Avoiding Traffic ctd.

- Which degrees of freedom do we have?
  - Select neighbor
  - Select next hop
  - Select ID !?

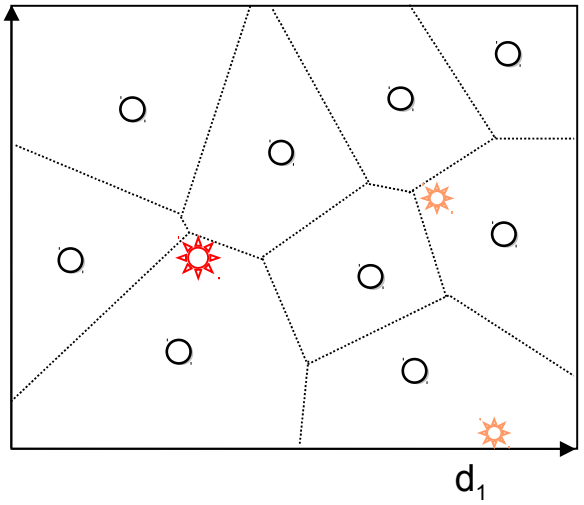
    (The respective others kept conventional…)

- Location-based neighbor selection
  - Pastry, Tapestry, etc.: only store the closest in routing tables
- Location-based next-hop selection
  - Any: from all neighbors that are closer to resource select the nearest
- Topology-based ID selection
  - „Learn“ ID depending on
- All of them have pros and cons

CAN revisited: Location aware DHT

- Synthetic coordinates used as ID in DHT

  $mapV(v) := \vec{v} = [v_1 \| \ldots \| v_d]$



- Registration
  - Map resource ("**o**") in the coordinate space

    $mapO(o) := \vec{o} = [o_1 \| \ldots \| o_d]$

  - Register at different coordinates using well known functions:

    $M1(\vec{o}) = -\vec{o} = (-o_1, \ldots, -o_d)$

    $M2(\vec{o}) = ((o_1 + o_{max}) \bmod (2 * o_{max}), \ldots, (o_d + o_{max}) \bmod (2 * o_{max}))$

- Routing
  - Greedy-Routing:

    $nextHop = v : |\vec{v} - \vec{o}| = \min \{ |\vec{v} - \vec{o}|, v \in Neighbors\}$

- Overlay-Construction
  - Select all "direct" neighbors in the coordinate space (in all directions)
  - Additional neighbors in different distances in diverse directions

Strufe: Ein Peer-to-Peer-basierter Ansatz für die Live-Übertragung multimedialer Daten. PhD thesis

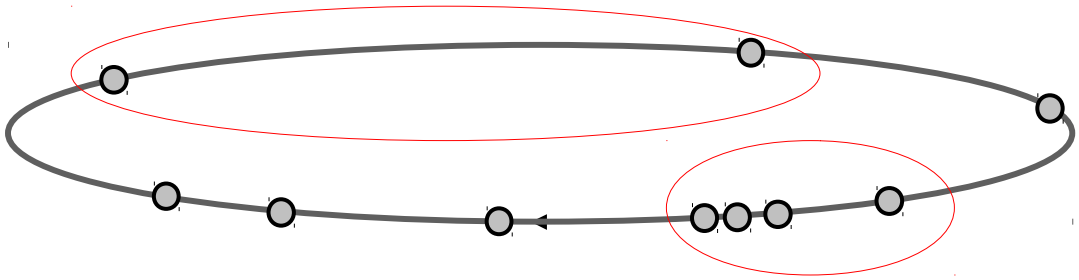
What About the Load at Peers?

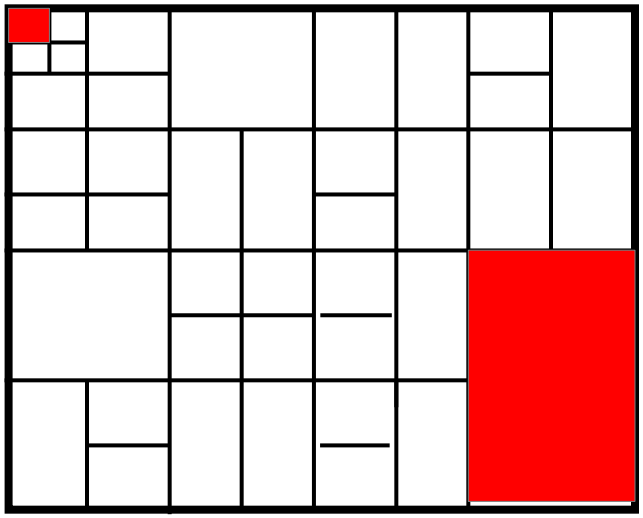
- „A major property of P2P systems (/DHT) is their inherent load balancing.“
  - Requests are served from all peers equally
  - Task of uploading files is shared between all downloading peers
  - Rather random neighbor selection leads to fair allocation of requests
  - Random ID selection leads to good distribution of the namespace…
- What kind of load?
  - Messaging load
  - Request processing load

Load Imbalance

So what can go wrong?

