HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

Winter semester 2014

Jun.-Prof. Dr.-Ing. Kalman Graffi

Heinrich Heine University Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

Structured P2P Overlays

- Kademlia

Interconnection Overlay Networks

- Hypercube
- De Bruijn Network

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

Structured Homogenous P2P Overlay Networks – Kademlia

This slide set is based on the lecture "Communication Networks 2" of Prof. Dr.-Ing. Ralf Steinmetz at TU Darmstadt

Kademlia

"Kademlia" vs. "Kad":

- Kademlia: Algorithm presented by Maymounkov and Mazières
- Kad: Implementation of the Kademlia algorithm used by eMule
- BitTorrent clients also implement Kademlia for their "trackerless mode"
  - Not compatible with Kad

Uses 160-bit-IDs

Uses XOR metric to determine distance between IDs

- XOR = bitwise eXclusive OR. Example:
- 10101010 = 170
- XOR 10010101 = 149
- = 00111111 = 63

XOR metric is symmetric: A XOR B == B XOR A

- Contrast to Chord

Kademlia Protocol

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

The Kademlia protocol consists of 4 RPCs:

- FIND_NODE(KEY):
  - Recipient returns <IP Address, UDP Port, Node ID> triples for k closest nodes he knows about
- FIND_VALUE(KEY):
  - Like FIND_NODE
  - With exception:
    - If recipient already stores the value, the value is returned instead of k closest nodes
- PING(IP ADDRESS)
  - Probes a node to see if it is online
- STORE(KEY, VALUE)
  - Instructs a node to store a <key,value> pair

Node state

Each node maintains routing table (k buckets)

- Routing tables of different peers may be different

For each 0 <= i < 160 every node

- keeps a list of <IP Address, UDP Port, Node ID> triples
- for k nodes within distance [2 ^i ; 2 ^(i+1) [→ routing bucket I
  - Distance of 2^i is the same as common prefix of i-1 bits
- in total k * 160 contacts
- often: tree presentation of routing buckets / other prefix-subtrees

Nodes learn from

- messages they receive or
- using the FIND_NODE method

Preference towards old contacts

- Study has shown that the longer a node has been up, the more likely it is to remain up another hour
- Resistance against DoS attacks by flooding the network with new nodes

k Buckets: Peer Selection Policy

If node u receives a message from node v
then it adds node v to its k-bucket
according to the following rules:

- IF v is already in a k-bucket
  THEN move v to the tail of the bucket
- IF v is not in the k-bucket and the bucket has fewer than k entries
  THEN insert recipient to the tail of list
- IF the appropriate k bucket is full AND least recently seen node is alive
  THEN move least recently seen node to tail of bucket and put node v
  to waiting list for the bucket
- IF the appropriate k bucket is full AND least recently seen node is not alive
  THEN remove least recently seen node from bucket and add node v at the tail

Note: approx. k = 20 in the real world

- Connection to nodes in buckets are NOT maintained

Presentation of k buckets

One presentation

Bucket for 2^159 – 2^160 Shared prefix = 0 bits

1	2	…	k	Waitinglist

Bucket for 2^158 – 2^159 Shared prefix = 1 bit

1	2	…	k	Waitinglist

…

Bucket for 2^0 – 2^1 Shared prefix = 159 bits

1	2	…	k	Waitinglist

Routing in Kademlia

The lookup initiator sends a FIND_NODE request
to α nodes closest to target ID

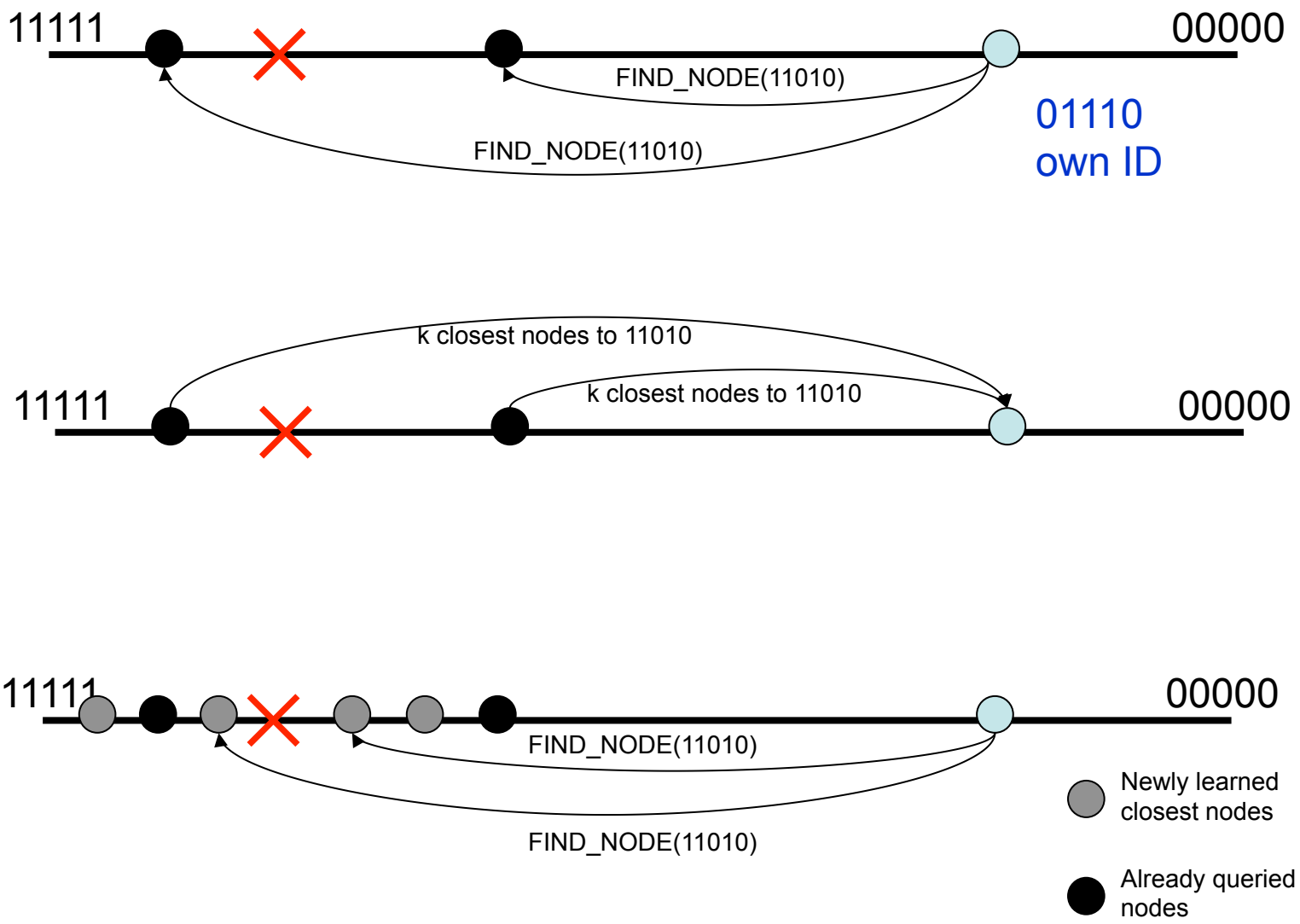
- α (alpha) is the parameter for the number of parallel lookups
- Parallel asynchronous queries
  - Overcome faulty nodes

