How to Design Robust Networks?
Connect to the Seniors!

Stefan Schmid
Christian Scheideler

Popular Networks

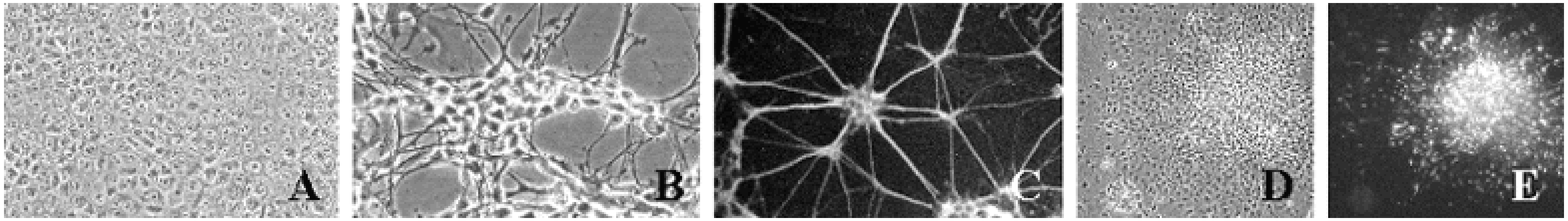
Social networks...

... peer-to-peer networks ...

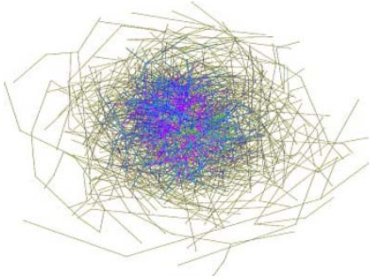
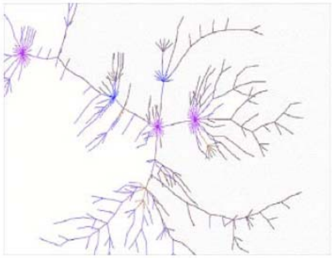
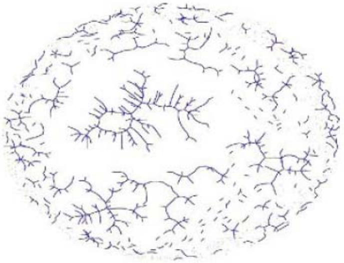
etc.

Interesting Properties

- „Small World Phenomenon“
  - Short chains of „friends“ between two people
  - Also neural networks of animals



- Topological properties of Gnutella
  - Tolerates random, but not worst-case failures

Another Useful Property? (1)

- Often older network participants are more reliable!

- Measurements in p2p networks:

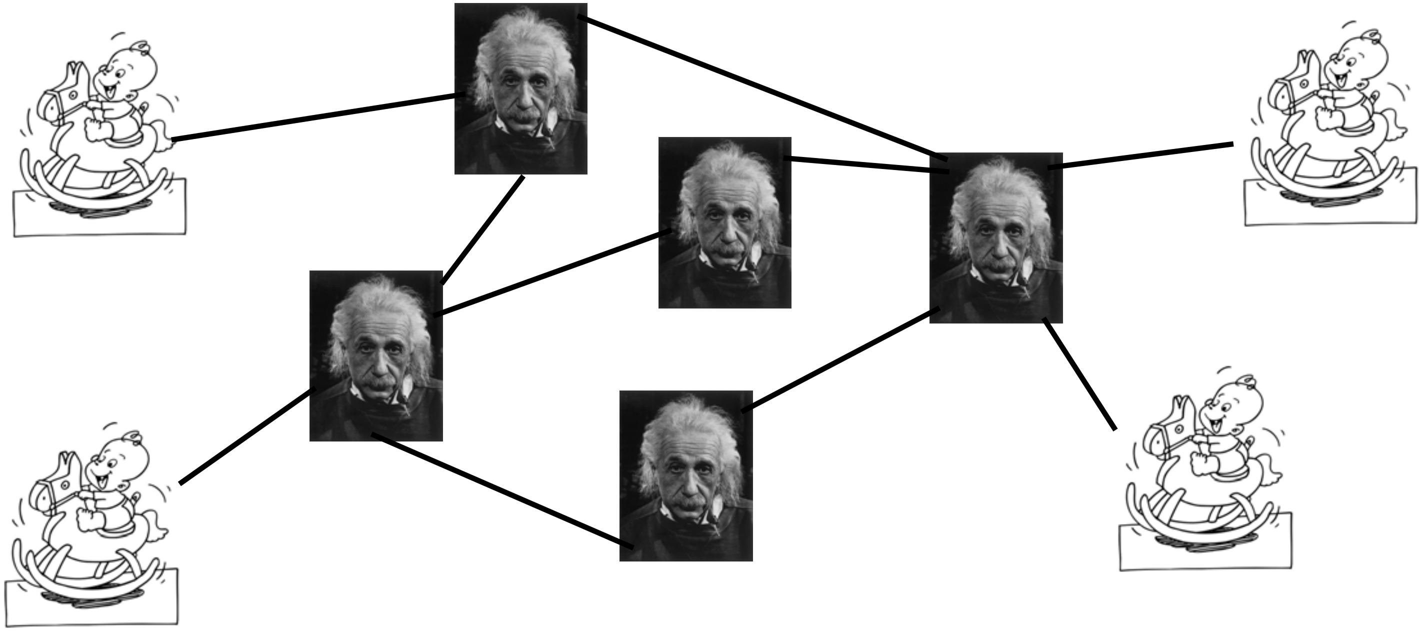
  - Nodes that have been online longer, will typically stay longer
  - Older neighbors often more stable

- E.g. Social networks?

  - Who already wrote many good Wikipedia articles, is likely to do so also in the future?