- Uneven distribution of names in ID space (Zipf!)
- Neighbor selection random (preferential attachment?) -> uneven in-degree, uneven incoming requests
- ID selection random -> normally distributed name space allocation

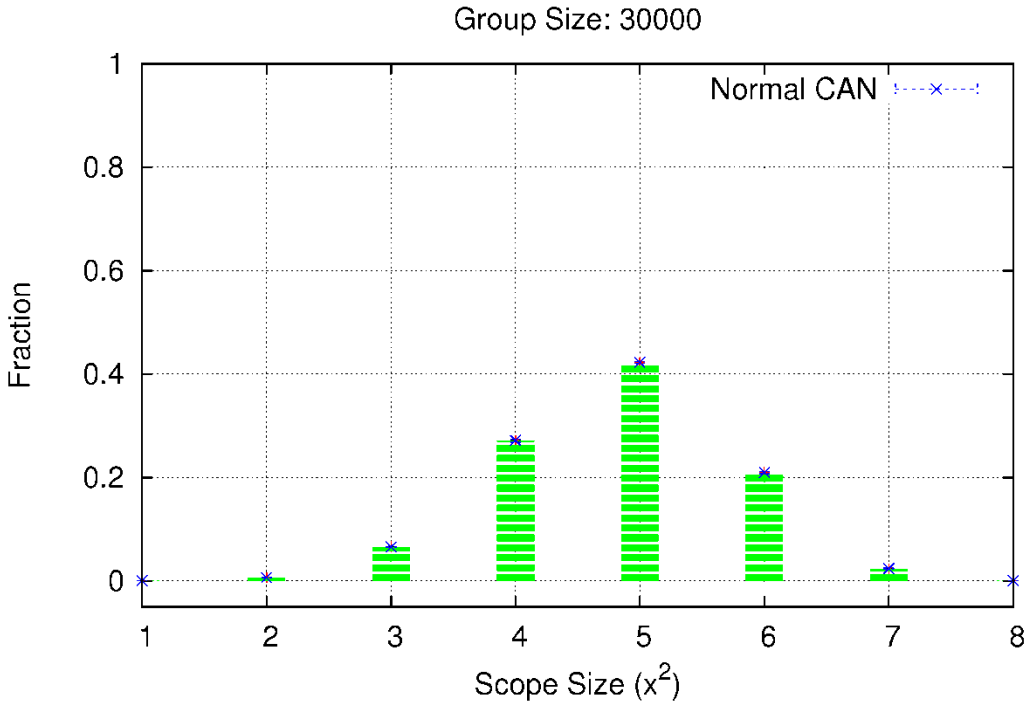


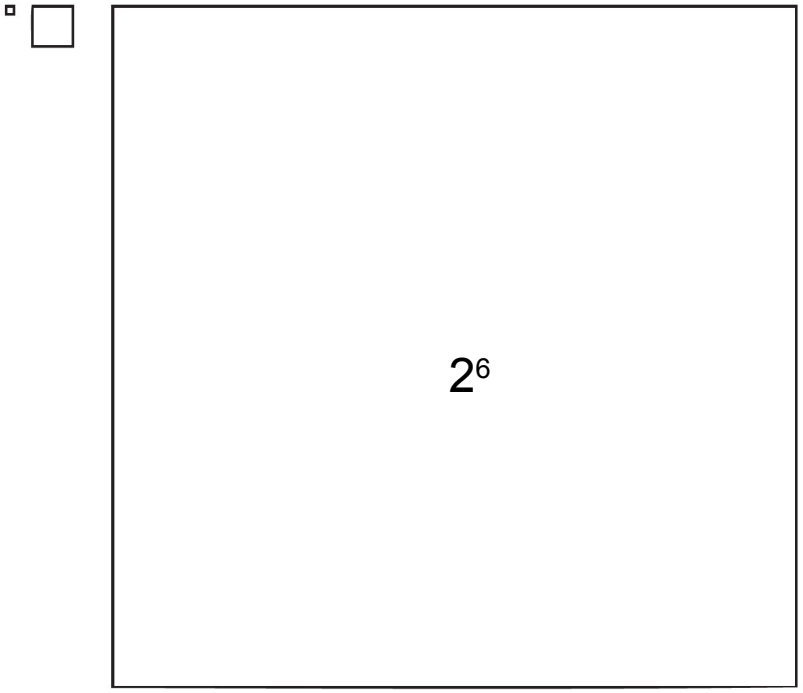


- Heterogeneity of peers! (High-end PC at TU Darmstadt vs. My mobile phone...)

