

---

# **Kademlia Documentation**

*Release 2.2.1*

**Brian Muller**

**May 03, 2020**



# CONTENTS

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Usage</b>	<b>5</b>
<b>3</b>	<b>Running Tests</b>	<b>7</b>
<b>4</b>	<b>Fidelity to Original Paper</b>	<b>9</b>
<b>5</b>	<b>Querying the DHT</b>	<b>11</b>
<b>6</b>	<b>Kademlia API</b>	<b>13</b>
6.1	kademlia package . . . . .	13
6.1.1	kademlia.crawling module . . . . .	13
6.1.2	kademlia.network module . . . . .	14
6.1.3	kademlia.node module . . . . .	14
6.1.4	kademlia.protocol module . . . . .	15
6.1.5	kademlia.routing module . . . . .	15
6.1.6	kademlia.storage module . . . . .	16
6.1.7	kademlia.utils module . . . . .	17
<b>7</b>	<b>Indices and tables</b>	<b>19</b>
	<b>Python Module Index</b>	<b>21</b>
	<b>Index</b>	<b>23</b>



---

**Note:** This library assumes you have a working familiarity with [asyncio](#).

---

This library is an asynchronous Python implementation of the [Kademlia distributed hash table](#). It uses [asyncio](#) to provide asynchronous communication. The nodes communicate using [RPC over UDP](#) to communicate, meaning that it is capable of working behind a [NAT](#).

This library aims to be as close to a reference implementation of the [Kademlia paper](#) as possible.



## INSTALLATION

The easiest (and best) way to install kademia is through `pip`:

```
$ pip install kademia
```





## USAGE

To start a new network, create the first node. Future nodes will connect to this first node (and any other nodes you know about) to create the network.

```
import logging
import asyncio

from kademia.network import Server

handler = logging.StreamHandler()
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
handler.setFormatter(formatter)
log = logging.getLogger('kademia')
log.addHandler(handler)
log.setLevel(logging.DEBUG)

loop = asyncio.get_event_loop()
loop.set_debug(True)

server = Server()
loop.run_until_complete(server.listen(8468))

try:
    loop.run_forever()
except KeyboardInterrupt:
    pass
finally:
    server.stop()
    loop.close()
```

Here's an example of bootstrapping a new node against a known node and then setting a value:

```
import logging
import asyncio
import sys

from kademia.network import Server

if len(sys.argv) != 5:
    print("Usage: python set.py <bootstrap node> <bootstrap port> <key> <value>")
    sys.exit(1)

handler = logging.StreamHandler()
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
```

(continues on next page)

(continued from previous page)

```
handler.setFormatter(formatter)
log = logging.getLogger('kademlia')
log.addHandler(handler)
log.setLevel(logging.DEBUG)

async def run():
    server = Server()
    await server.listen(8469)
    bootstrap_node = (sys.argv[1], int(sys.argv[2]))
    await server.bootstrap([bootstrap_node])
    await server.set(sys.argv[3], sys.argv[4])
    server.stop()

asyncio.run(run())
```

---

**Note:** You must have at least two nodes running to store values. If a node tries to store a value and there are no other nodes to provide redundancy, then it is an exception state.

---

## RUNNING TESTS

To run tests:

```
$ pip install -r dev-requirements.txt  
$ pytest
```



## **FIDELITY TO ORIGINAL PAPER**

The current implementation should be an accurate implementation of all aspects of the paper except one - in Section 2.3 there is the requirement that the original publisher of a key/value republish it every 24 hours. This library does not do this (though you can easily do this manually).



## QUERYING THE DHT

If you just want to query the network, you can use the example query script. For instance:

```
$ python examples/get.py 1.2.3.4 8468 SpecialKey
```

The query script is simple:

```
import logging
import asyncio
import sys

from kademia.network import Server

if len(sys.argv) != 4:
    print("Usage: python get.py <bootstrap node> <bootstrap port> <key>")
    sys.exit(1)

handler = logging.StreamHandler()
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
handler.setFormatter(formatter)
log = logging.getLogger('kademia')
log.addHandler(handler)
log.setLevel(logging.DEBUG)

async def run():
    server = Server()
    await server.listen(8469)
    bootstrap_node = (sys.argv[1], int(sys.argv[2]))
    await server.bootstrap([bootstrap_node])

    result = await server.get(sys.argv[3])
    print("Get result:", result)
    server.stop()

asyncio.run(run())
```

Check out the examples folder for other examples.





