FACULTY OF ENGINEERING AND ARCHITECTURE

PEER-TO-PEER SYSTEMS

Chapter 8: Parallel and Distributed Software Systems

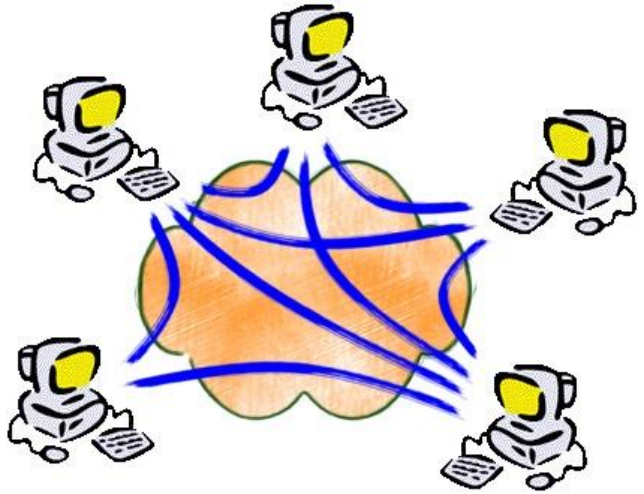
GHENT UNIVERSITY

OUTLINE

8.1 Introduction

8.2 Overlays

8.3 Distributed Hash Tables



GHENT UNIVERSITY

OUTLINE

1. Introduction
    1. Why P2P systems ?
    2. P2P generations
2. Overlays
3. Distributed Hash Tables

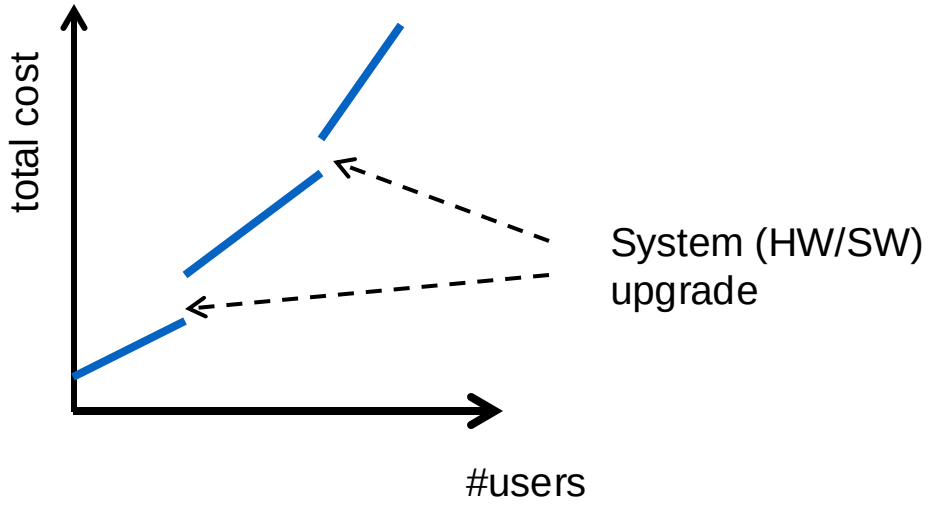
Peer-to-Peer systems versus Client-Server

GHENT UNIVERSITY

THE RATIONALE FOR P2P

Scaling and cost of client server systems

popular services need large server infrastructures
-> high investment and operational cost



Observation

- performance of edge devices grows (gap client – server closes)
- performance only occasionally fully used

GHENT UNIVERSITY

THE RATIONALE FOR P2P

"How to unleash the power of Internet's dark matter ?"

P2P philosophy

make heavily use of edge resources
(CPU, storage, I/O devices, Information, ...)

#users ~ #edge devices
-> infrastructure grows together
(automatically) with user base

Need for software systems that can survive without central servers.

WHY IS P2P DIFFICULT ?

Nodes
- large number of nodes
- unstable
- under control of end users

Interconnection infrastructure
- slow
- unreliable

No central control
- more complex managment (control infrastructure can fail)

=>

- decentralized control
- self-organization
- handling failing nodes
- spreading load dynamically

GHENT UNIVERSITY

THE HISTORY OF P2P

Generation I : File sharing with centralized control

- index maintained in a centralized infrastructure
- download purely P2P
  -> poor scalability
  -> vulnerable to attacks

Generation II : File sharing with decentralized control

- self-organization through overlay construction
- two variants:
- pure P2P (all nodes identical)
- hybrid P2P (hierarchical structuring)
- scalable and robust

Structured vs. Unstructured P2P Systems

Generation III : P2P middleware

- middleware services offered:
- data placement
- data lookup
- automatic replication/caching
- authentication/security
- applications built on top of P2P middleware