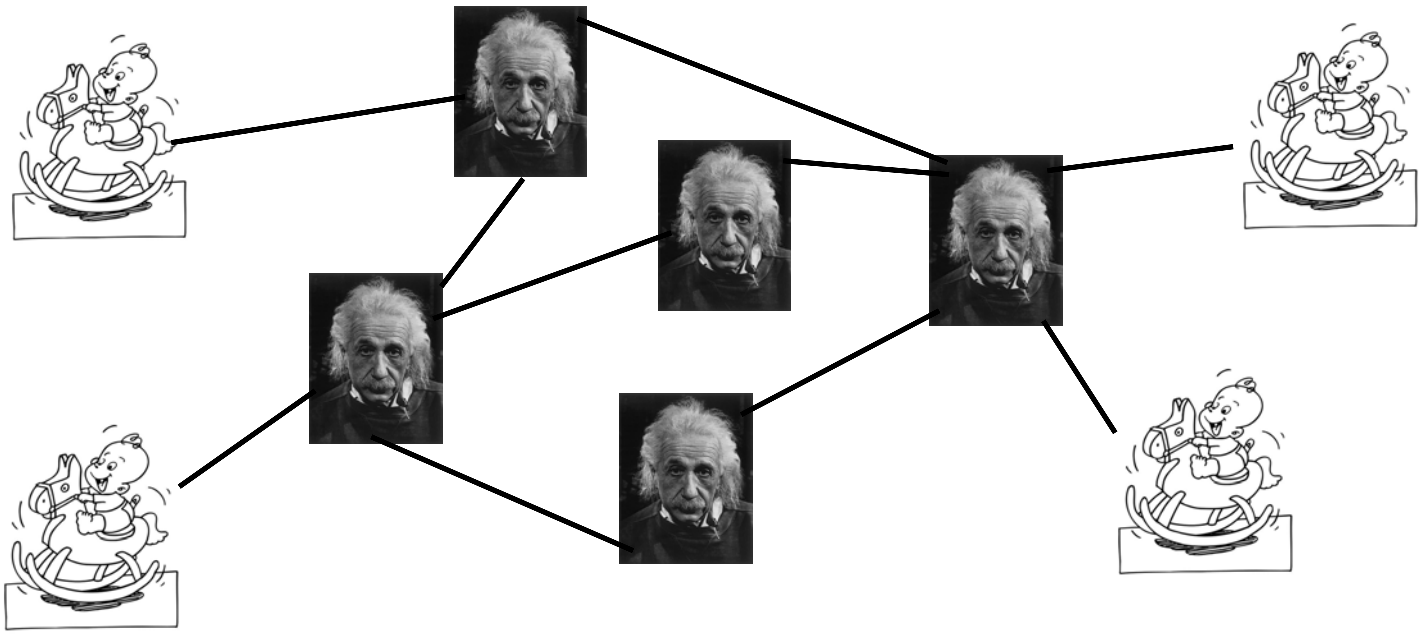
Another Useful Property? (2)

- Idea: if everybody only connects to older network participants...
  - ... nodes would have stable neighborhoods!
  - ... one is resilient against attacks by „young troublemakers“



Another Useful Property? (3)

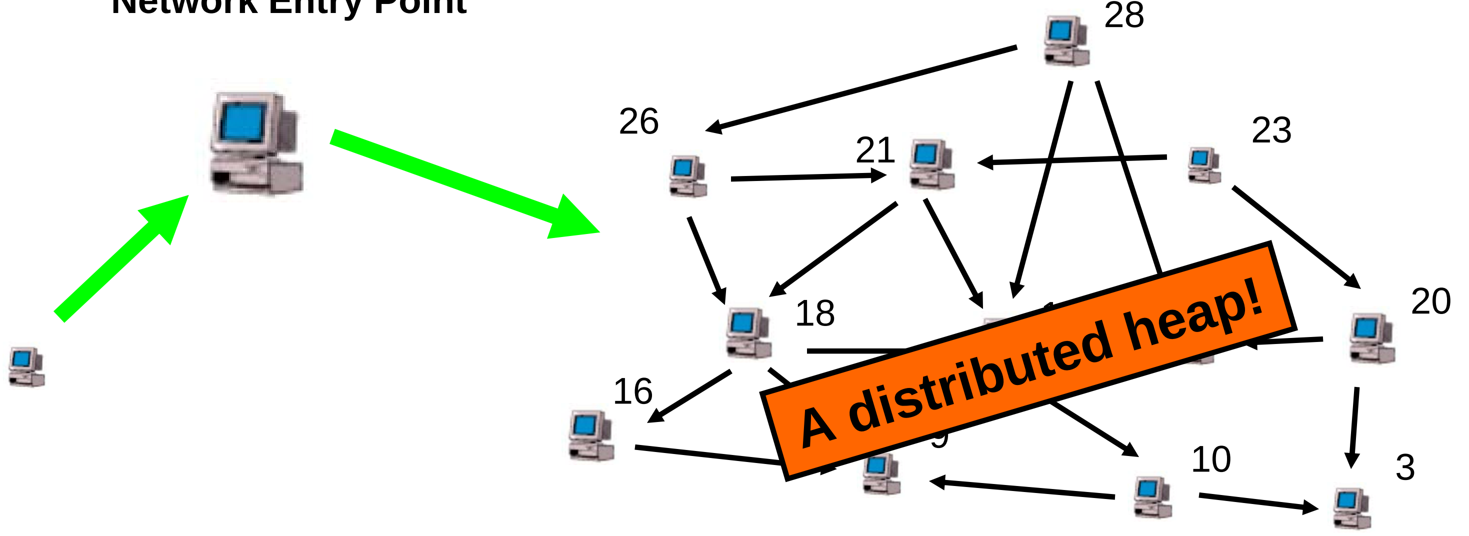
- Implications:
  - [Communication paths] of the „seniors“ never include younger nodes -
  - Young nodes cannot overload network (rate control in „core network“)



Model

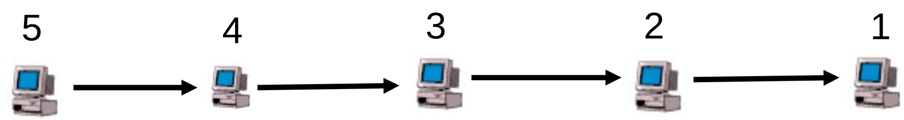
- How to implement such an idea?
- Idea: A central server assigns joining nodes a rank
  - Nodes only connect to nodes that arrived earlier (lower rank)

Network Entry Point



A naive Solution

- Our goal is achieved with:



- Problem: Scalability
  - Large diameter, not robust to join/leave, etc.