Receiving nodes respond with k nodes closest to target ID

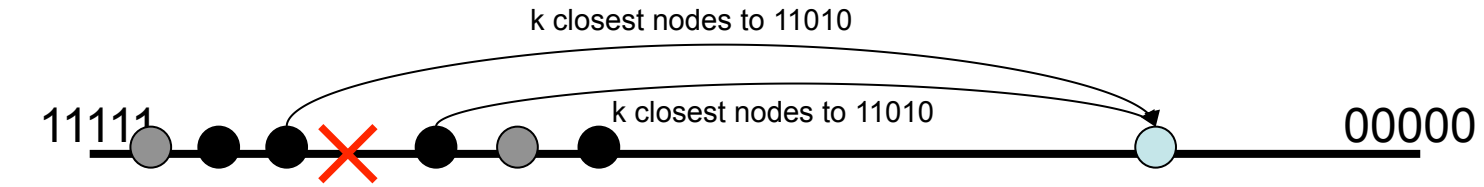
Lookup initiator resends FIND_NODE request
to α nodes learned from previous requests

Procedure stops
if the set of k closest nodes does not change anymore

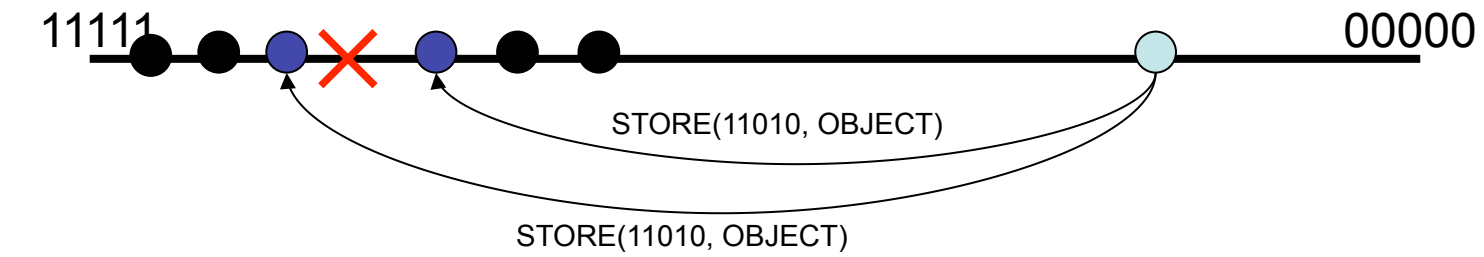
Routing from 00010 to 11010, α = 2, k = 2



Routing from 00010 to 11010, α = 2, k = 2



No new nodes discovered → stop lookup process



Newly learned closest nodes

Already queried nodes

k closest nodes

Kademlia: Enhancements

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Parallel queries

- For one query, α (alpha) concurrent lookups are sent
- More traffic load, but lower response times

Network maintenance

- In Chord: active fixing of fingers
- In Kademlia: learning for bypassing queries
- Check if peer IDs fit better in routing table

Large routing tables

- In Chord: 1 finger per distance 2^i to 2^(i+1)
- In Kademlia: k contacts per distance 2^i to 2^(i+1)
- Increased robustness

Kademlia Limitations

Kademlia functions

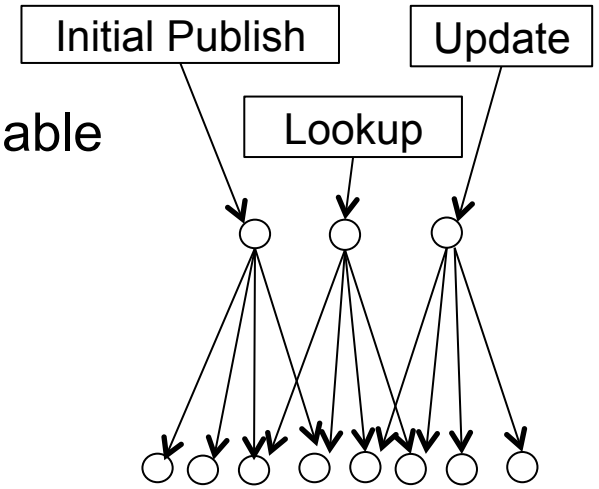
- Publish (key, value) pair ONCE
- Retrieve value for given key

Different than goals than Pasty / Key-based Route

- Pastry: Find (single) responsible peer for given key
- Pastry: Allows update of data!

Kademlia limitations

- Only 18% of nodes storing key/value reachable
- No updates possible (replication difficult)
- Key/values to be refreshed every 24h



Kad – Introduction

Kad is the Kademlia implementation of the file sharing applications eMule/aMule

Uses UDP for communication

Has been developed …

- … to improve source lookup and
- … to become independent from servers

Kad can be used …

- … to find sources for files and
- … to perform keyword searches

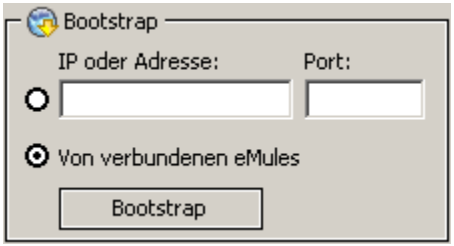
Differences to Kademlia

- ID length
  - 128 bits due to MD4 hashing of eDonkey network (usually displayed in hex)
- Lookup
  - Two phases (1. Node lookup; 2. Action, e.g. source request)
  - Does not terminate when one result is found
- Variable k: k = 10 (α is still 3)

Kad – Bootstrap process

Bootstrap

- A node can bootstrap using …
  - … a node known via the eDonkey network
  - … a user-provided host that is already part of the Kad network
  - … a downloaded hosts.dat file



Kad – Publication and search

Source publication

- A client obtains IDs for a file by hashing
  - The file content → 1st level pointer
  - Each part of the file name → 2nd level pointer
- And announces the file at peers responsible for the IDs