GHENT UNIVERSITY

OUTLINE

1. Introduction
2. Overlays
3. Distributed Hash Tables

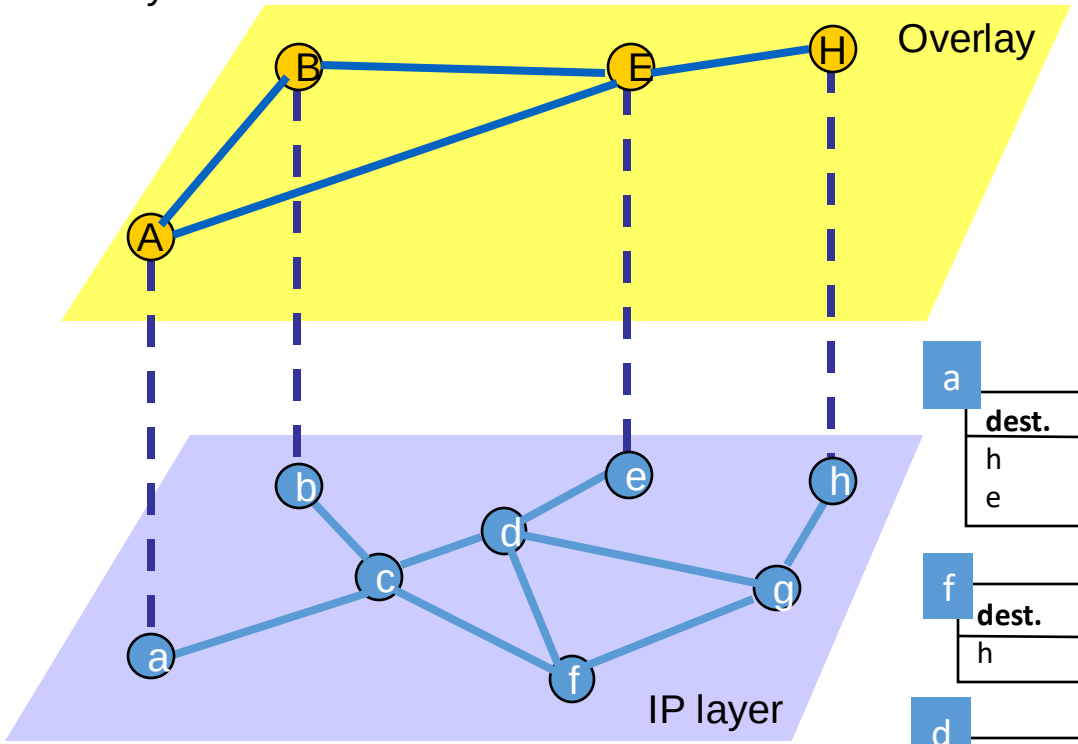
GHENT UNIVERSITY

OVERLAYS

GHENT UNIVERSITY

OVERLAY ROUTING

Full mesh overlay topology
scales badly !



A

dest.	next hop
H	B
E	E

B

dest.	next hop
H	E

E

dest.	next hop
H	H

a

dest.	next
h	c
e	c

c

dest.	next
h	f
e	d

f

dest.	next
h	g

g

dest.	next
h	h

d

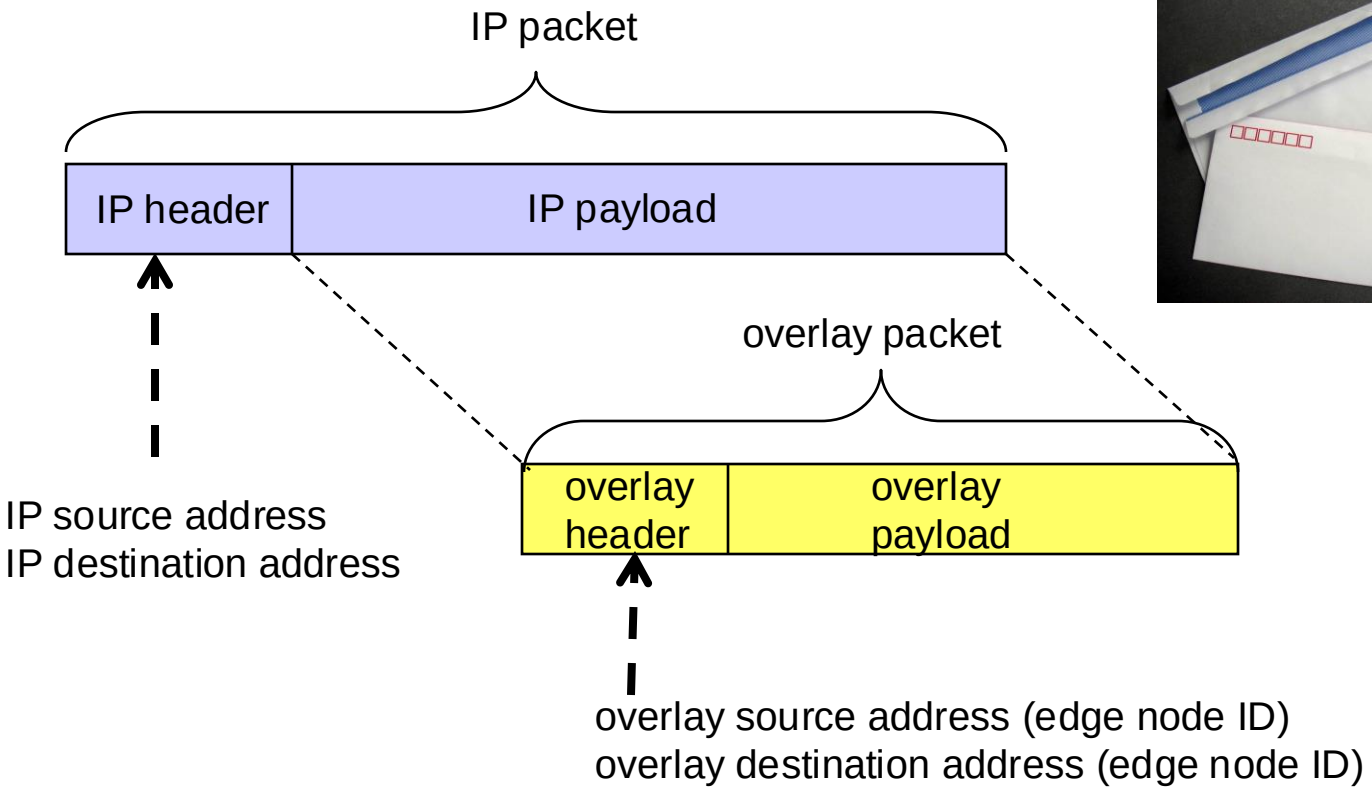
dest.	next
e	e

b

dest.	next
e	c

GHENT UNIVERSITY

OVERLAY ROUTING



IP layer and overlay layer have own
- addressing scheme
- routing protocol

CONTENT BASED ROUTING

Use ID of object to manipulate instead of node ID

-> overlay can redirect to closest replica

-> overlay can optimize placement and #replica's

Globally Unique ID (GUID)

- constructed through hash of object state
- content itself
- object description
- overlay = distributed hash table (DHT)

Responsibilities of DHT

- find object given GUID (route requests)

-> map GUID to node ID

- announce GUIDs of new objects
- remove GUIDs of old objects
- manage with nodes becoming unavailable

```
ff478f2d-e398-449a-93d4-62b0727
adffed61-73f9-42a1-8d54-90f3537
ec359ec3-99be-4c44-8449-be7alab
9349e44d-5367-4b20-b6b1-d421c5a
54fbb946-f9e3-498f-b04c-e91584d
c8d1308c-76d2-452e-8d80-6d08483
f1a841a3-b456-4e92-99b5-04f3d86
bala4110-01cb-4a4a-967e-b8adbf5
63d06b7a-8cf5-46d5-bcdd-ffd3be7
4ffa8171-78e0-455a-9cbe-e9fee93
```