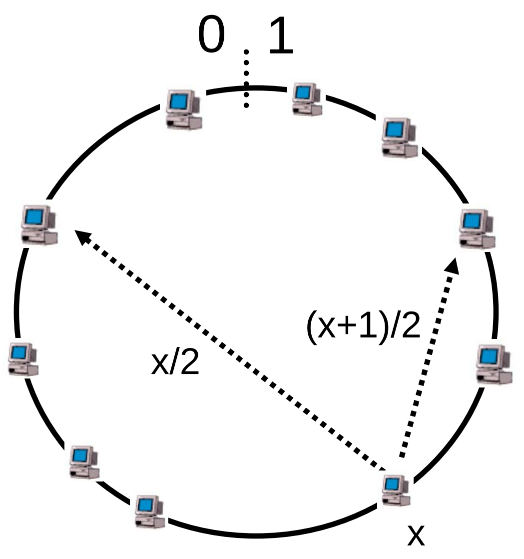
- Better topologies: hypercubes, pancake graphs, ...

Simple Approach for „good“ Peer-to-Peer Topologies

- Naor & Wieder: *The Continuous-Discrete Approach*

- Simplified version:
  - Nodes join at random position in **[0,1)**
  - Connects to points **x/2** and **(1+x)/2**
  - If there is no node, rounding is necessary („continuous => discrete“)
  - Details less interesting here

- Result: a kind of de Bruijn Graph
  - constant degree
  - logarithmic diameter
  - simple routing



Routing

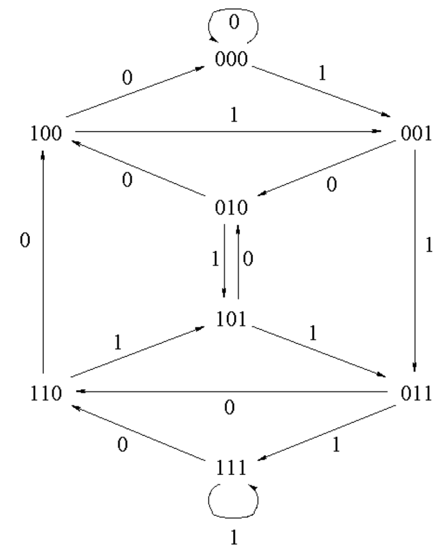
- Naor & Wieder: *The Continuous-Discrete Approach*

- Node u at binary position (**0**....)

  u = **11010111**

  to node

  v = **01000101**



- Ideal path over points:

111010111

011101011

101110101

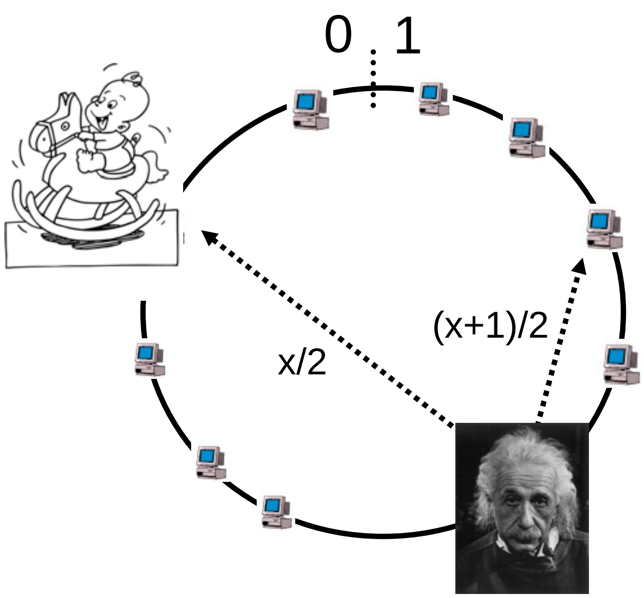
...

01000101

Assumption: All nodes k = log n bit positions, correct one bit per step („next last “ of v).

Idea and Problem

- Network Entry Point assigns random position [0,1)...
- ... and then build topology according to Continuous-Discrete Approach!
- Problem?

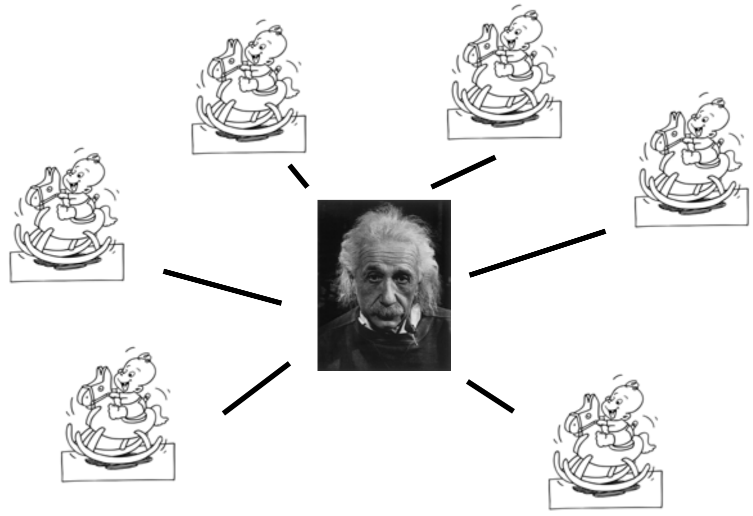


There could be a younger node there!