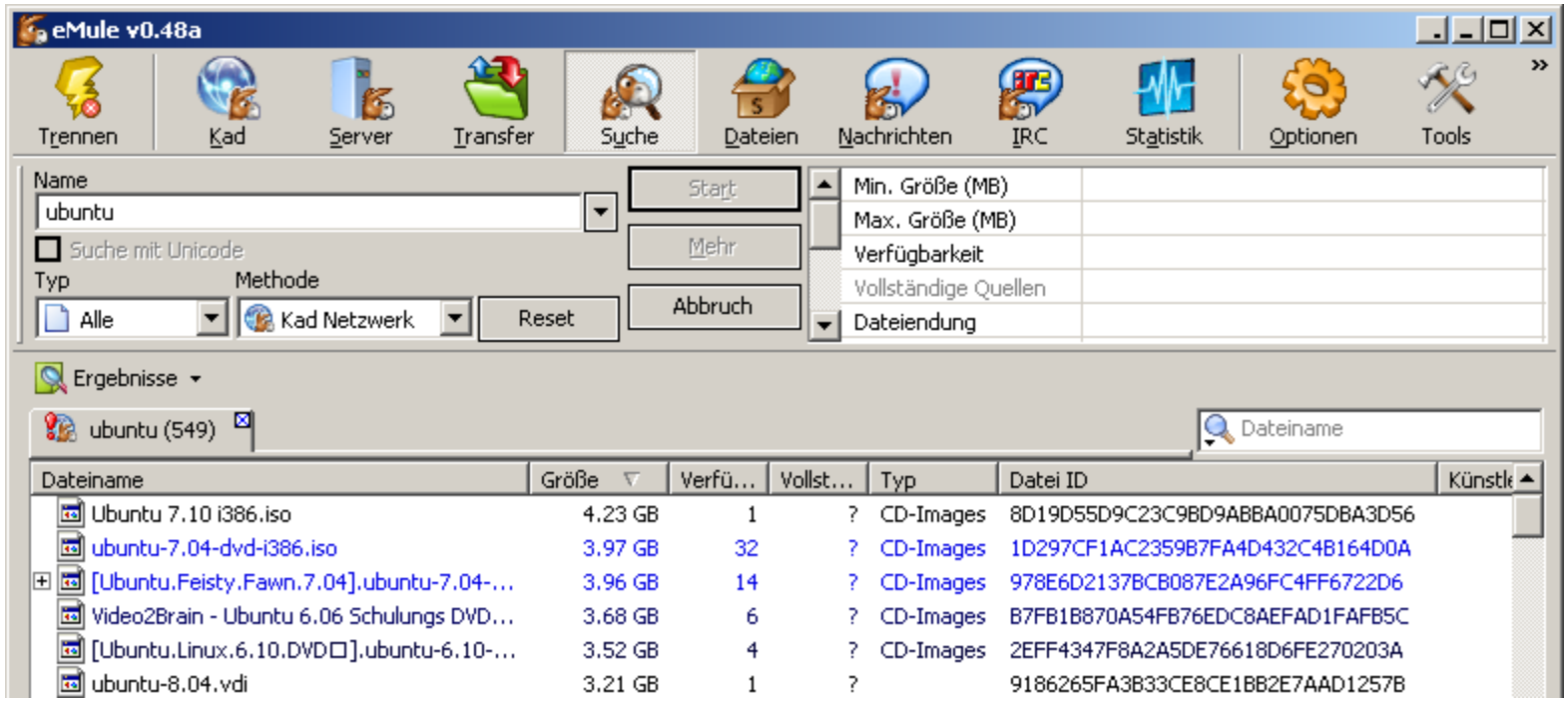
Source search

- If ID of a file is known
- Kad can be used to search for peers that offer that file

Keyword search

- Truncate search term at space characters → string tokens
- Hash tokens, lookup for token ID → obtain fitting file names
- Return (IDs, filenames)
- User chooses at GUI which files to download

Kad – Keyword search example



Kad – Lookup procedure

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

At the beginning of a lookup procedure

- Searching node creates a search list
  - containing the k (= 10!) closest nodes to the target ID it knows about
- The list is ordered by XOR distance (closest at the top)

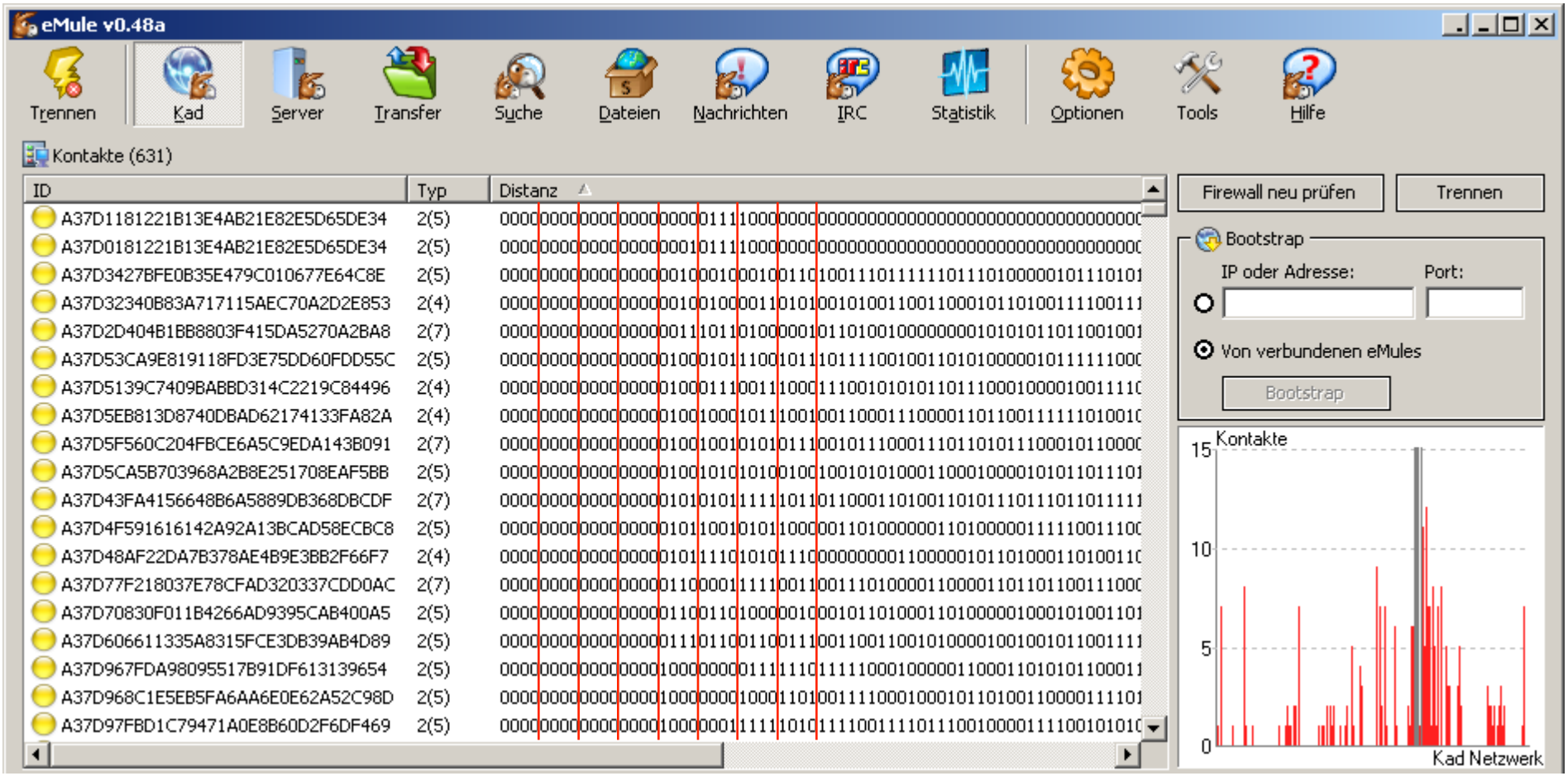
Lookup procedure (repeat until closest k nodes do not change):

- Node picks α (= 3) random nodes from the search list
- Searching node sends a REQUEST message to them
  - Asks for closer nodes
- The queried nodes answer
  - with nodes they know about which are closer to the target than themselves
- Those nodes are inserted into the search list.