Difference in Area Sizes…

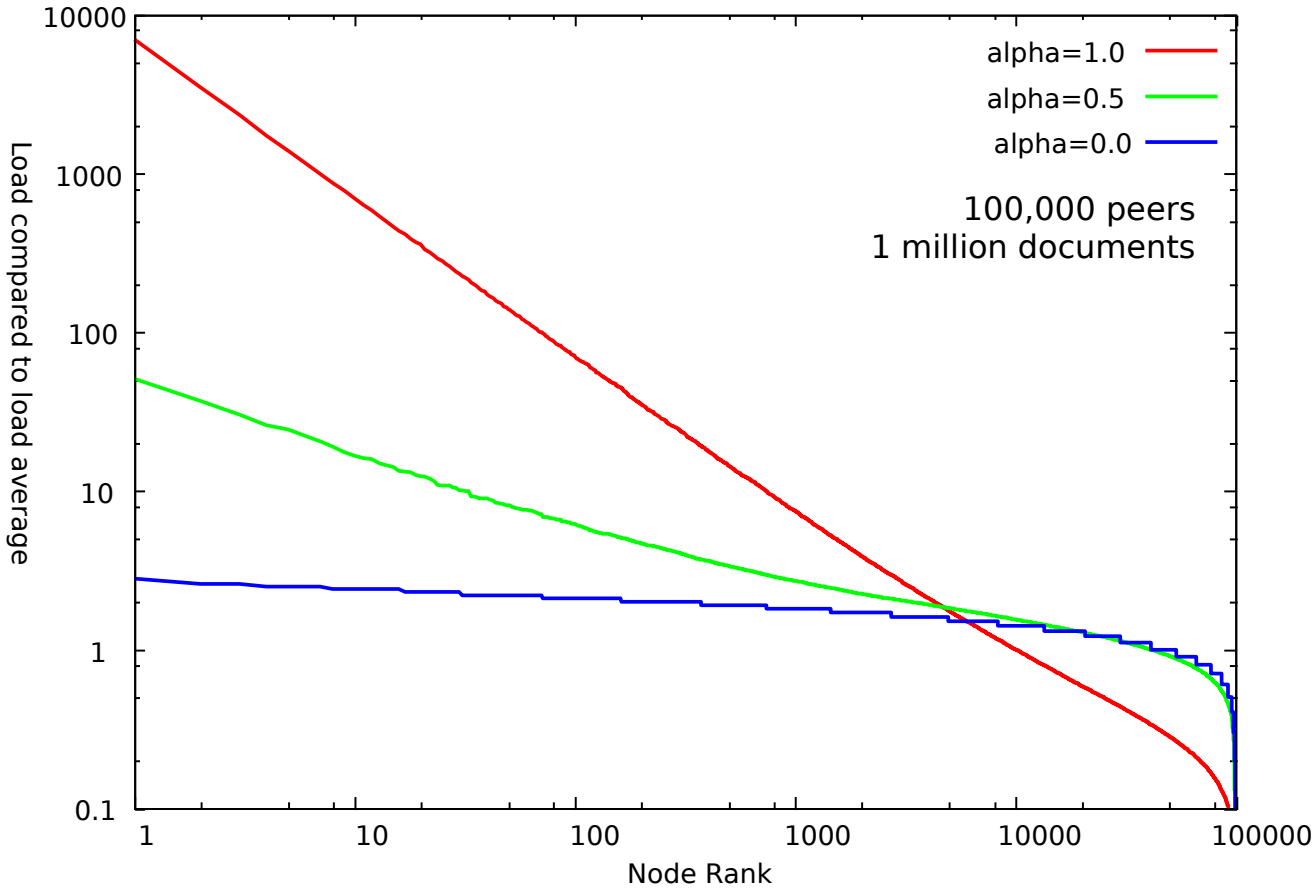
- Nodes in CAN are allocated areas differing up to factor 2^8 in size easily (s.b., only 30k nodes)…





Tiny example for comparison…

Zipfian Load



- For natural languages (e.g. full text search) in a keyspace:
  - Expected load on the most loaded peer is 7000x average
  - The loaded peer probably has only average capacity