## KADEMLIA API

Kademlia is a Python implementation of the Kademlia protocol which utilizes the `asyncio` library. The best place to start is the `examples` folder before diving into the API.

### 6.1 kademlia package

The best place to start is the `examples` folder before diving into the API.

#### 6.1.1 kademlia.crawling module

**class** `kademlia.crawling.NodeSpiderCrawl` (*protocol, node, peers, ksize, alpha*)

Bases: `kademlia.crawling.SpiderCrawl`

Create a new `C{SpiderCrawl}`er.

##### Parameters

- **protocol** – A `KademliaProtocol` instance.
- **node** – A `Node` representing the key we’re looking for
- **peers** – A list of `Node` instances that provide the entry point for the network
- **ksize** – The value for `k` based on the paper
- **alpha** – The value for `alpha` based on the paper

##### **async find()**

Find the closest nodes.

**class** `kademlia.crawling.RPCFindResponse` (*response*)

Bases: `object`

A wrapper for the result of a RPC find.

**Parameters** **response** – This will be a tuple of (<response received>, <value>) where <value> will be a list of tuples if not found or a dictionary of {'value': v} where v is the value desired

##### **get\_node\_list()**

Get the node list in the response. If there’s no value, this should be set.

##### **get\_value()**

##### **happened()**

Did the other host actually respond?

`has_value()`

**class** `kademlia.crawling.SpiderCrawl` (*protocol, node, peers, ksize, alpha*)

Bases: `object`

Crawl the network and look for given 160-bit keys.

Create a new `C{SpiderCrawl}`er.

**Parameters**

- **protocol** – A `KademliaProtocol` instance.
- **node** – A `Node` representing the key we’re looking for
- **peers** – A list of `Node` instances that provide the entry point for the network
- **ksize** – The value for `k` based on the paper
- **alpha** – The value for `alpha` based on the paper

**class** `kademlia.crawling.ValueSpiderCrawl` (*protocol, node, peers, ksize, alpha*)

Bases: `kademlia.crawling.SpiderCrawl`

Create a new `C{SpiderCrawl}`er.

**Parameters**

- **protocol** – A `KademliaProtocol` instance.
- **node** – A `Node` representing the key we’re looking for
- **peers** – A list of `Node` instances that provide the entry point for the network
- **ksize** – The value for `k` based on the paper
- **alpha** – The value for `alpha` based on the paper

**async find()**

Find either the closest nodes or the value requested.

## 6.1.2 kademlia.network module

## 6.1.3 kademlia.node module

**class** `kademlia.node.Node` (*node\_id, ip=None, port=None*)

Bases: `object`

Simple object to encapsulate the concept of a Node (minimally an ID, but also possibly an IP and port if this represents a node on the network). This class should generally not be instantiated directly, as it is a low level construct mostly used by the router.

Create a Node instance.

**Parameters**

- **node\_id** (*int*) – A value between 0 and  $2^{160}$
- **ip** (*string*) – Optional IP address where this Node lives
- **port** (*int*) – Optional port for this Node (set when IP is set)

**distance\_to** (*node*)

Get the distance between this node and another.

**same\_home\_as** (*node*)

**class** `kademlia.node.NodeHeap` (*node*, *maxsize*)

Bases: `object`

A heap of nodes ordered by distance to a given node.

Constructor.

@param *node*: The node to measure all distances from. @param *maxsize*: The maximum size that this heap can grow to.

**get\_ids** ()

**get\_node** (*node\_id*)

**get\_uncontacted** ()

**have\_contacted\_all** ()

**mark\_contacted** (*node*)

**popleft** ()

**push** (*nodes*)

Push nodes onto heap.

@param *nodes*: This can be a single item or a C{list}.