Solution?

Solution and another Problem

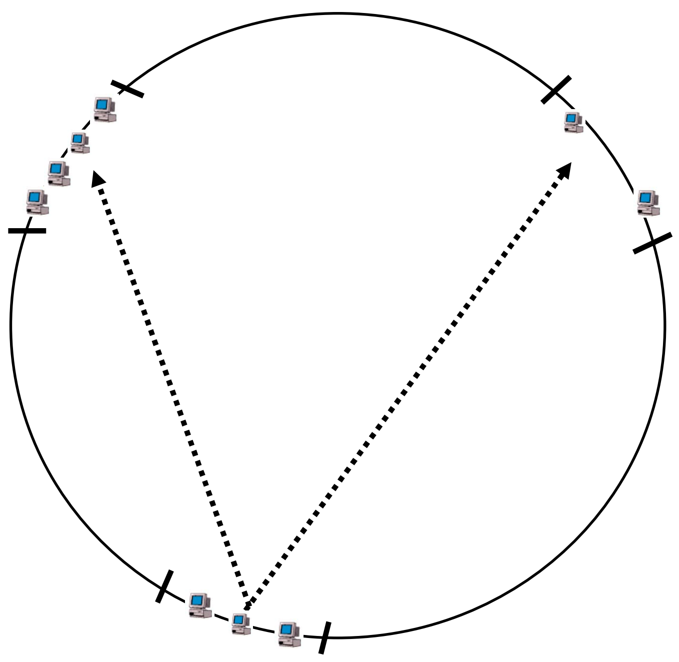
- Connect to corresponding older node close to this position
- Everything solved? Other problems?
- Analysis shows, that older nodes can be congested, as everybody tries to connect to them!



Idea?

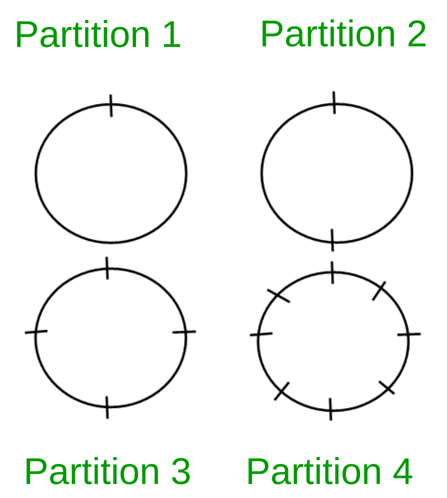
Redundancy

- Solution: we connect to more than one node!
- Allows us to „load-balance“



The Algorithm (1)

- Assume, each node u knows n_u = # living nodes that are older (can be estimated, see later)
- Divide [0,1) Circle in fixed Intervals / Levels of exponentially decreasing sizes



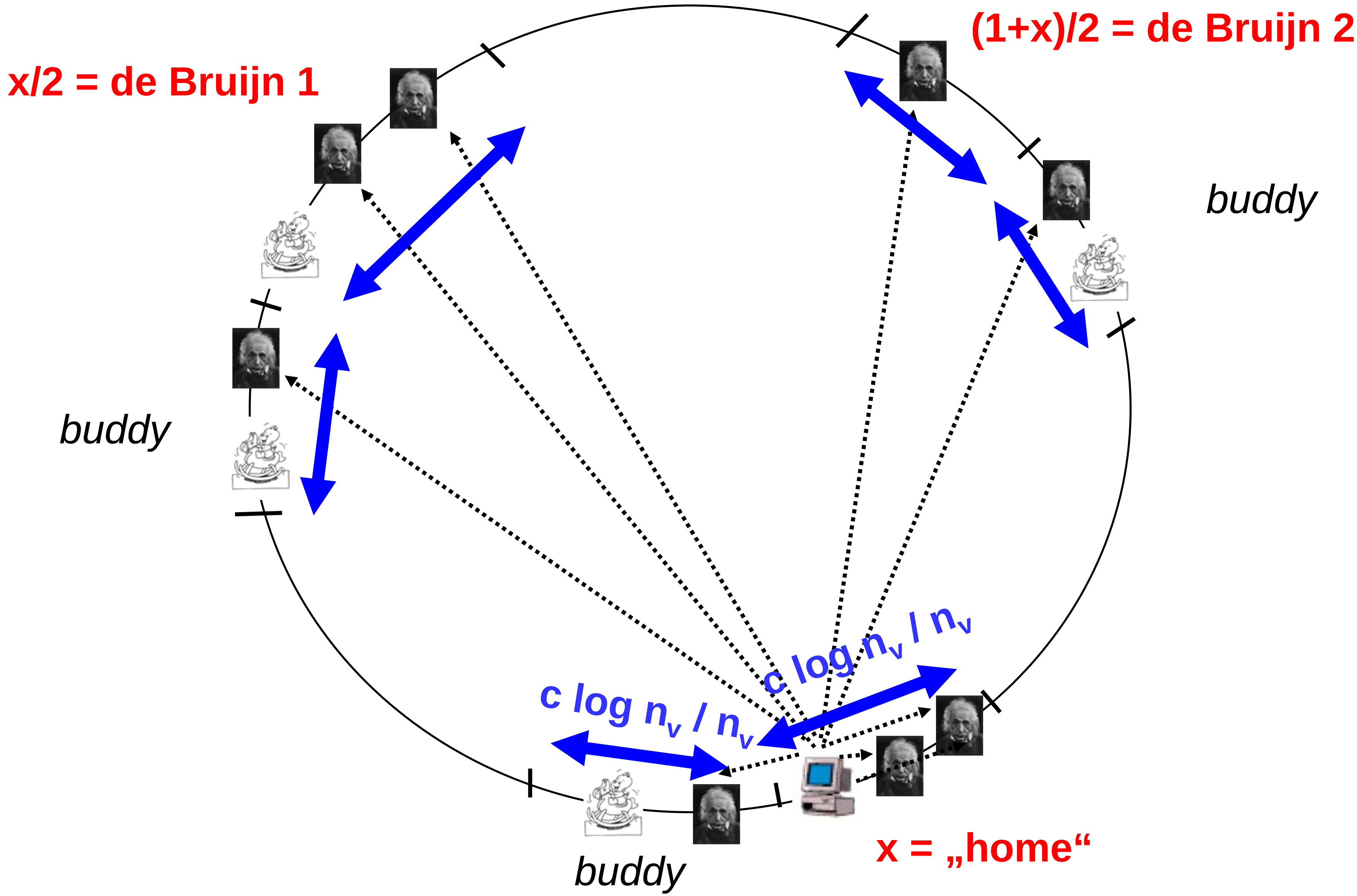
The Algorithm (2)