Terpstra et al.: BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search

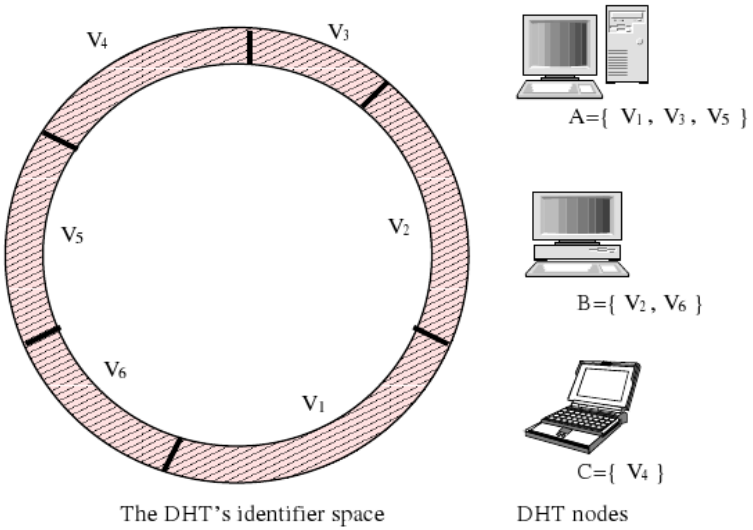
So what can we do?

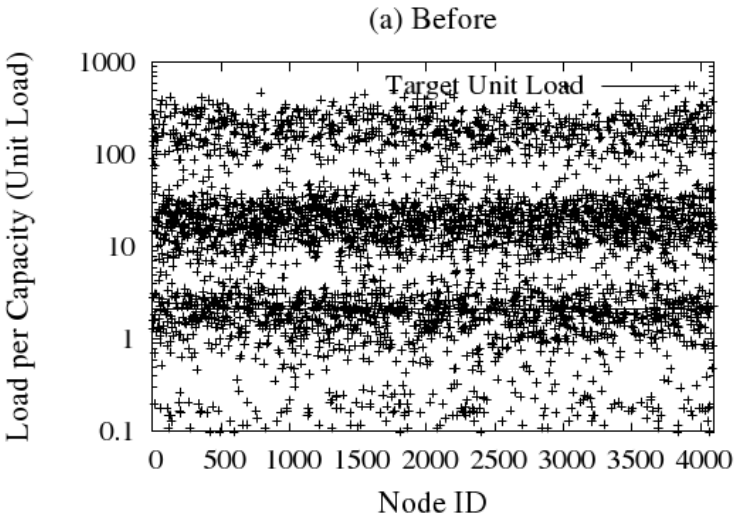
- Don‘t know the mis-balance (and very hard to gather)
- Goals:
  - Balance request load
  - Balance name space allocation
  - (Allow for heterogenous nodes to adapt load?)
  - **Side note: we‘re /NOT/ thinking security here!**

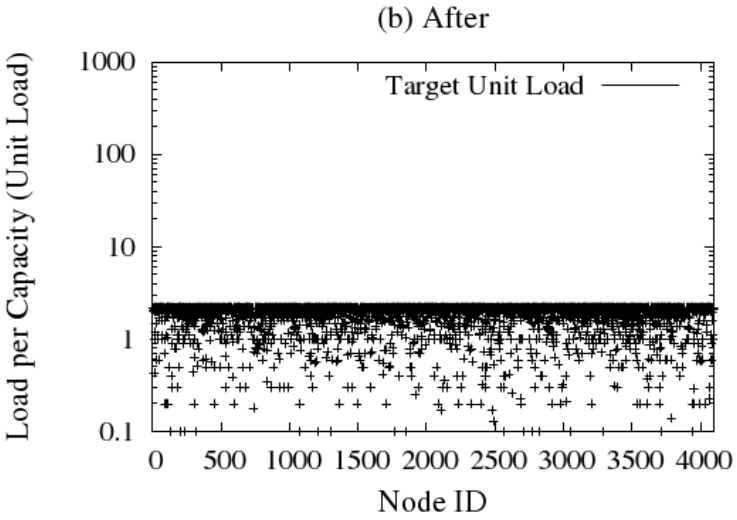
- Possible solutions (with three examples)
  - Mitigate (change protocol/system slightly, statistically for the better)
  - Control (change protocol to guarantee better balance deterministically)
  - Adapt (Only change when necessary)

Virtual Servers (Mitigation attempt)

- Virtual Servers:
  - Areas in name space normally distributed
  - All areas are small, if there are only enough peers!
  - Idea: why don‘t we assign x „servers“ with small area to