When the closest k nodes have been found:

- the specific action is performed
- (e.g. SEARCH REQUEST or PUBLISH REQUEST)

Kad – List of nodes in eMule



Node ID: A37D1601…

Kademlia – Evaluation

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Advantages

- Has successfully proven function in big real world scenarios
- Resistant to node failures
- Proximity of nodes is used

Drawbacks

- Large routing table required
- Not deterministic routing:
  - Node A performs a lookup: finds k closest nodes
  - Node B performs a lookup: finds “other” k closest nodes
    - k closest node not necessarily the SAME!

Future work required

- Security issues
  - Node IDs can be chosen freely

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

Interconnection Overlay Networks
– Hypercube Networks

This slide set is based on the lecture "Communication Networks 2" of Prof. Dr.-Ing. Ralf Steinmetz at TU Darmstadt

Interconnection Networks

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Motivation

- Assumption 1: no churn
- Assumption 2: number of nodes controllable
- Goal: overlay network for ID-based routing
- Efficient network topologies are possible

Efficient topologies

- Fixed node degree
- Routing in O (log N)

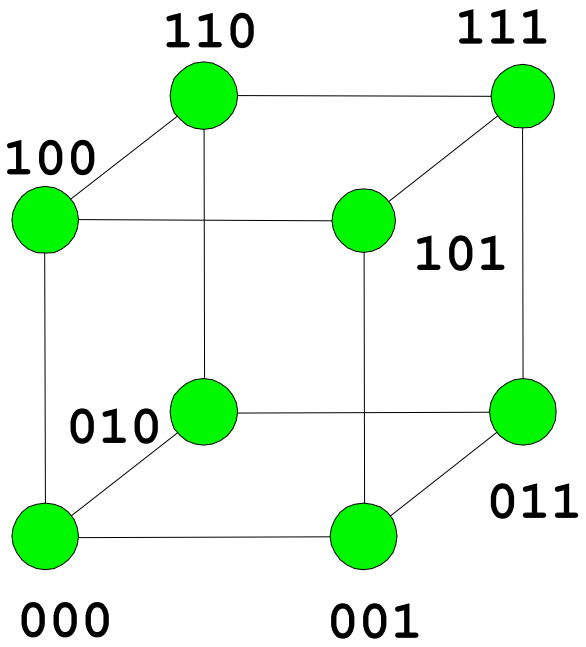
Differences to DHTs

- DHT: provides key-value lookup functionality
- Interconnection networks: only ID-based routing

Hypercube Networks

Example:

- 3-dimensional Hypercube Network



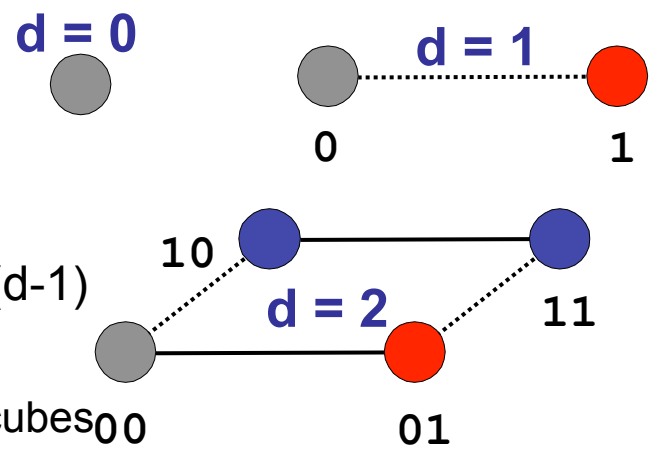
d-dimensional binary hypercube definition

- A d-dimensional binary hypercube network consists of 2^d
  - interconnected nodes
  - whose addresses are represented by d-bit binary numbers.
- A node n has d adjacent nodes
  - whose addresses are obtained by reverting each bit of its address
  - fixed number of contacts per node!

Hypercube Networks: How to construct

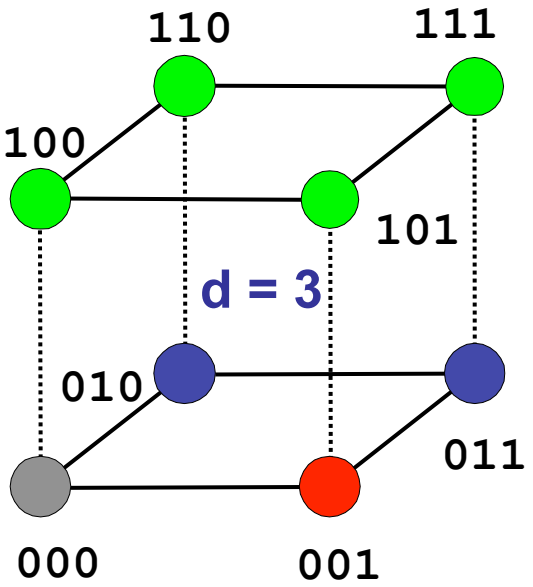
Construction algorithm

- if (d == 0)
  draw a node
- else
  - 1. draw two hypercubes of dimension (d-1)
  - 2. add arcs between
    - corresponding nodes of the two hypercubes



Identification algorithm

- 1. use the first bit
  - to decide which of the two sub-hypercubes the node is in.
- 2. use the remaining d-1 bits as
  - an address within that sub-hypercube.



Hypercube Networks: Routing Algorithm

A simple source-to-destination routing algorithm for binary hypercube networks

- Node failure is ignored

To route from node i to node j,

- the bits of i which are different from those of j are changed,
  - traversing a link with each bit changed,
  - until j is reached.
- Since each node has links corresponding to each of the n bit positions,
  - any bit can be changed at any step in the route;
- →
  - the number of eligible links at each step equals the distance from the destination.