- Node v connects to three Intervals
  - **$I_{v,0}$** & buddy: Home-interval with position x
    plus other half of the (i-1)-interval
  - **$I_{v,1}$** & buddy: Interval with position x/2, plus buddy
  - **$I_{v,2}$** & buddy: Interval with position (1+x)/2, plus buddy

- Interval is chosen such that it includes at least **c log n_v** older nodes (c = const.)! (If not possible, set level to 0.)

- Establish forward edges to these nodes. Store all incoming edges as backward edges!

Overview „Forward Edges“



Oblivious Structure

Note:

Our distributed heap structure is oblivious

- Node positions independent of join/leave history
- This is an advantage for dynamic systems: allows for efficient joins/leaves!

... how efficient is the system?!?!

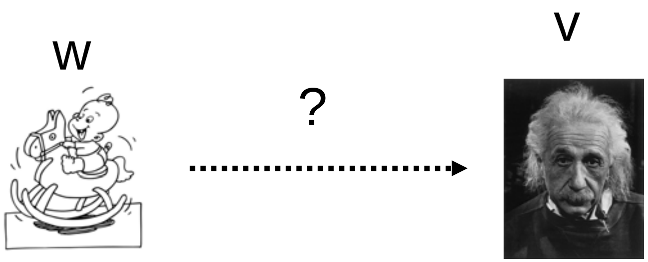
Forward Edge Degree

- Number of forward edges?
- Interval chosen s.t. at least c log n_v older nodes contained
- Number of nodes in interval binomially distributed
- In total there are 3 intervals with 3 buddies, so 6 intervals
- Chernoff: **O(log n_v)** older nodes, w.h.p.

Forward Degree is logarithmic in number of older nodes currently alive, w.h.p.!

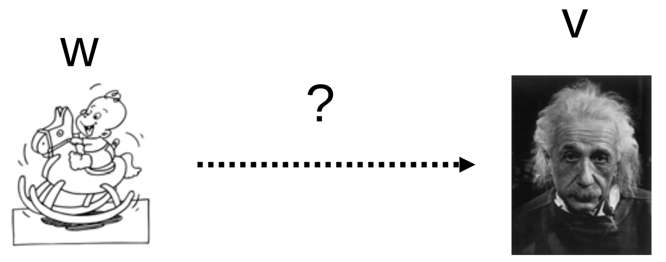
Backward Node Degree (1)

- Nodes also store incoming edges...
- Number of backward edges?
- Which node is expected to have highest in-degree?
- Alive node v with smallest rank (oldest)...
  How to compute?
- Prob that a node w has a connection to v?



Backward Node Degree (2)

- Probability that w has a connection to v?



- Node w connects to intervals of size at most

$$\mathbf{2\ c\ \log n_w / n_w}$$

w.h.p.

As w connects to 6 intervals (incl. buddies) of this size, the probability is

$$\mathbf{P[w\ connects\ to\ v] \leq 12\ c\ \log n_w / n_w}$$

Backward Node Degree (3)

- So in total?

$$
\begin{aligned}
\mathbf{E[In\text{-}degree\ of\ w]} &= \sum_{w \in V} \mathbf{12\ c\ log\ n_w / n_w} \\
&\leq \sum_{i=1}^{n} \mathbf{12\ c\ log\ n / i} \\
&\in \mathbf{O(c\ log^2\ n)}
\end{aligned}
$$

Backward degree / in-degree is in $O(\log^2 n)$ w.h.p.!

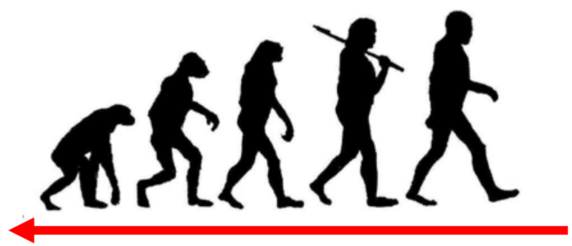
Routing

- Goal: Routing „as usual“ in de Bruijn graphs (fixing bits)
- „Slogan“: Use forward edges as long as possible:

  Thus independent of younger nodes!