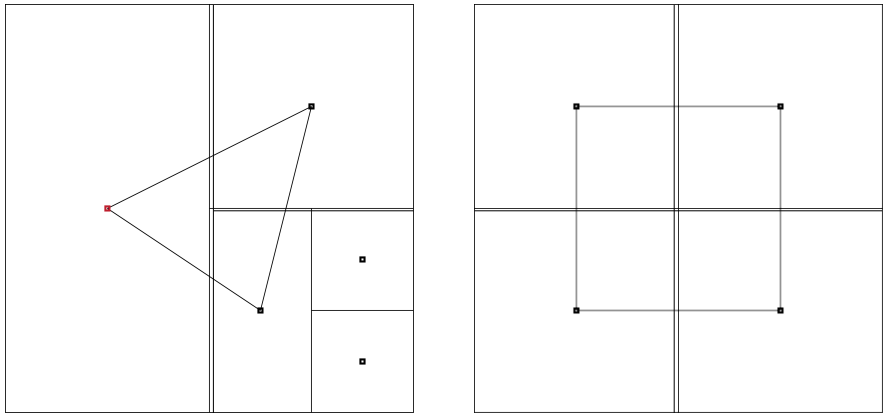


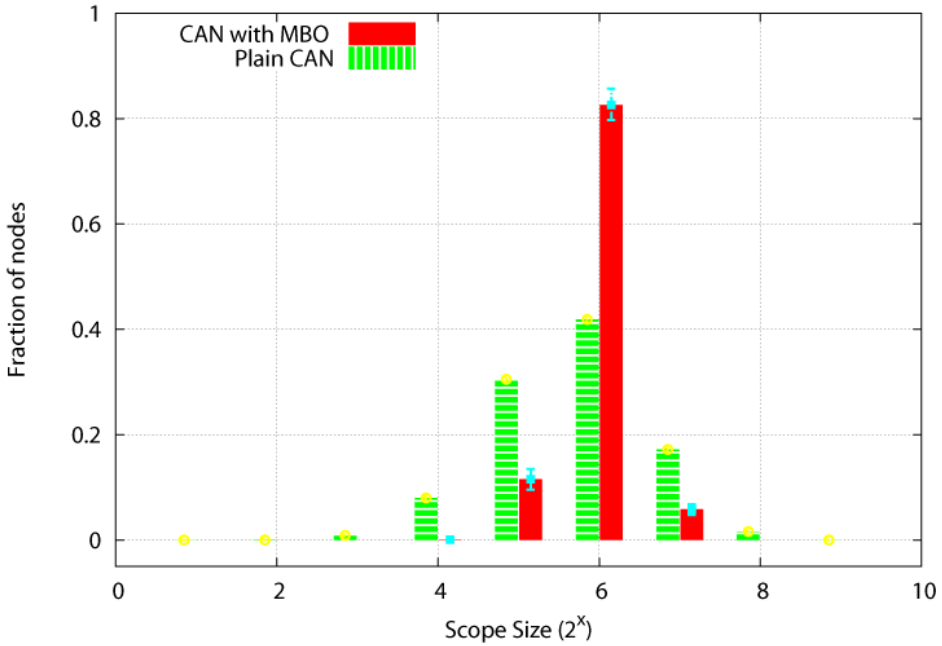


Y. Zhu et al.: Towards efficient load balancing in structured P2P systems

Force Balance (Control)

- Topological changes lead to better balanced allocation / incoming requests
- Check direct environment and optimize to local balance
  - Motif-based is one option
  - (check relations between nodes in local environment)