**remove** (*peers*)

Remove a list of peer ids from this heap. Note that while this heap retains a constant visible size (based on the iterator), it's actual size may be quite a bit larger than what's exposed. Therefore, removal of nodes may not change the visible size as previously added nodes suddenly become visible.

## 6.1.4 kademlia.protocol module

## 6.1.5 kademlia.routing module

**class** `kademlia.routing.KBucket` (*rangeLower*, *rangeUpper*, *ksize*, *replacementNodeFactor*=5)

Bases: `object`

**add\_node** (*node*)

Add a C{Node} to the C{KBucket}. Return True if successful, False if the bucket is full.

If the bucket is full, keep track of node in a replacement list, per section 4.1 of the paper.

**depth** ()

**get\_nodes** ()

**has\_in\_range** (*node*)

**head** ()

**is\_new\_node** (*node*)

**remove\_node** (*node*)

**split** ()

**touch\_last\_updated** ()

**class** `kademlia.routing.RoutingTable` (*protocol*, *ksize*, *node*)

Bases: `object`

@param *node*: The node that represents this server. It won't be added to the routing table, but will be needed later to determine which buckets to split or not.

**add\_contact** (*node*)

**find\_neighbors** (*node*, *k=None*, *exclude=None*)

**flush** ()

**get\_bucket\_for** (*node*)

Get the index of the bucket that the given node would fall into.

**is\_new\_node** (*node*)

**lonely\_buckets** ()

Get all of the buckets that haven't been updated in over an hour.

**remove\_contact** (*node*)

**split\_bucket** (*index*)

**class** `kademlia.routing.TableTraverser` (*table*, *startNode*)

Bases: `object`

## 6.1.6 kademlia.storage module

**class** `kademlia.storage.ForgetfulStorage` (*ttl=604800*)

Bases: `kademlia.storage.IStorage`

By default, max age is a week.

**\_\_getitem\_\_** (*key*)

Get the given key. If item doesn't exist, raises `C{KeyError}`

**\_\_setitem\_\_** (*key*, *value*)

Set a key to the given value.

**cull** ()

**get** (*key*, *default=None*)

Get given key. If not found, return default.

**iter\_older\_than** (*seconds\_old*)

Return the an iterator over (key, value) tuples for items older than the given `secondsOld`.

**class** `kademlia.storage.IStorage`

Bases: `abc.ABC`

Local storage for this node. `IStorage` implementations of `get` must return the same type as put in by `set`

**abstract** **\_\_getitem\_\_** (*key*)

Get the given key. If item doesn't exist, raises `C{KeyError}`

**abstract** **\_\_setitem\_\_** (*key*, *value*)

Set a key to the given value.

**abstract** **get** (*key*, *default=None*)

Get given key. If not found, return default.

**abstract** **iter\_older\_than** (*seconds\_old*)

Return the an iterator over (key, value) tuples for items older than the given `secondsOld`.

### 6.1.7 kademlia.utils module

General catchall for functions that don't make sense as methods.

`kademlia.utils.bytes_to_bit_string` (*bites*)

`kademlia.utils.digest` (*string*)

**async** `kademlia.utils.gather_dict` (*dic*)

`kademlia.utils.shared_prefix` (*args*)

Find the shared prefix between the strings.

For instance:

```
sharedPrefix(['blahblah', 'blahwhat'])
```

returns 'blah'.



## INDICES AND TABLES

- genindex
- modindex
- search





## PYTHON MODULE INDEX

### k

`kademlia`, 13

`kademlia.crawling`, 13

`kademlia.node`, 14

`kademlia.routing`, 15

`kademlia.storage`, 16

`kademlia.utils`, 17



## Symbols

`__getitem__()` (*kademlia.storage.ForgetfulStorage method*), 16  
`__getitem__()` (*kademlia.storage.IStorage method*), 16  
`__setitem__()` (*kademlia.storage.ForgetfulStorage method*), 16  
`__setitem__()` (*kademlia.storage.IStorage method*), 16

## A

`add_contact()` (*kademlia.routing.RoutingTable method*), 15  
`add_node()` (*kademlia.routing.KBucket method*), 15