Hypercube Networks: Routing Example

Route from 010 to 101

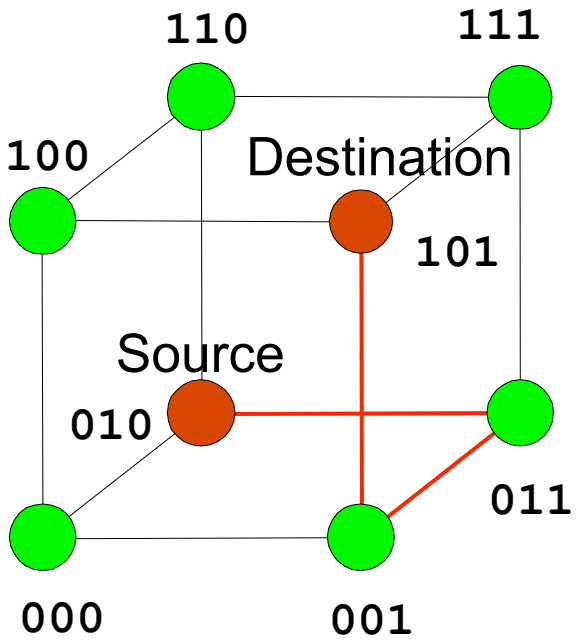
- 010 has 3 options (000, 110, 011)
  - → distance = 3
  - Select 011 as the next hop
- 011 has 2 options (001, 111)
  - → distance = 2
  - Select 001 as the next hop
- 001 is direct neighbor
  - → distance = 1
  - Forward to destination

IF

- nodes fail

THEN

- alternative paths may be followed in dynamic environments



Hypercube Networks: Properties

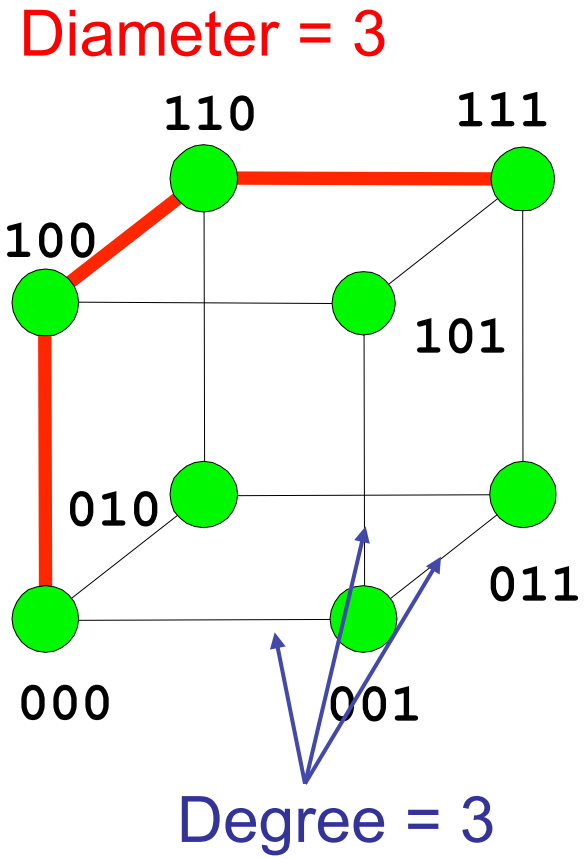
Basic properties:

Diameter of the network
- i.e. worst case node distance
- increases logarithmically with respect to the network size

Node degree
- i.e. number of neighbors
- increases logarithmically with respect to the network size

Network is both
- vertex- and
- edge-symmetric graph

Hypercube is a hierarchically recursive network



Hypercube Networks: Limitations

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Exponentially expandable

- Network size is defined only for 1, 2, 4, 8, … nodes
  - for binary hypercubes
- Not incrementally expandable
  - i.e. network cannot be defined for any arbitrary integer

Node degree increases logarithmically with respect to the network size

- → increasing the required maintenance cost

Mostly appropriate for static environments

- with infrequent joins and leaves

Interconnection Networks: Evaluation

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Scenarios:

- Controlled environments (provider defined topologies)

Benefits

- Fixed and low node degree
- Fast routing: O (log N)

Drawbacks

- Exponentially expandable
  - Network size is defined only for binary graphs
    - Node count: 1, 2, 4, 8, 16 … 2^i
  - Not incrementally expandable
    - i.e. basic network cannot be defined for any arbitrary integer
- Cannot handle churn

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

Interconnection Overlay Networks
– de Bruijn Networks

This slide set is based on the lecture "Communication Networks 2" of Prof. Dr.-Ing. Ralf Steinmetz at TU Darmstadt

Interconnection Networks

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Motivation

- Assumption 1: no churn
- Assumption 2: number of nodes controllable
- Goal: overlay network for ID-based routing
- Efficient network topologies are possible

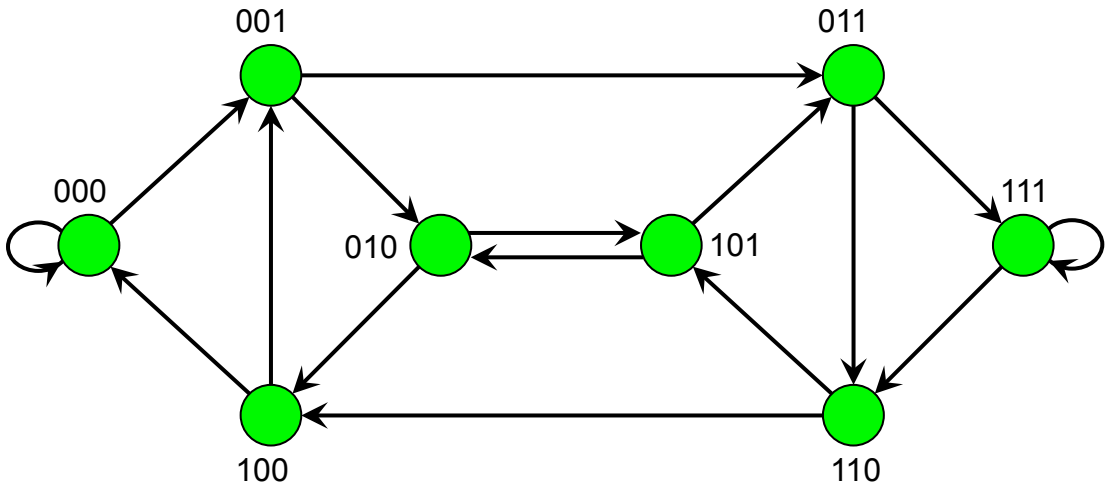
Efficient topologies

- Fixed node degree
- Routing in O (log N)