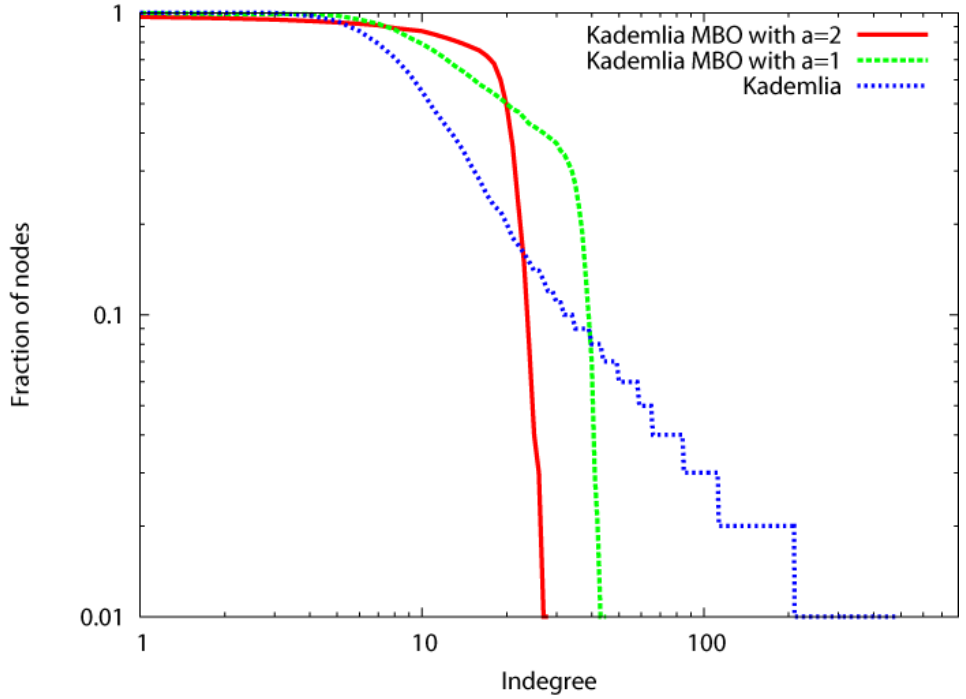




Krumov, Schweizer, Bradler, Strufe: MOPS: Optimizing Structured Peer-to-Peer Networks Based on their Local Motif-Signature

Balancing Kad Request Processing

- Same idea for Kad:
  - Balance the in-degree
  - the same number of requests are expected for all
  - Problem: what is the mean? How would it be balanced?
  - Idea: again just use relations!



„Grow“ the DHT Adaptively

- In case of overload, balance!
  - Recall: in Tapestry objects are stored in multiple peers with same prefix…
  - Tapestry/Pastry: increase number of routed bits
- Kademlia‘s structure over the namespace is a tree…
  - Increase the branch length…

- ID-allocation wrt balance: P-Grid
  - „Attract“ more peers to regions with high load and grow branch
  - Actually, why use these Hashes in the first place?
  - Just load-balance over name space of objects!
  - Downside: creating an over-sophisticated system with lots of messaging
  - … P-Grid not really used by networking people, but database‘rs love it!

P2P-Network Design

- Recall:
  - Main primary problems of P2P was:
    - Connectivity
    - Resource Location
  - So far solved:
    - Connectivity: be redundantly connected
    - Resource location: send and delegate request
      - „Where?“ (Routing, flooding)
      - „How many?“ (Request/registration replica)
- Desgining a P2P system with demanded requirements

Designing P2P: Degrees of Freedom?

- What can a **joining** node (or the designer) decide upon?
  - „Free“ choice of ID
  - Selecting neighbors
    - At degree **d**
    - Which
- Which freedom do we have to **store** data?
  - Leave local, let others ask for it
  - Register where?
  - How often **(deterministic?)**?
- …for **lookup** (..and delegation…)?
  - Request where?
  - How often?

Designing P2P (more general)

- P2P search/location systems implement a name/id resolution
- How is name space **distributed** – and where is it **implemented**?
  - No explicit name space distribution
    - Arbitrary assignment (implemented everywhere, gnutella)
    - Hierarchic assignment (eDonkey, fasttrack) **(and the SN?)**
  - Explicit name space distribution
    - Assigned by structured allocation

What about Kademlia (non-deterministic routing)?

- Routing: request/delegate to where item may be!
- And: after bootstrapping: hang on to the crowd (stay connected)

Designing P2P: 3 Main Design Decisions

- Neigbor selection
  - Which
  - How many
- Requesting / Delegation (routing)
  - Where?
  - Deterministic?
  - Iterative/Recursive?
- Replication
  - Content / Registrations
  - Queries

Designing P2P: Hands on!

- „We want to use DHT for our in-house data management. It has to be highly reliable and extremely fast. Help us!“
- What do you do?