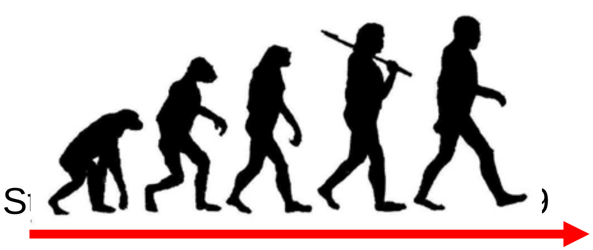
Idea

Phase 1: Along „Forward Edges“ to olders



Phase 2 (if destination not reached yet):

„Descent“ to younger nodes



Routing: Phase 1 (1)

- Recall: de Bruijn Routing

- Node u at binary position (**0**....)

  u = **11010111**

  to node

  v = **01000101**

- Ideal path via:

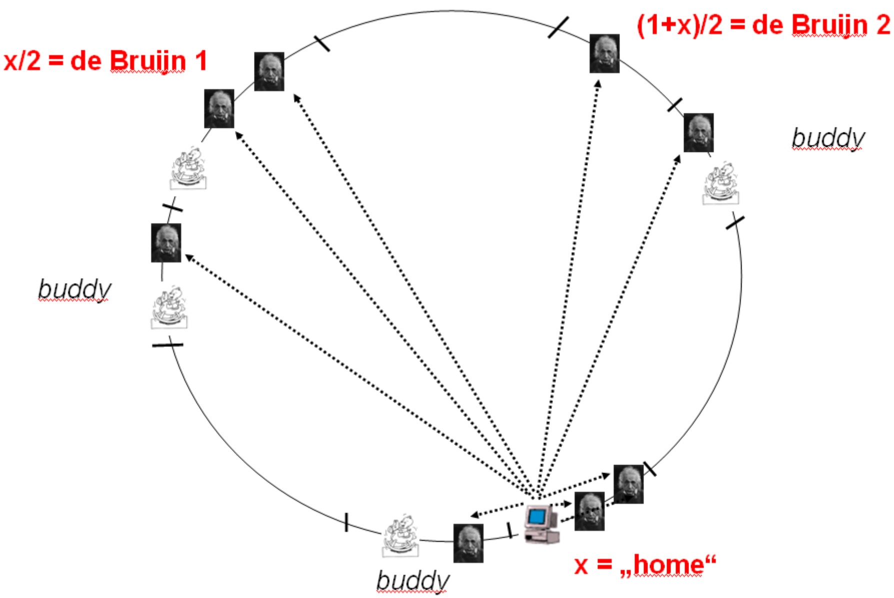
  z_1 = 111010111

  z_2 = 011101011

  z_3 = 101110101

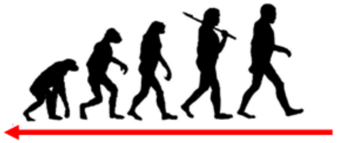
  ...

  z_t = 01000101



We have connections to entire intervals! For load balancing, apply the following strategy in step i: Forward Message to youngest node reachable with forward edges, Whose home interval contains z_i.

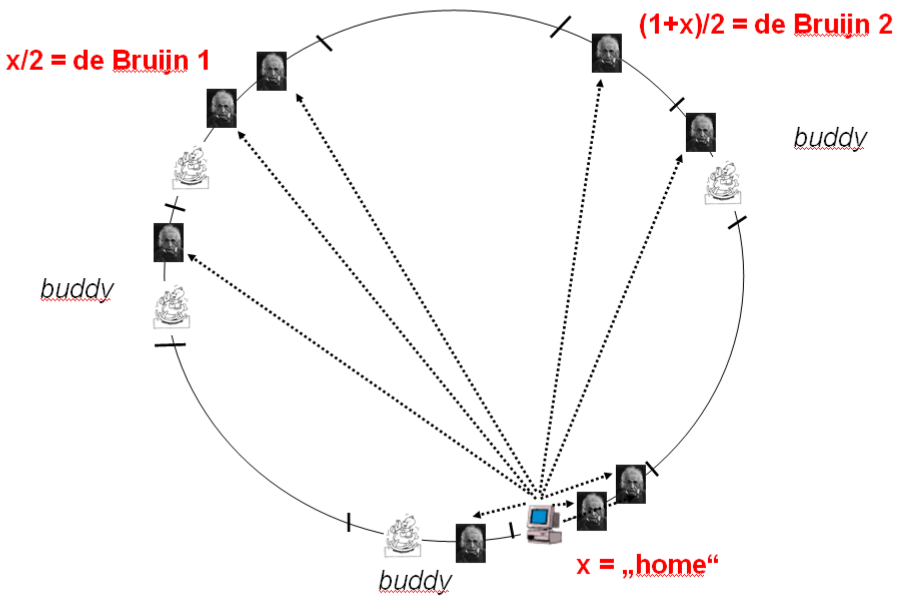
Routing: Phase 1 (2)



In other words, a node sends a message to its youngest older neighbor whose interval contains the theoretic de Bruijn position („emulation“). Thus, older nodes are not overloaded!!

From de Bruijn property it follows: Phase 1 takes at most log hops!

... but what about congestion?!



Routing: Phase 1 (3)

Claim: In Phase 1, a packet of a node u ends at a node whose order is not larger than $n_u/2$!

From this, it follows that the congestion is small...

Let δ_i be the difference of the order of node u and of the node reached after the i-th hop.

What is the probability, that the first node has order $\mathbf{n_u - \delta_1}$?

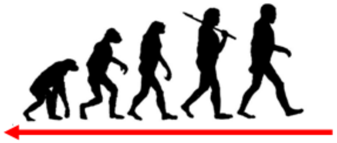
Probability that all younger older nodes are not in the corresponding interval!

Constant factor as interval sizes can differ (random process)!

$$\alpha_0 \cdot \frac{c \log n_u}{n_u} \cdot \left(1 - \frac{c \log n_u}{n_u}\right)^{\delta_1 - 1}$$

Probability that node of corresponding order is in this interval!

Routing: Phase 1 (4)



Claim: In Phase 1, a packet of a node u ends at a node whose order is not larger than $n_u/2$!