Differences to DHTs

- DHT: provides key-value lookup functionality
- Interconnection networks: only ID-based routing

de Bruijn Networks



r-dimensional binary de Bruijn digraph (directed graph) consists of

- 2^r nodes and 2^(r+1) edges

Each node corresponds to

- An r-digit binary string

There is a directed edge

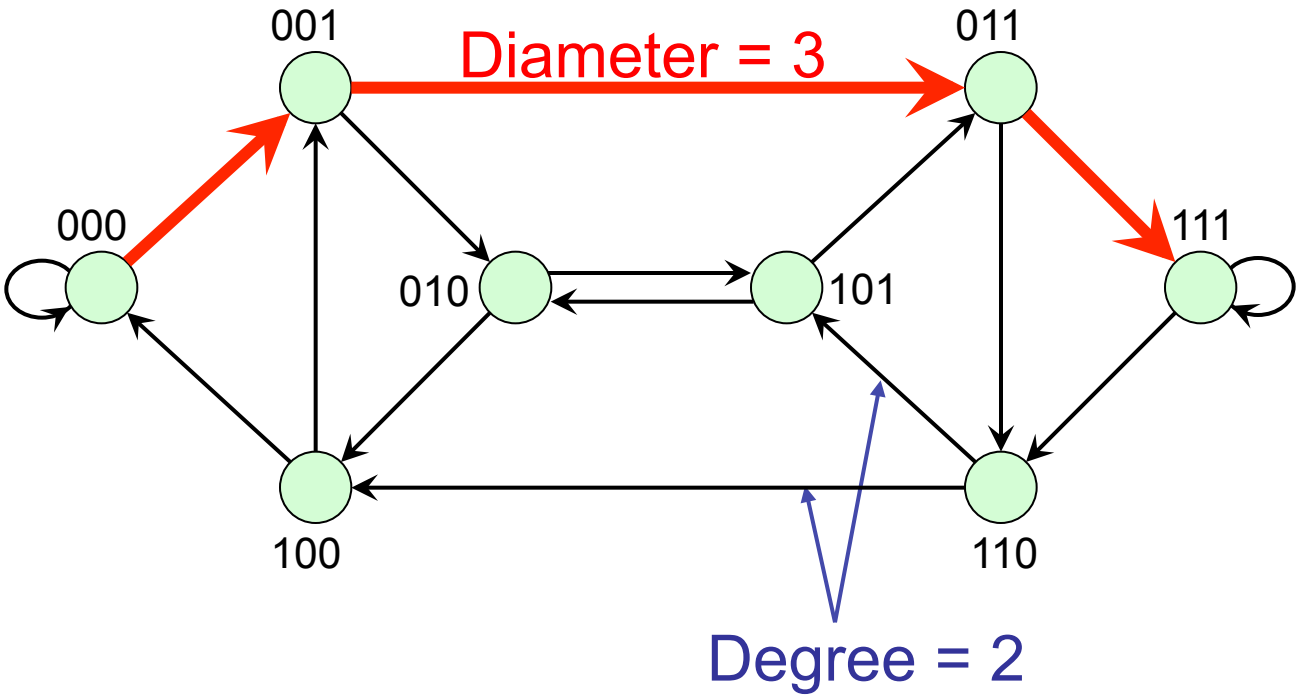
- from each node (u1u2…ur) with ui either 0 or 1
- To (u2 u3 … ur-1 0) and (u2 u3 … ur-1 1)
- Principle: drop first character and add 0 or 1 to the end

Number of in and out edges per node: 2 each

de Bruijn Networks: Properties

Basic characteristics

- Average distance is very close to the diameter
- Constant vertex degree
- Logarithmic diameter
- Adjacency is based on left shift by 1 position



de Bruijn Networks: Routing Algorithm

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Assumptions

- Dimension of the network
- Source node K
- Destination node L

Operation: shift_match(s, K, L), where 0 <= shift <= d

- returns TRUE if and only if $k_{1+shift}k_{2+shift}\ldots k_D = l_1 l_2 \ldots l_{D-shift}$
- returns FALSE otherwise
  - ~ colloq: cut the same no. of bits in the source ID left as in destination ID right

Operation: merge(shift, K, L), where 0 <= shift <= D.

- returns the sequence of length (D + shift) given by $k_1 k_2 \ldots k_D l_{D-shift+1} l_{D-shift+2} \ldots l_D$

Shortest-path algorithm (Sivarajan and Ramaswami)

shortest_path(K, L)

- shift = 0
- while (shift_match(shift, K, L) == FALSE and shift < D)
- do shift = shift + 1
- return merge(shift, K, L)

de Bruijn Networks: Routing Example

Example: route
- from node with ID K: 001
- to node L: 101

Common match: 00__1__01

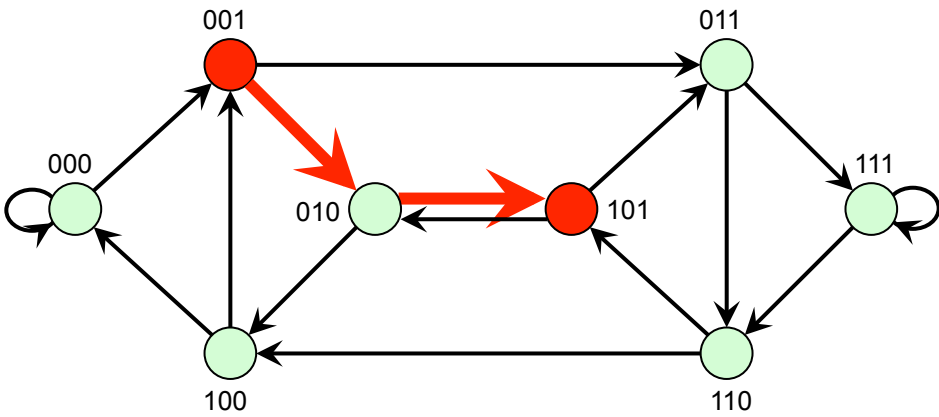
Only shift left operation is allowed
- i.e. only 2 entries in the routing table

Node address			Link
0	0	1	0
0	1	0	1
1	0	1	

Example
from node 001

to node 101

Left – shift operation:
- Drop left letter
- Add new letter on right side
  - Corresponds to using link