GHENT UNIVERSITY

DHT ROUTING VS. IP ROUTING

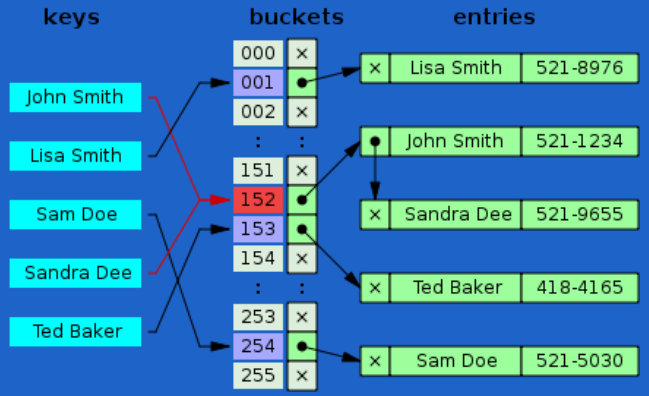
	IP routing	DHT routing
Size	IPv4 : 2^{32} nodes IPv6 : 2^{128} nodes but: hierarchical structured	> 2^{128} GUIDs flat
Load balancing	routes determined by topology (e.g. OSPF routes)	any algorithm can be used objects can be relocated to optimize routing
Dynamics	static (time constant 1h) asynchronous w.r.t. application	can be dynamic synchronous or asynchronous
Resilience	built in	ensured through replication of objects
Target	IP destination maps to 1 node	GUID can map to different replicas

OUTLINE

1. **Introduction**
2. **Overlays**
3. **Distributed Hash Tables**
    1. Introduction
    2. Circular routing (1D)
    3. Pastry (1D)
    4. Chord (1D)
    5. CAN (nD)

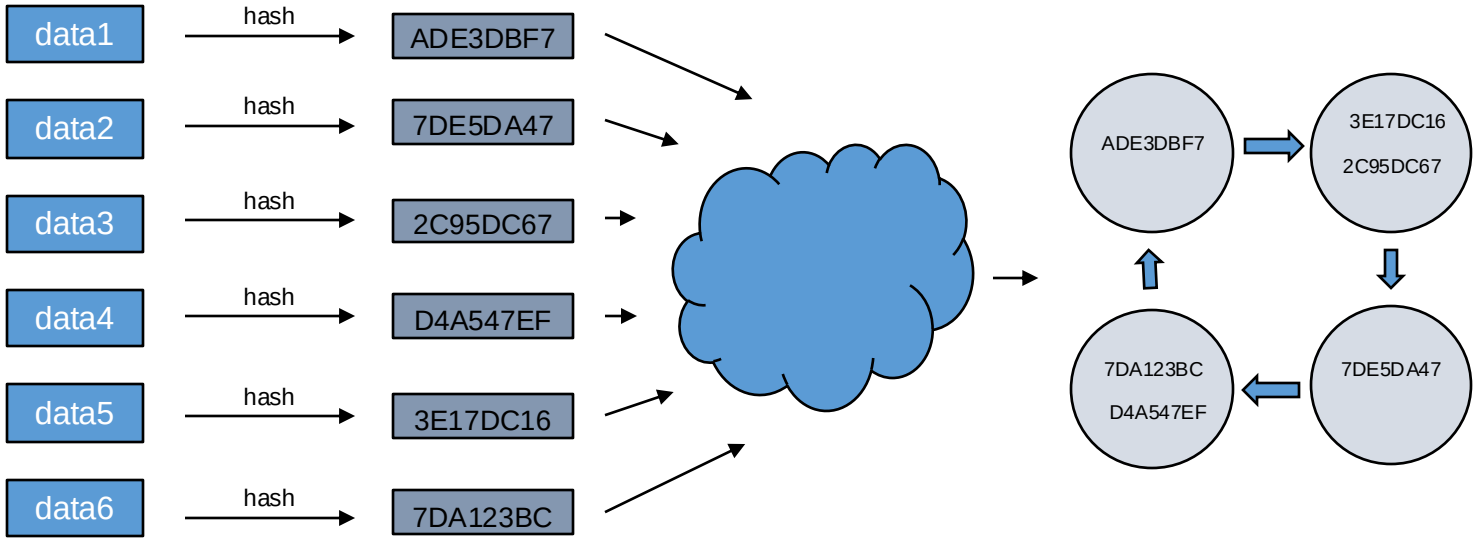
GHENT UNIVERSITY

DISTRIBUTED HASH TABLES



GHENT UNIVERSITY

DISTRIBUTED HASH TABLES



GHENT UNIVERSITY

BOOTSTRAPPING

The bootstrapping problem

how to find a network node ?

Approaches

- pre-configured static addresses of stable nodes
  - should have fixed IP address
  - should always be on
- DNS-service

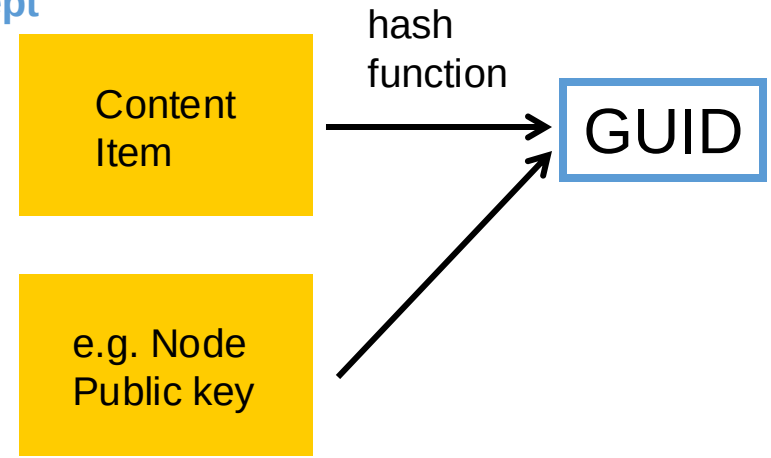
  domain name resolves to nodes

Configuration info made available

- routing table bootstrap info
- GUID space the node is responsible for
- protocol information