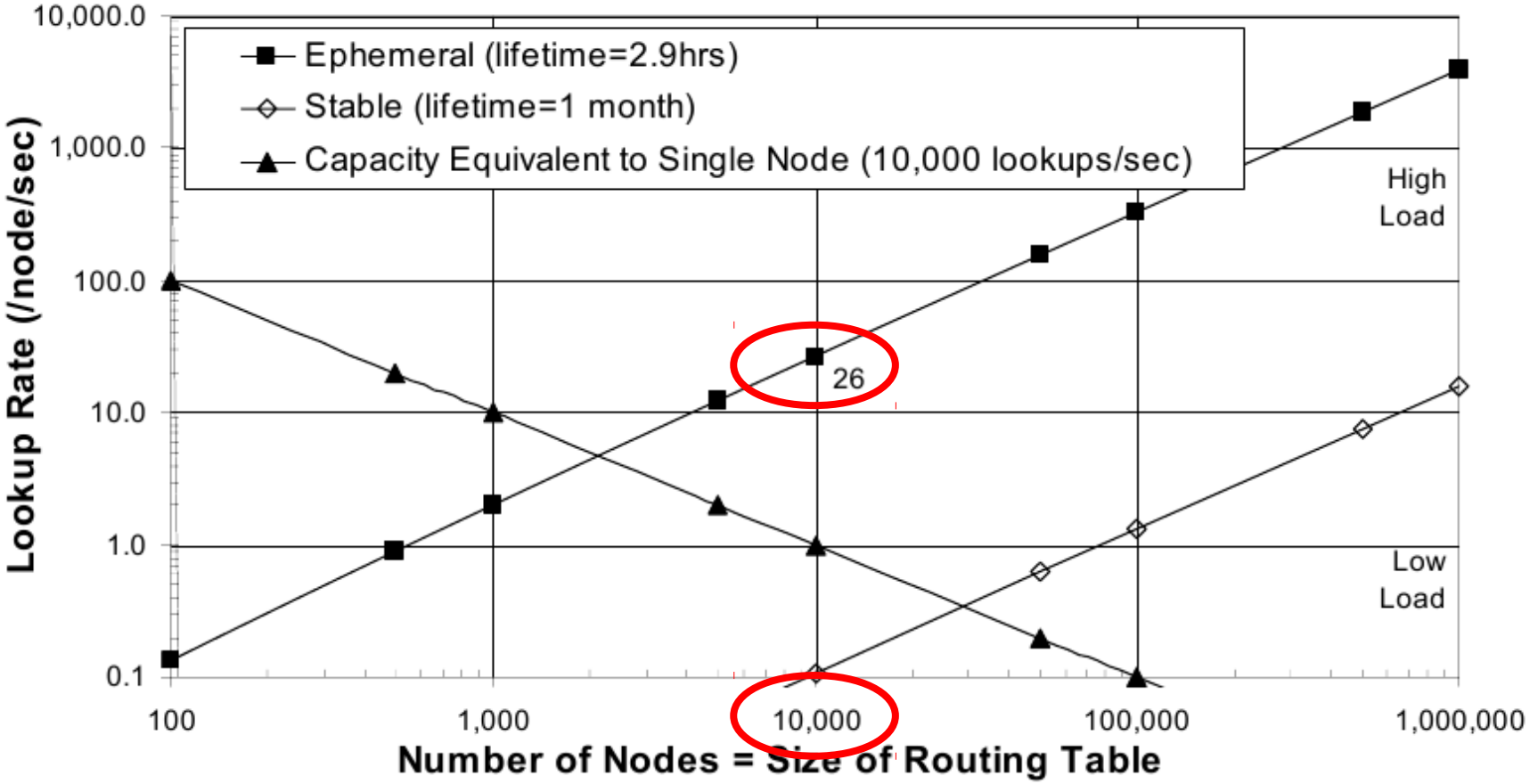
D-P2P: Fast Reliable Lookup

- One-hop DHTs!

- Select $\sqrt{n}$ neighbors each (few more)
- Create DHT
- All your base are belong to us.

- Does this work in a file-sharing setting?
- Recall: **Churn!**
- What happens?
- At which request rate (/node/sec) does this make sense?

Designing P2P: Hands on (1/2)!



- Mgmt. overhead equals saved requests

Source: Risson et al.: Stable High-Capacity One-Hop Distributed Hash Tables

Designing P2P: Hands on (2/2)!

- „We have a distributed set up, and we need reliable, fast lookups – PLUS: comprehensive search!“
- What do you do?

D-P2P: Fast Comprehensive Search

- Comprehensive search: Find **all** content matching request!