For general i?

$$\alpha_i \cdot \frac{c \log n_{u_i}}{n_{u_i}} \cdot \left(1 - \frac{c \log n_{u_i}}{n_{u_i}}\right)^{\delta_{i+1}-\delta_i-1}$$

Analoguous....

Claim: In Phase 1, a packet of a node u ends at a node whose order is not larger than $n_u/2$!

We know that phase 1 requires at most log many steps.

What is the probability, that there is a **$\delta_k > n_u/2$**?

(This would constitute a contradiction to the claim.)

Let δ_k be the first δ_i, with this property.

Path has at most log n_u many hops.

$$\sum_{k=1}^{\log n_u} \sum_{\delta_1,\ldots,\delta_k} \prod_{i=0}^{k-1} \alpha_i \cdot \frac{c \log n_{u_i}}{n_{u_i}} \cdot \left(1 - \frac{c \log n_{u_i}}{n_{u_i}}\right)^{\delta_{i+1}-\delta_i-1}$$

All possible divisions...

Probability of a given δ_i sequence!

Routing: Phase 1 (5)

Claim: In Phase 1, a packet of a node u ends at a node whose order is not larger than $n_u/2$!

How many ways are there to select the δ_i's?

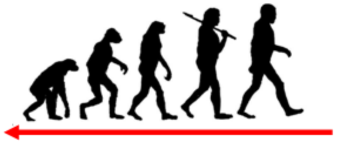
(The first k-1 are hence smaller than $n_u/2$...)

$$(n_u/2)\binom{n_u/2}{k-1}$$

A larger one...

k-1 smaller ones...

Routing: Phase 1 (6)



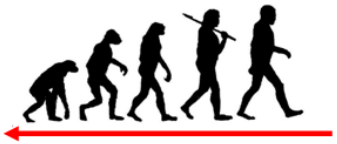
Claim: In Phase 1, a packet of a node u ends at a node whose order is not larger than $n_u/2$!

Long calculations show.....

$$
\begin{aligned}
& \sum_{k=1}^{\log n_u} \sum_{\delta_1,\ldots,\delta_k} \prod_{i=0}^{k-1} \alpha_i \cdot \frac{c \log n_{u_i}}{n_{u_i}} \cdot \left(1 - \frac{c \log n_{u_i}}{n_{u_i}}\right)^{\delta_{i+1}-\delta_i-1} \\
\leq\ & \sum_{k=1}^{\log n_u} \sum_{\delta_1,\ldots,\delta_k} 2^{\log n_u} \prod_{i=0}^{k-1} \frac{c \log n_{u_i}}{n_{u_i}} \cdot \exp\left[-\left(\delta_{i+1}-\delta_i-1\right) \frac{c \log n_{u_i}}{n_{u_i}}\right] \\
\leq\ & \sum_{k=1}^{\log n_u} \sum_{\delta_1,\ldots,\delta_k} n_u \prod_{i=0}^{k-1} \frac{c \log n_u}{n_u/2} \cdot \exp\left[-\left(\delta_{i+1}-\delta_i-1\right) \frac{c \log n_{u_i}}{n_{u_i}}\right] \\
\leq\ & \sum_{k=1}^{\log n_u} \sum_{\delta_1,\ldots,\delta_k} 2n_u \prod_{i=0}^{k-1} \frac{c \log n_u}{n_u/2} \cdot \exp\left[-\left(\delta_{i+1}-\delta_i\right) \frac{c \log n_u}{n_u}\right] \\
\leq\ & \sum_{k=1}^{\log n_u} \sum_{\delta_1,\ldots,\delta_k} 2n_u \left[\frac{c \log n_u}{n_u/2}\right]^k \cdot \exp\left[\sum_{i=0}^{k-1} -\left(\delta_{i+1}-\delta_i\right) \frac{c \log n_u}{n_u}\right] \\
\leq\ & 2n_u \sum_{k=1}^{\log n_u} (n_u/2) \binom{n_u/2}{k-1} \left[\frac{c \log n_u}{n_u/2}\right]^k \cdot \exp\left[-\delta_k \frac{c \log n_u}{n_u}\right] \\
\leq\ & n_u^2 c \log n_u \sum_{k=1}^{\log n_u} \left(\frac{n_u/2}{k-1}\right)^{k-1} \left[\frac{c \log n_u}{n_u/2}\right]^{k-1} \cdot \exp\left[-\frac{n_u}{2} \cdot \frac{c \log n_u}{n_u}\right] \\
\leq\ & n_u^2 c \log n_u \sum_{k=1}^{\log n_u} \left(\frac{ec \log n_u}{k-1}\right)^{k-1} e^{-c \log n_u/2} \\
\leq\ & n_u^2 c \log n_u \sum_{k=1}^{\log n_u} (ec)^{\log n_u} \cdot e^{-c \log n_u/2} \\
\leq\ & n_u^2 c \log^2 n_u \cdot e^{-c \log n_u/4} \in O(n_u^{-c/8}).
\end{aligned}
$$

With high probability this is not the case. So the claim is true!

Routing: Phase 1 (7)

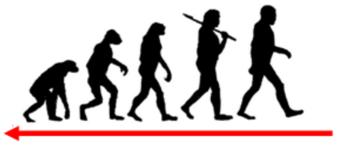


Why is this good for congestion?

How to measure the congestion?

Let‘s define a **Random Routing Problem**:
Each node wants to send a message to *one random other node.*
Congestion = Number of messages pass a given node?
(in expectation or w.h.p.)

Routing: Phase 1 (8)



Why is this good for congestion?

More interesting: Number of messages through v *w.h.p.?*

Which nodes send a message over node v?

We know: w.h.p. only those, whose order is between 2 n_v and n_v!

How long are these paths?