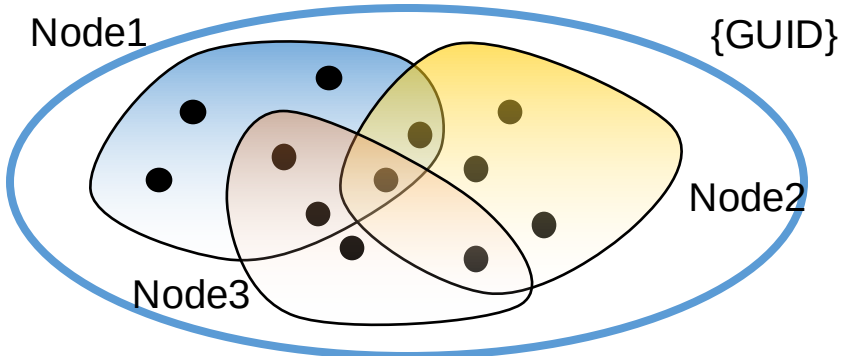
DHT : GUID ROUTING

Concept



Node is responsible for range of GUIDs it belongs to

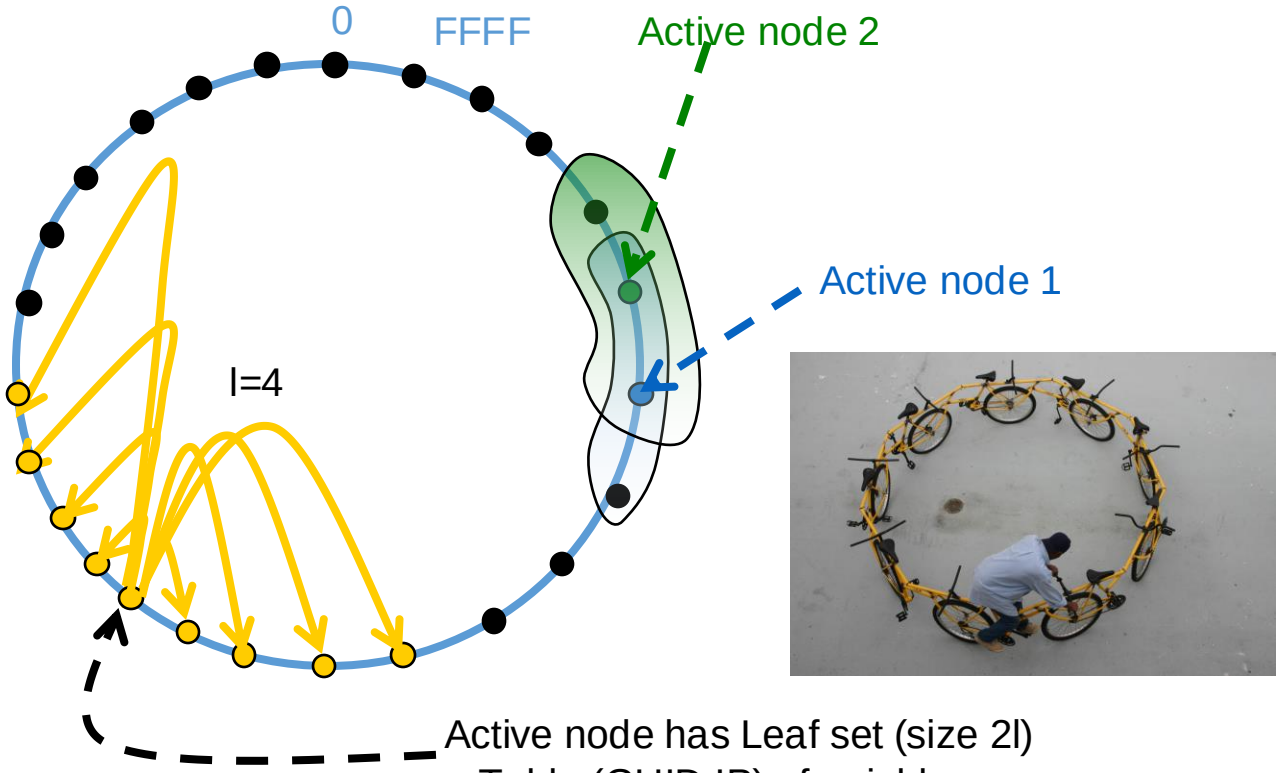
Nodes are responsible for part of GUID space