## B

`bytes_to_bit_string()` (*in module kademlia.utils*), 17

## C

`cull()` (*kademlia.storage.ForgetfulStorage method*), 16

## D

`depth()` (*kademlia.routing.KBucket method*), 15  
`digest()` (*in module kademlia.utils*), 17  
`distance_to()` (*kademlia.node.Node method*), 14

## F

`find()` (*kademlia.crawling.NodeSpiderCrawl method*), 13  
`find()` (*kademlia.crawling.ValueSpiderCrawl method*), 14  
`find_neighbors()` (*kademlia.routing.RoutingTable method*), 16  
`flush()` (*kademlia.routing.RoutingTable method*), 16  
`ForgetfulStorage` (*class in kademlia.storage*), 16

## G

`gather_dict()` (*in module kademlia.utils*), 17  
`get()` (*kademlia.storage.ForgetfulStorage method*), 16  
`get()` (*kademlia.storage.IStorage method*), 16

`get_bucket_for()` (*kademlia.routing.RoutingTable method*), 16  
`get_ids()` (*kademlia.node.NodeHeap method*), 15  
`get_node()` (*kademlia.node.NodeHeap method*), 15  
`get_node_list()` (*kademlia.crawling.RPCFindResponse method*), 13  
`get_nodes()` (*kademlia.routing.KBucket method*), 15  
`get_uncontacted()` (*kademlia.node.NodeHeap method*), 15  
`get_value()` (*kademlia.crawling.RPCFindResponse method*), 13

## H

`happened()` (*kademlia.crawling.RPCFindResponse method*), 13  
`has_in_range()` (*kademlia.routing.KBucket method*), 15  
`has_value()` (*kademlia.crawling.RPCFindResponse method*), 13  
`have_contacted_all()` (*kademlia.node.NodeHeap method*), 15  
`head()` (*kademlia.routing.KBucket method*), 15

## I

`is_new_node()` (*kademlia.routing.KBucket method*), 15  
`is_new_node()` (*kademlia.routing.RoutingTable method*), 16  
`IStorage` (*class in kademlia.storage*), 16  
`iter_older_than()` (*kademlia.storage.ForgetfulStorage method*), 16  
`iter_older_than()` (*kademlia.storage.IStorage method*), 16

## K

`kademlia`  
 module, 13  
`kademlia.crawling`  
 module, 13  
`kademlia.node`  
 module, 14

kademlia.routing  
  module, 15  
kademlia.storage  
  module, 16  
kademlia.utils  
  module, 17  
KBucket (*class in kademlia.routing*), 15

## L

lonely\_buckets() (*kademlia.routing.RoutingTable method*), 16

## M

mark\_contacted() (*kademlia.node.NodeHeap method*), 15  
module  
  kademlia, 13  
  kademlia.crawling, 13  
  kademlia.node, 14  
  kademlia.routing, 15  
  kademlia.storage, 16  
  kademlia.utils, 17

## N

Node (*class in kademlia.node*), 14  
NodeHeap (*class in kademlia.node*), 14  
NodeSpiderCrawl (*class in kademlia.crawling*), 13

## P

popleft() (*kademlia.node.NodeHeap method*), 15  
push() (*kademlia.node.NodeHeap method*), 15

## R

remove() (*kademlia.node.NodeHeap method*), 15  
remove\_contact() (*kademlia.routing.RoutingTable method*), 16  
remove\_node() (*kademlia.routing.KBucket method*), 15  
RoutingTable (*class in kademlia.routing*), 15  
RPCFindResponse (*class in kademlia.crawling*), 13

## S

same\_home\_as() (*kademlia.node.Node method*), 14  
shared\_prefix() (*in module kademlia.utils*), 17  
SpiderCrawl (*class in kademlia.crawling*), 14  
split() (*kademlia.routing.KBucket method*), 15  
split\_bucket() (*kademlia.routing.RoutingTable method*), 16

## T

TableTraverser (*class in kademlia.routing*), 16  
touch\_last\_updated() (*kademlia.routing.KBucket method*), 15

## V

ValueSpiderCrawl (*class in kademlia.crawling*), 14