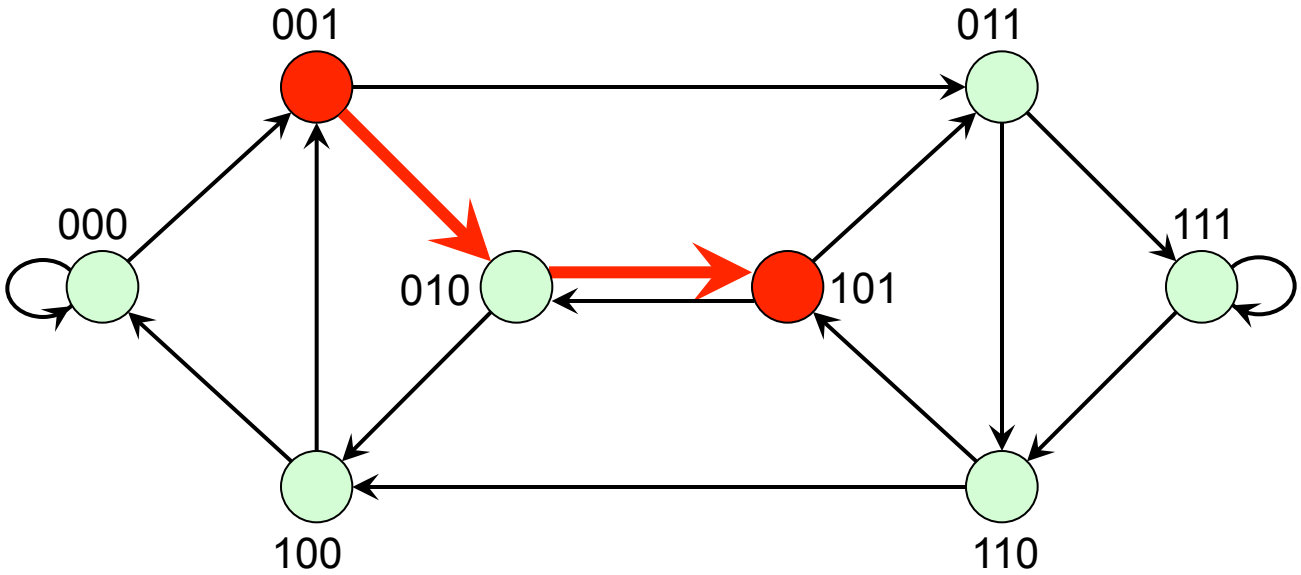


de Bruijn Networks: Routing Example

i.e. Route message from source (001) to (101)
shift = 0
shift_match(0, 001, 101) = FALSE … 001 101
shift = 1
shift_match(1, 001, 101) = FALSE … X01 10X
shift = 2
shift_match(2, 001, 101) = TRUE … XX1 1XX

return merge (2, 001,101) = 00101 … 01

- 2 forwarding steps



de Bruijn Networks: Routing Example

Example: route

- from node with ID 11110
- to node 01001

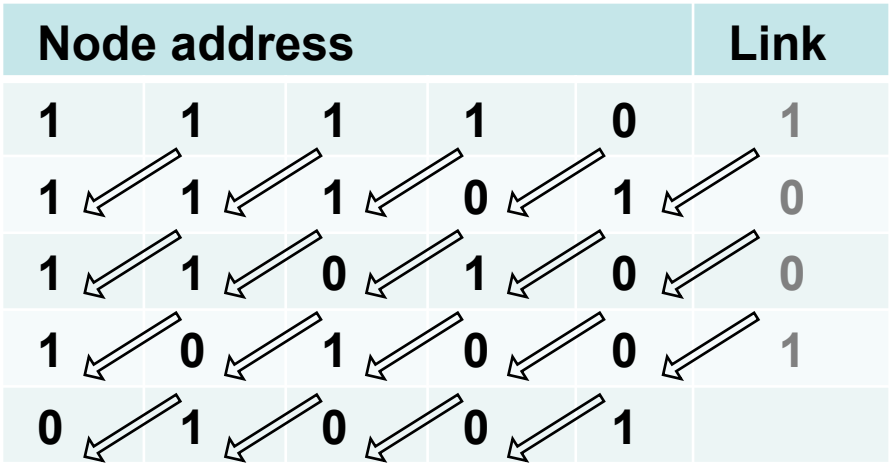
Common match:

1111_0_1001

Bits to insert stepwise: 1001

Only shift left operation is allowed

- i.e. only 2 entries in the routing table



de Bruijn Networks: How to construct

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Partial Line Digraph Algorithm: recursive construction

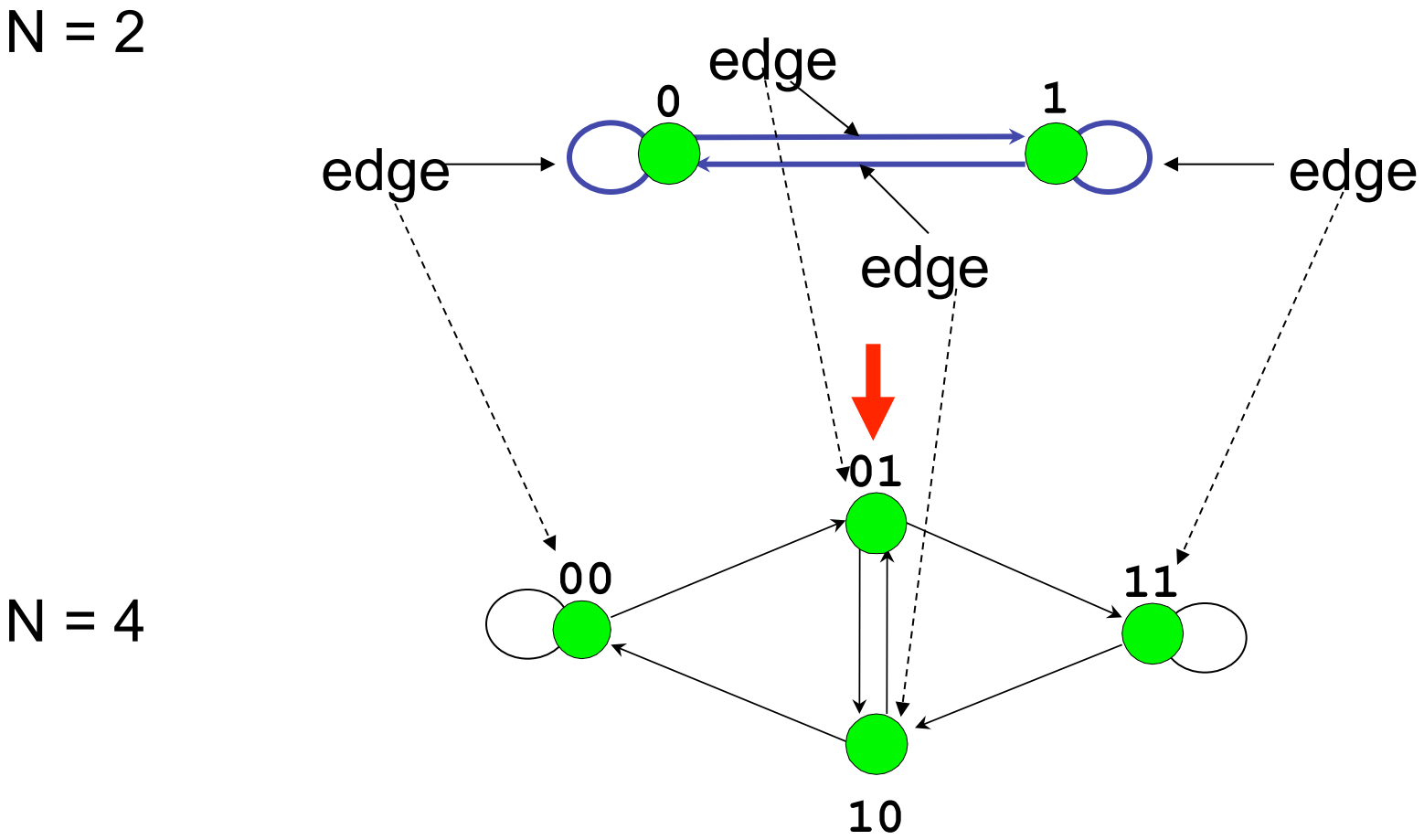
- The N-node graph (dimension r+1)
  can be obtained from two N/2 –node graphs (dimension r)