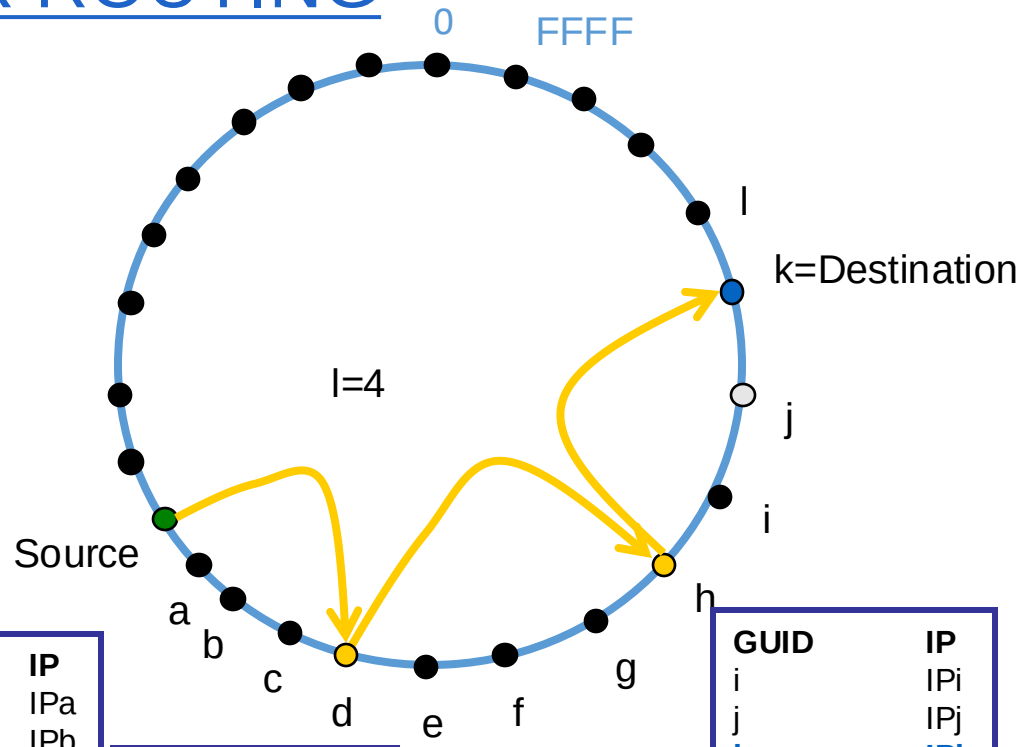
GUIDs in intersection are replicated

CIRCULAR ROUTING

Suppose : GUID consists of 4 hex symbols



CIRCULAR ROUTING



GUID	IP
a	IPa
b	IPb
c	IPc
d	**IPd**

GUID	IP
e	IPe
f	IPf
g	IPg
h	**IPh**

GUID	IP
i	IPi
j	IPj
k	**IPk**
l	IPl

N active nodes
leaf set size = 2l

=>

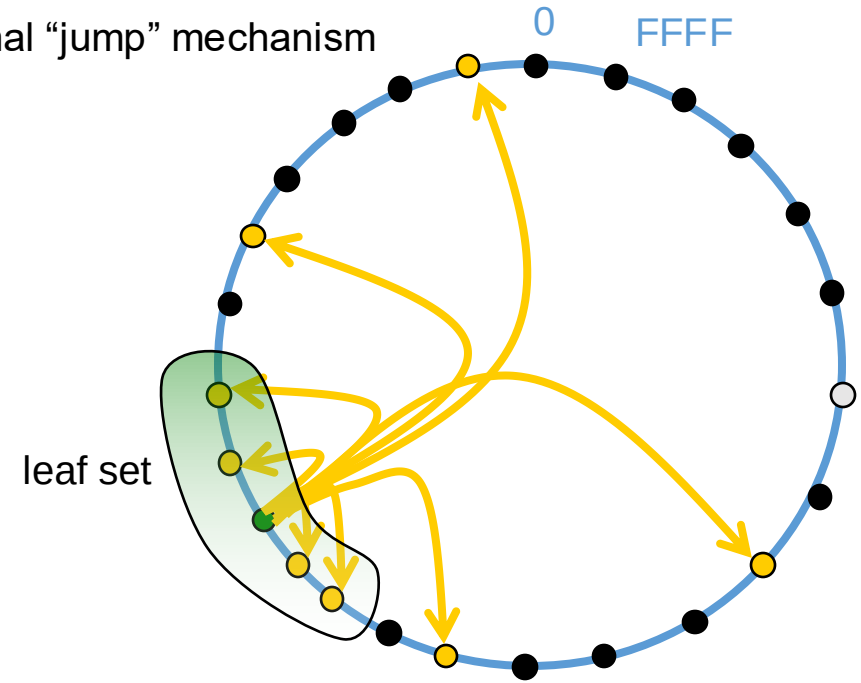
worst case
N/2l hops

GHENT UNIVERSITY

PREFIX ROUTING

Optimize routing efficiency
- allow for longer jumps
- without increasing leaf set

=> need for additional “jump” mechanism



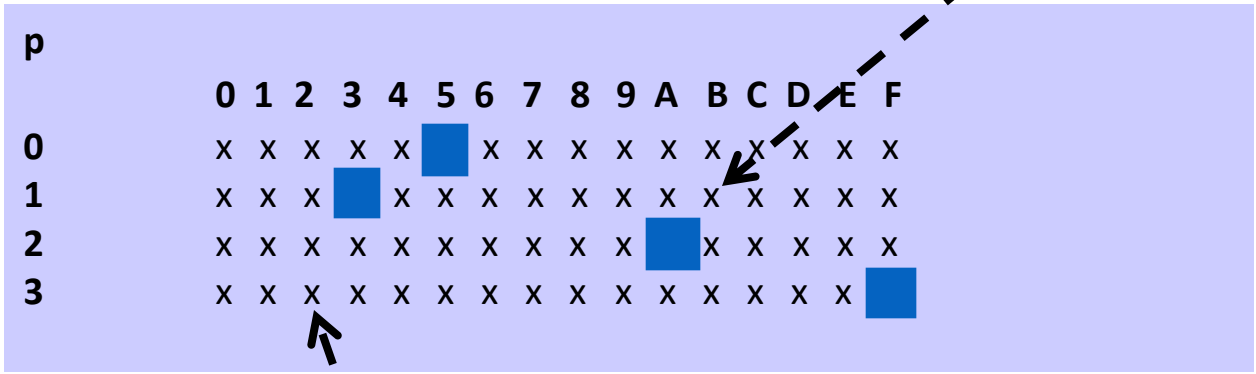
PREFIX ROUTING

Routing table structure @active node

n-bit GUID -> n/4 routing table rows
(16 bit -> 4 rows)

p = longest prefix match
Routing table of node with GUID 53AF

[GUID,IP] of next hop for message destinated for 5Cxx



[GUID,IP] of next hop for message destinated for 53A3 (=closest active node to 53A3)

GHENT UNIVERSITY

PREFIX ROUTING

Message m from S(ource) to D(estination) arrives in C(urrent) node
Leaf set : size 2l (L_{-l} -> L_l)
R : routing matrix

```
if (L_-l ≤ D ≤ L_l) {
        forward m
                - to GUID_i found in leaf set
                - to current node C
} else {
        find longest prefix match p of D and C
        c = symbol (p+1) of D
        if(R_pc ≠ null) forward m to R_pc
        else {
                find GUID_i in L or R with |GUID_i – D|<|GUID_i-C|
                forward m to GUID_i
        }

}
```

Observation:

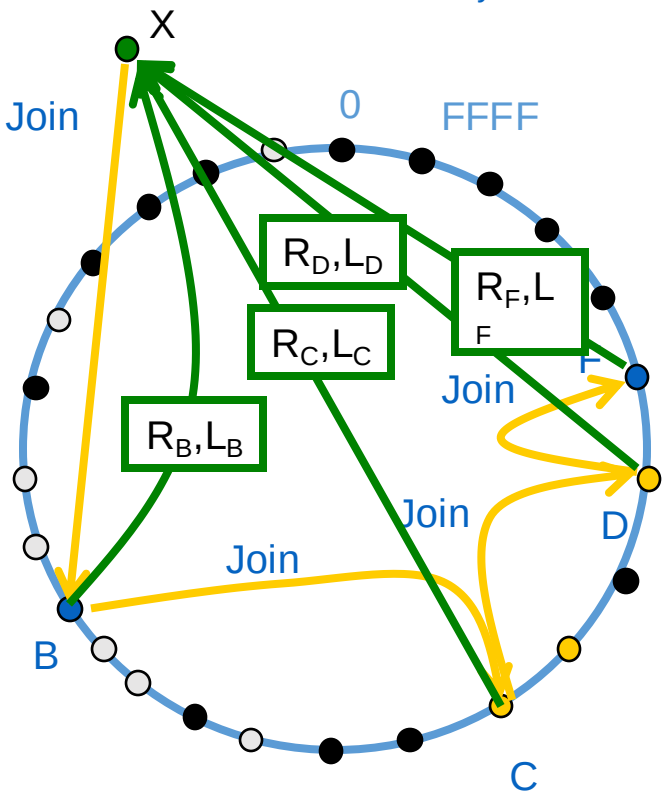
correct routing possible with empty routing table

NODE DYNAMICS : JOINING

X wants to join
- how to construct R_X, L_X ?
- how to adapt R of all other nodes ?

Special “nearest neighbour” discovery algorithm to find a nearby active node B

1. X sends join(X,$GUID_X$) to B
2. DHT routes join the usual way to node F
   min |$GUID_F$-$GUID_X$|
3. All nodes on routing path send info to X to construct R_X,L_X



How to efficiently build the routing table?

NODE DYNAMICS : JOINING

Constructing R_X

$\{B_0, B_1, B_2, \ldots, B_{N-1}\}$ = set of physical nodes visited by Join message

B_0=B

B_{N-1}=F

Row 0 :

- used to route GUID with no common prefix
- bootstrap node B_0 (very) close to X

=> $R_X[0]=R_{B0}[0]$

Row 1 :

- used to route GUID with common prefix 1
- prefix($GUID_{B1}$,$GUID_X$)≥1

=> $R_X[1]=R_{B1}[1]$

...

Row i : => $R_X[i]=R_{Bi}[i]$

Constructing L_X

X should be neighbour of F

initial choice : $L_X=L_F$

GHENT UNIVERSITY

NODE DYNAMICS : LEAVING

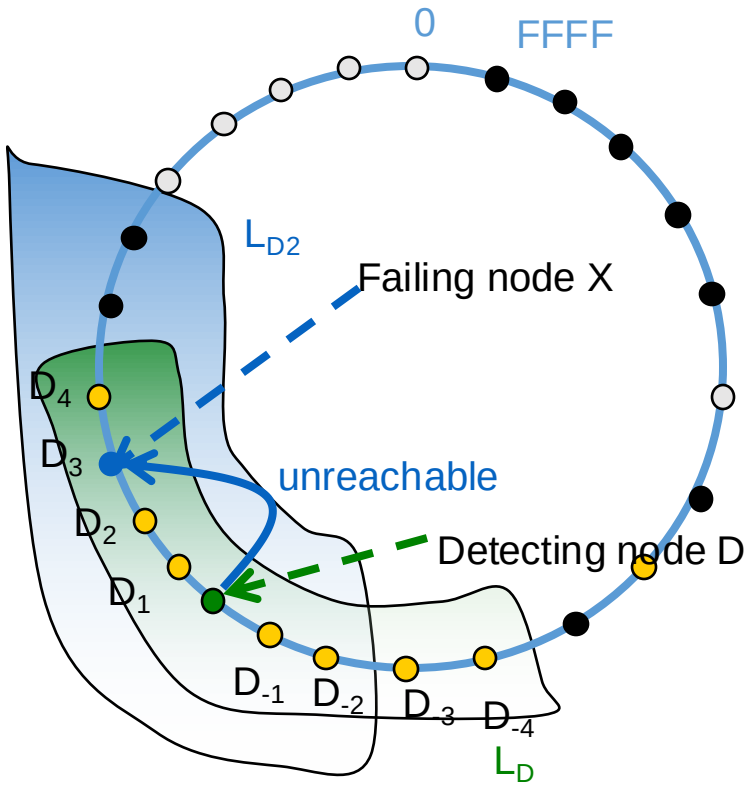
Only leaf sets are repaired

l=4

L_D

GUID(D_{-4}) IP(D_{-4})
GUID(D_{-3}) IP(D_{-3})
GUID(D_{-2}) IP(D_{-2})
GUID(D_{-1}) IP(D_{-1})
GUID(D_1) IP(D_1)
GUID(D_2) IP(D_2)
GUID(D_3) IP(D_3)
GUID(D_4) IP(D_4)

1. D finds node close to $GUID_X$ -> D_2
2. Get L_{D2}
3. Adapt L_D based on
   -> Remove D_3
   -> Add D_5
4. Propagate to neighbours



GHENT UNIVERSITY

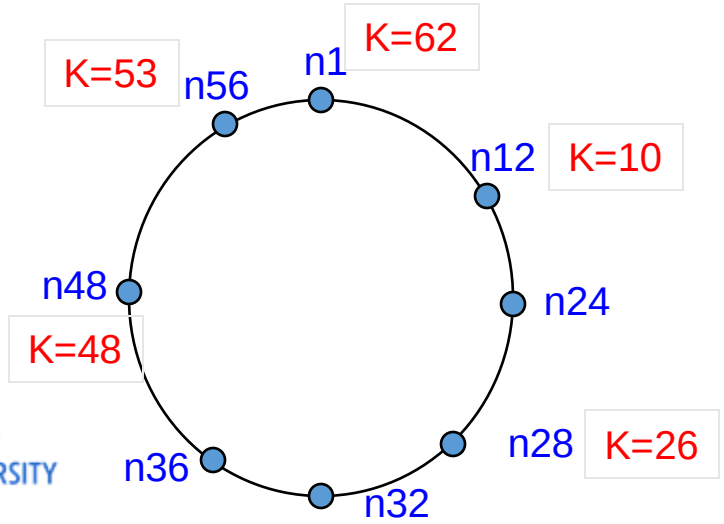
CONSISTENT HASHING

Hash function (SHA-1) produces m-bit identifiers

hash(nodeIP)
hash(key)