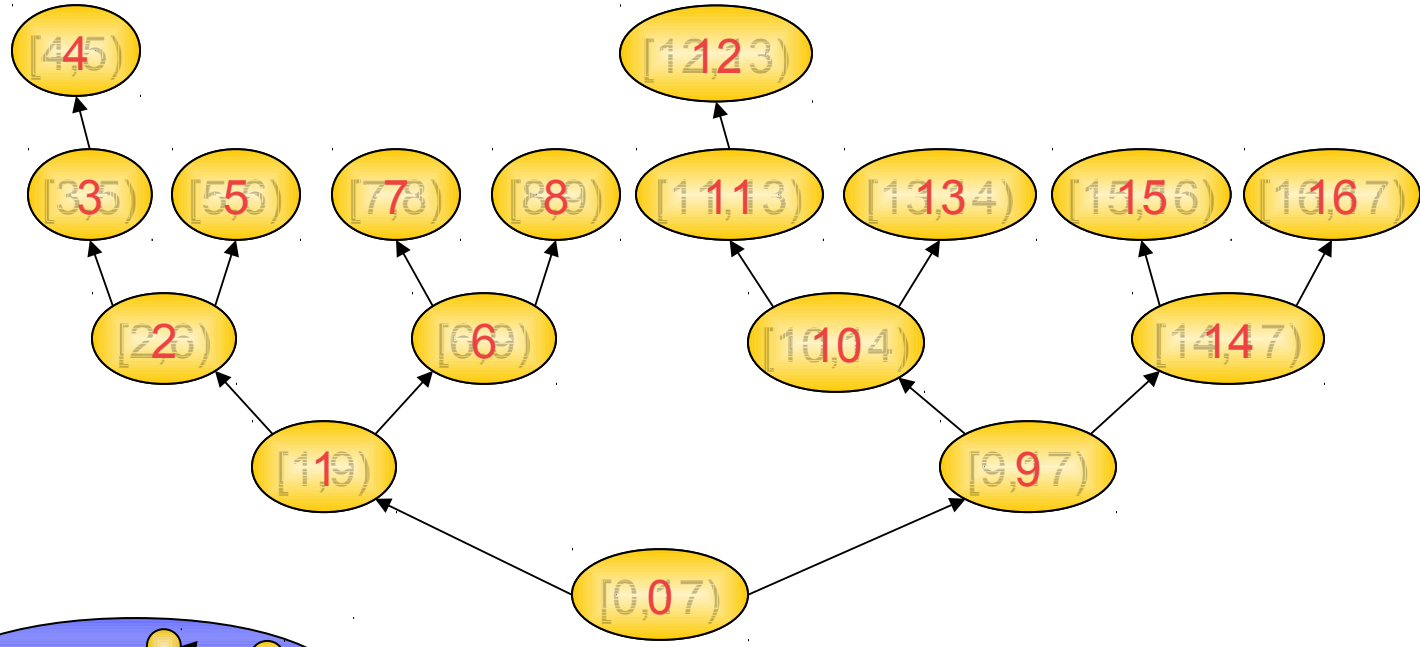
- Ask **all** nodes -> Ring/Tree routing in DHT, broadcast request
- But it‘s supposed to be fast!

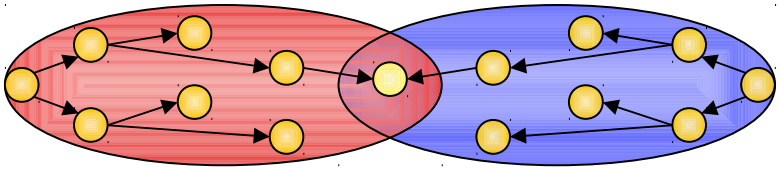
- Keep name space local (unstructured -> comprehensive)
- Efficiency: random walks
- Speed: replicate! (Registration and walker)
- Reliability: that‘s tricky ☺
  - Reliability is always bound to probability…
- Increase (provably) probability that walker hits registration

D-P2P: Fast Comprehensive Search: BubbleStorm

- The „trick“ to make it provable:
  - Be random!
  - We can prove a lot about entirely random choices, hence:
- Neighbor selection
  - Choose random neighbors (not entirely true, need means for mgmt.)
- Delegation
  - Random walks
- …with replication
  - Fanning to replicate registrations and to decrease delay for hits

D-P2P: Example Bubblecast Execution





Source: Terpstra, Leng, Buchmann: Bubblestorm

Designing P2P: Hands on (3)!

- „We live in a country with a totalitarian government, freedom of speech isn‘t given, opinions are dangerous. Can we have a censorship resistant system to anonymously publish our jokes?“

- What do you do?
- …ask what the adversary looks like…
  - May be able to resolve IP->home address
  - May be able to quickly confiscate devices
  - Puts „victim“ in jail if „illegal“ content is found on device
- Ask which protection is needed
  - Source of data must not be identifiable
  - Data should be durable/available

D-P2P: Censorhip Resistant Publication

- What **you** would do:
- Neighbor selection
  - Plenty, at least neighbors according to some structure (DHT)
- Delegation/Registration
  - Structured
  - Recursive (protect the data store)
  - Push the data into the overlay
  - Re-register continuously
  - Is the client that‘s asking protected?
- With replication
  - Keep data everywhere, cf. Tapestry (trade-off wrt. size)

D-P2P: Censorship Resistant Publication

- What others did (freenet)…
- Neighbor selection
  - Random
- Delegation
  - „Steepest-ascent hill-climbing“ (greedy, no real routing metric..)
- Replication
  - As expected