Algorithm description

- Replace every edge of the N/2-node graph with a node
  - Edge (u1u2…ur, u2u3…ur+1) → Node (u1u2…ur+1)

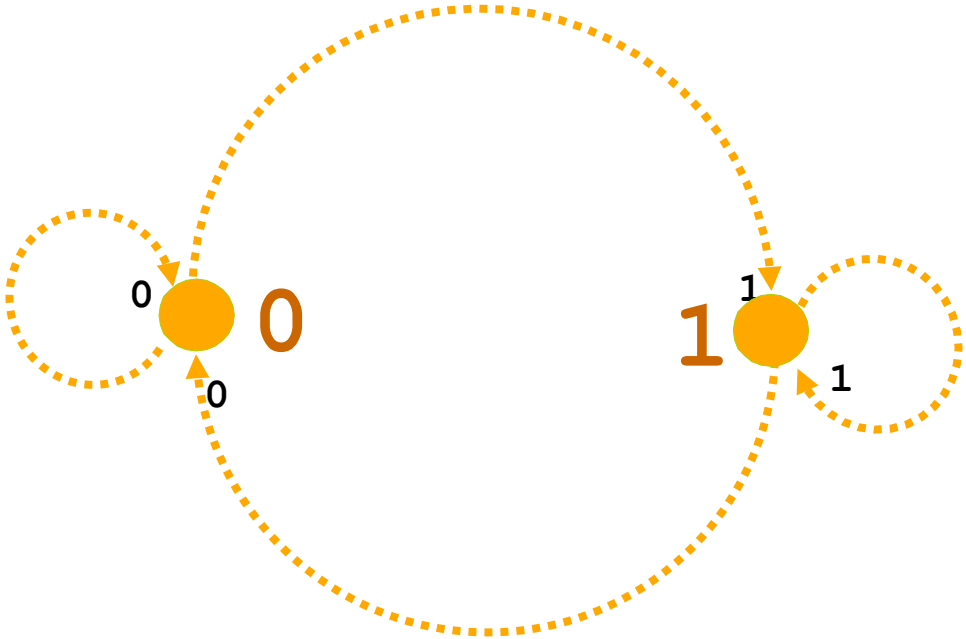
- Insert a directed edge between pairs of nodes
  that correspond to consecutive directed edges in the N/2-node graph
  - Edge (u1u2…ur, u2u3…ur+1) and Edge (u2u3…ur+1, u3u4…ur+2)
  - Replaced by (u1u2…ur+1, u2u3…ur+2)

de Bruijn Networks: Construction Example

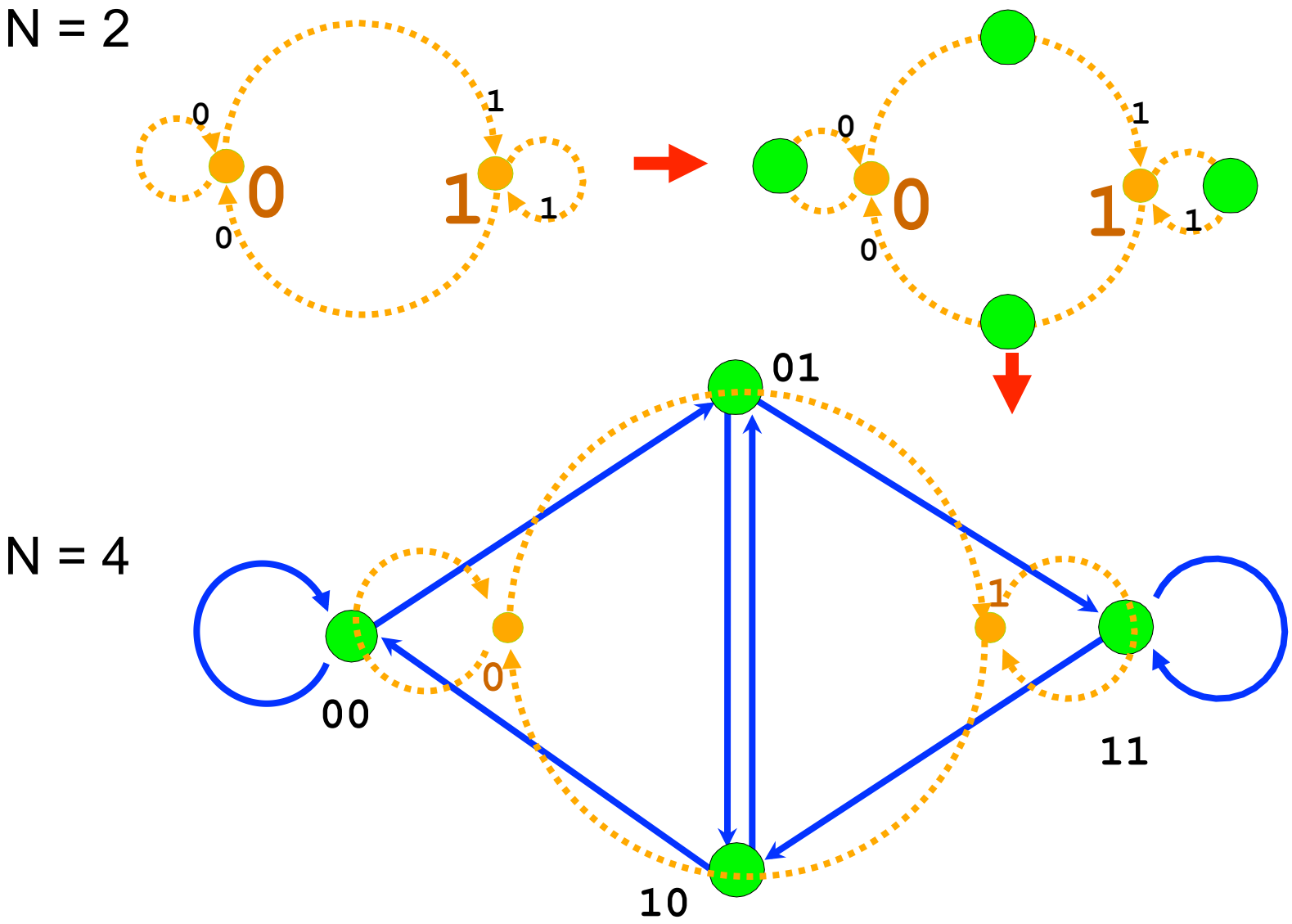


de Bruijn Networks: Construction Example

N = 2



de Bruijn Networks: Construction Example



Interconnection Networks: Evaluation

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Scenarios:

- Controlled environments (provider defined topologies)

Benefits

- Fixed and low node degree
- Fast routing: O (log N)

Drawbacks

- Exponentially expandable
  - Network size is defined only for binary graphs
    - Node count: 1, 2, 4, 8, 16 … 2^i
    - Not incrementally expandable
    - i.e. basic network cannot be defined for any arbitrary integer
- Cannot handle churn