Identifiers mapped to identifier circle modulo 2^m

Key k -> hash(k) -> node(hash(k)) = successor(k)
= node with smallest identifier ≥ hash(k)



m = 6
8 nodes
5 keys stored

GHENT UNIVERSITY

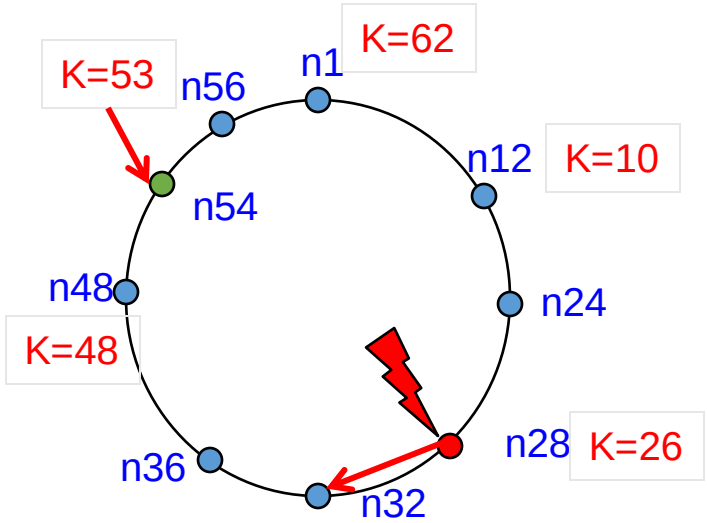
JOINING AND LEAVING

X Leaving

keys associated with X -> reassigned to successor(X)

X Joining

X gets some of the keys assigned to successor(X)



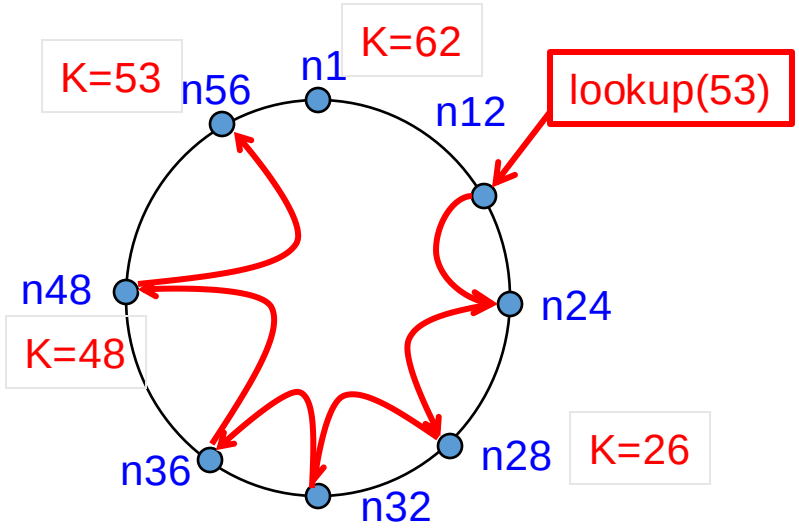
GHENT UNIVERSITY

SIMPLE ROUTING

Request lookup(id) arrives at node n

Node n knows id of its successor

```
n.findSuccessor(id) {
        if id in (n,successor]  return successor
        else return successor.findSuccessor(id)
}
```



(a,b]
Clockwise interval
Excluding a, including b

MULTI-DIMENSIONAL ROUTING

CAN : Content addressable network

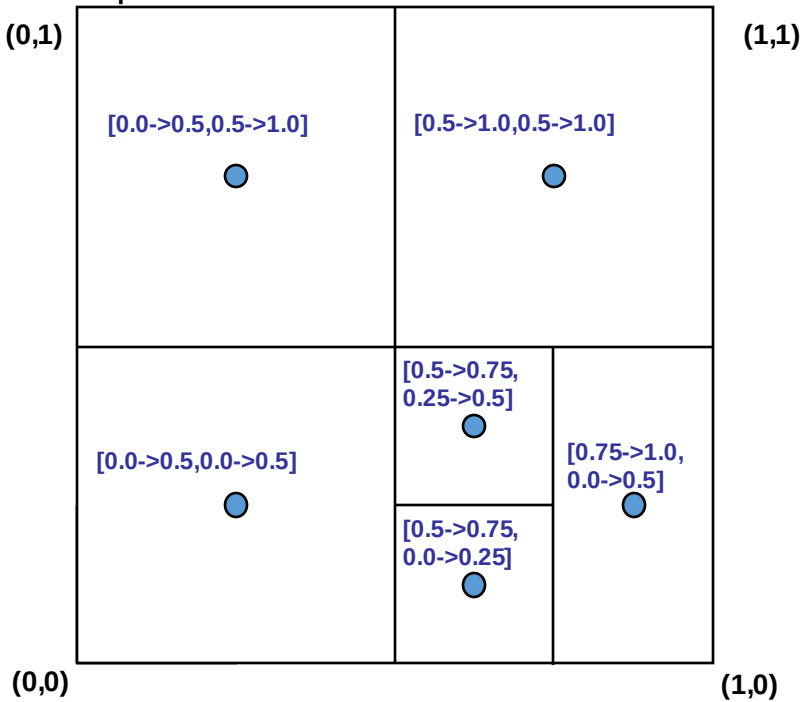
Basic idea

use d-dimensional space to map keys

node is responsible for d-dimensional hypercube

- Cartesian coordinates used
- space is d-torus

Example 2D



GHENT UNIVERSITY

MULTI-DIMENSIONAL ROUTING

Mapping content to node

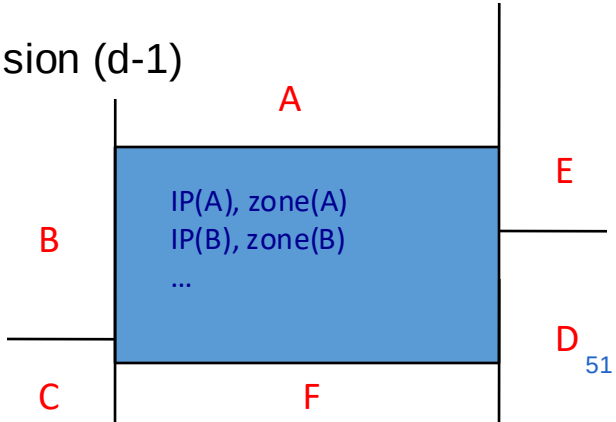
- Key-Value pair <K,V> mapped to point P in Cartesian space
  P=hash(K)
- P belongs to region owned by node N
- N stores <K,V>

