. OpenDHT: a Public DHT Service and Its Uses. In *Proceedings of ACM SIGCOMM'05*, pages 73–84, Philadelphia, PA, USA, August 2005.

[101] Rodrigo Rodrigues and Barbara Liskov. High availability in DHTs: Erasure Coding vs. Replication. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS'05)*, 2005.

[102] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389, October 2008.

[103] Jonathan Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245, April 2010.

[104] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware'01)*, Heidelberg, Germany, November 2001.

[105] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP'01)*, Lake Louise, Banff, Canada, October 2001.

[106] Juan A. Sanchez, Rafael Marin-Perez, and Pedro M. Ruiz. Beacon-less Geographic Routing in Real Wireless Sensor Networks. In *Proceedings of the 4th IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems (MASS'07)*, Pisa, Italy, October 2007.

[107] Maurice Lorrain Schlumberger. *De-Bruijn Communications Networks.* PhD thesis, Stanford University, Palo Alto, CA, USA, 1974.

[108] Hendrik Schulze and Klaus Mochalski. Internet Study 2008/2009. ipoque GmbH, 2009. [Online]. Available: `http://www.ipoque.com/study/ipoque-Internet-Study-08-09.pdf` Accessed 30-Sep-2010.

[109] Sanjay Shakkottai, Theodore S. Rappaport, and Peter C. Karlsson. Cross-Layer Design for Wireless Networks. *IEEE Communications Magazine*, 41(10):74–80, October 2003.

[110] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022, January 2001.

[111] V. Srivastava and M. Motani. Cross-Layer Design: A Survey and the Road Ahead. *IEEE Communications Magazine*, 43(12):112–119, December 2005.

[112] Moritz Steiner, Taoufik En Najjary, and Ernst W Biersack. A global view of KAD. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference (IMC'07)*, San Diego, CA, USA, October 2007.

[113] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet Indirection Infrastructure. In *Proceedings of ACM SIGCOMM'02*, pages 73–86, Pittsburgh, PA, USA, 2002.

[114] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM'01*, pages 149–160, San Diego, CA, USA, August 2001.

[115] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003.

[116] Daniel Stutzbach and Reza Rejaie. Improving Lookup Performance Over a Widely-Deployed DHT. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM'06)*, Barcelona, Spain, April 2006.

[117] Karthikeyan Sundaresan, Vaidyanathan Anantharaman, Hung-Yun Hsieh, and Raghupathy Sivakumar. ATP: A Reliable Transport Protocol for Adhoc Networks. In *Proceedings of the 4th ACM International Symposium*

on Mobile Ad Hoc Networking and Computing (MobiHoc'03), pages 64–75, Annapolis, MD, USA, 2003.

[118] R. Thurlow. RPC: Remote Procedure Call Protocol Specification Version 2. RFC 5531, May 2009.

[119] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. A Survey of DHT Security Techniques. *ACM Computing Surveys*, 43(2), June 2009. `http://www.globule.org/publi/SDST_acmcs2009.html`, to appear.

[120] András Varga. INET Framework for OMNeT++. http://inet.omnetpp.org.

[121] András Varga. The OMNeT++ Discrete Event Simulation System. In *Proceedings of the 15th European Simulation Multiconference (ESM'01)*, Prague, Czech Republic, June 2001.

[122] Laurent Viennot, Laurent Viennot, and Projet Hipercom. Complexity Results on Election of Multipoint Relays in Wireless Networks. Technical report, Report RR-3584, INRIA, France, December 1998.

[123] Guohui Wang, Bo Zhang, and T. S. Eugene Ng. Towards Network Triangle Inequality Violation Aware Distributed Systems. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC'07)*, pages 175–188, San Diego, CA, USA, 2007.

[124] Dave Winer. XML-RPC Specification, 1999. [Online]. Available: `http://www.xmlrpc.com/spec` Accessed 30-Sep-2010.

[125] Rolf Winter, Thomas Zahn, and Jochen Schiller. Random Landmarking in Mobile, Topology-Aware Peer-to-Peer Networks. In *Proceedings of the 10th International Workshop on Future Trends in Distributed Computing Systems (FTDCS'04)*, pages 319–324, Washington, DC, USA, 2004.

[126] Bernard Wong, Aleksandrs Slivkins, and Emin Gün Sirer. Meridian: A Lightweight Network Location Service without Virtual Coordinates. In *Proceedings of ACM SIGCOMM'05*, pages 85–96, Philadelphia, PA, USA, August 2005.

[127] Kai Xu, Ye Tian, Nirwan Ansari, and Senior Member. TCP-Jersey for Wireless IP Communications. *IEEE Journal on Selected Areas in Communications*, 22:747–756, 2004.

[128] Wei Ye, John Heidemann, and Deborah Estrin. Medium Access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks. *IEEE/ACM Transactions on Networking*, 12(3):493–506, 2004.

[129] Thomas Zahn and Jochen Schiller. MADPastry: A DHT Substrate for Practicably Sized MANETs. In *Proceedings of the 5th Workshop on Applications and Services in Wireless Networks (ASWN'05)*, Paris, France, June 2005.

[130] Bo Zhang, T. S. Eugene Ng, Animesh Nandi, Rudolf Riedi, Peter Druschel, and Guohui Wang. Measurement Based Analysis, Modeling, and Synthesis of the Internet Delay Space. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement (IMC'06)*, pages 85–98, Rio de Janeriro, Brazil, 2006.

[131] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.

[132] Jerry Zhao and Ramesh Govindan. Understanding Packet Delivery Performance in Dense Wireless Sensor Networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*, Los Angeles, CA, USA, November 2003.

[133] Han Zheng, Eng Keong Lua, Marcelo Pias, and Timothy G. Griffin. Internet Routing Policies and Round-Trip-Times. In *Proceedings of the Passive and Active Measurement Workshop*, Boston, MA, USA, March 2005.