Retrieving entry for key K

- compute P=hash(K)
- if P not owned by requester or neighbours
  -> route request in the CAN network

Self-organizing routing

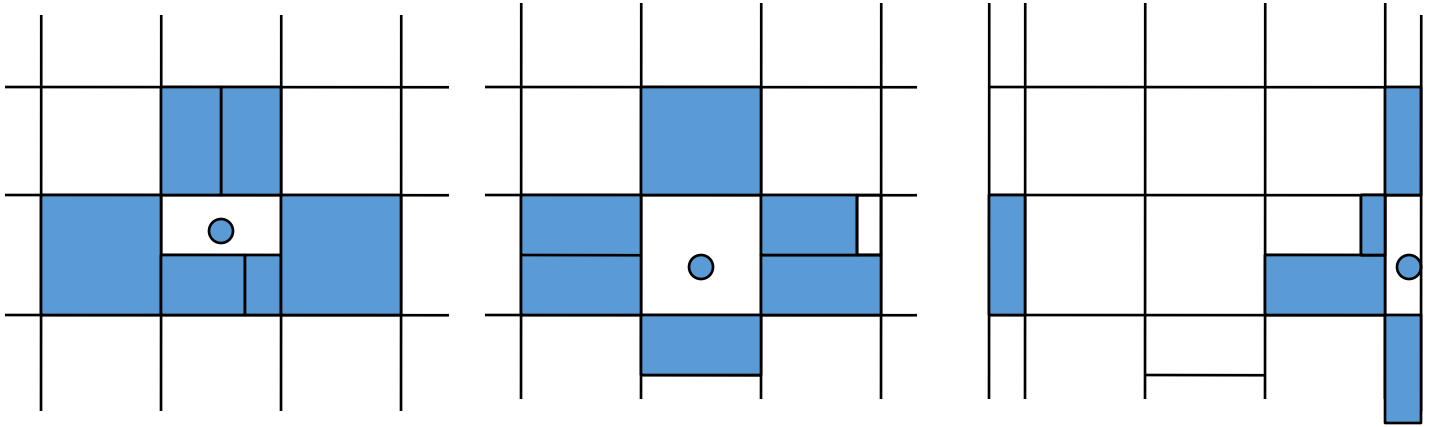
node learns and stores set of neighbours
neighbour : shares hyperplane of dimension (d-1)
-> in (d-1) dimension, intervals overlap
info stored per neighbour n
IP(n), zone(n)



GHENT UNIVERSITY

MULTI-DIMENSIONAL ROUTING

Neighbour nodes : examples



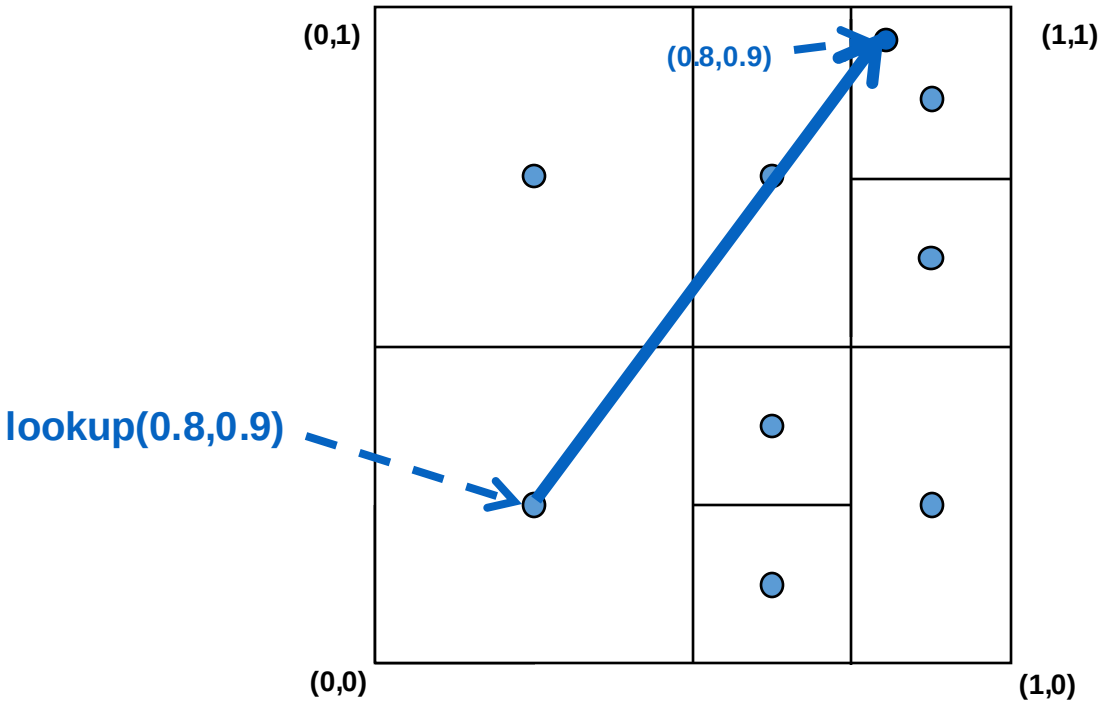
Suppose equally partitioned zones (regular d-dimensional grid)
-Number of neighbours : 2d
-Average routing path length = $(d\, n^{1/d})/4$

GHENT UNIVERSITY

MULTI-DIMENSIONAL ROUTING

Routing in d dimensions

Follow straight line from source to destination



GHENT UNIVERSITY

MULTI-DIMENSIONAL ROUTING

Routing in d dimensions

Forward to closest neighbour (greedy forwarding)
-> each node keeps track of at least d neighbours
-> space complexity O(d)
-> lookup cost $O(dN^{1/d})$ (N number of nodes)