Due to de Bruijn strategy: $\log(2\,n_v)$, over intervalls of size at most $2\,c\,\log n_v / n_v$, w.h.p.

Probability that in the i-th hop, one comes across an intervall of v is at most $2\,c\,\log n_v / n_v$. So the expected number is:

$$n_v \log(2n_v) \cdot \frac{2c \log n_v}{n_v} \in O(c \log^2 n_v)$$

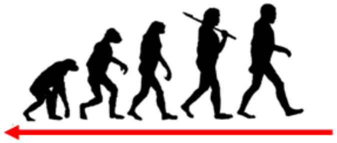
„sender“

Path length

prob in i-th hop

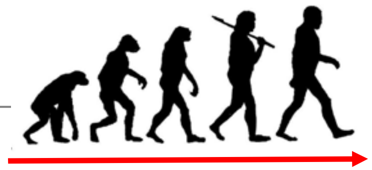
All holds also w.h.p.!

Routing: Phase 1 (9)

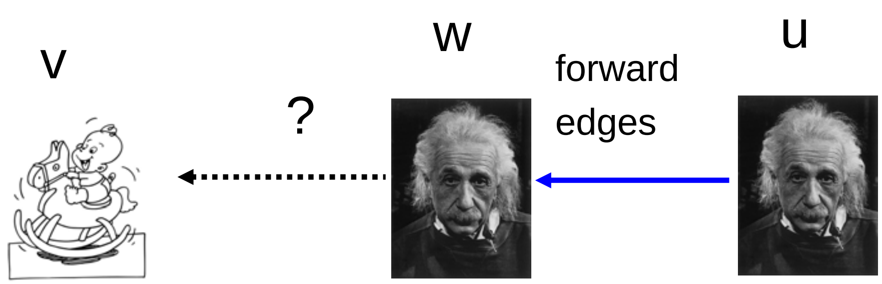


Congestion in phase 1 is thus at most $O(\log^2 n)$!

Routing: Phase 2 (1)

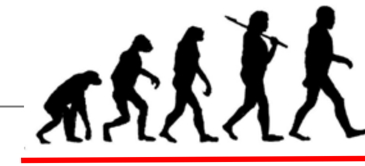


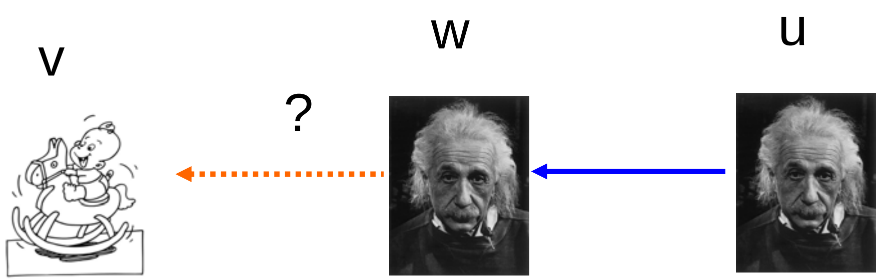
- But what if an old node wants to send something to a young one?!



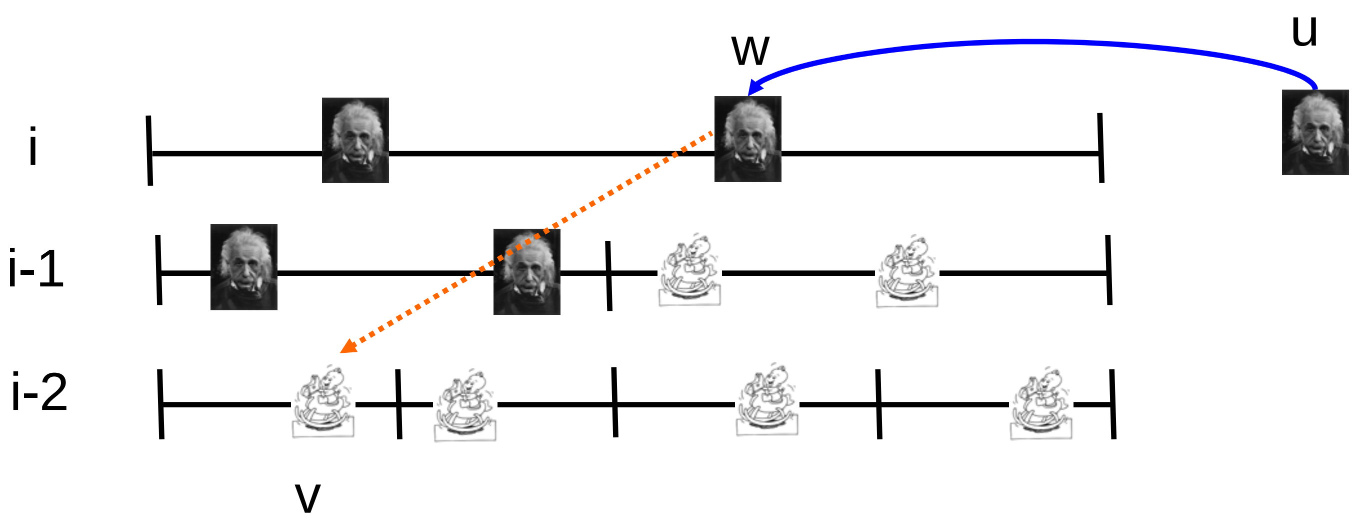
- Second Routing Phase
  - Phase 1: as long along Forward Edges until a node is reached whose interval includes v.
  - Then Phase 2: Backward Edges are allowed (give up invariant)!

Routing: Phase 2 (2)

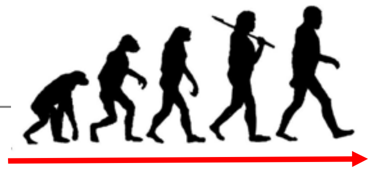


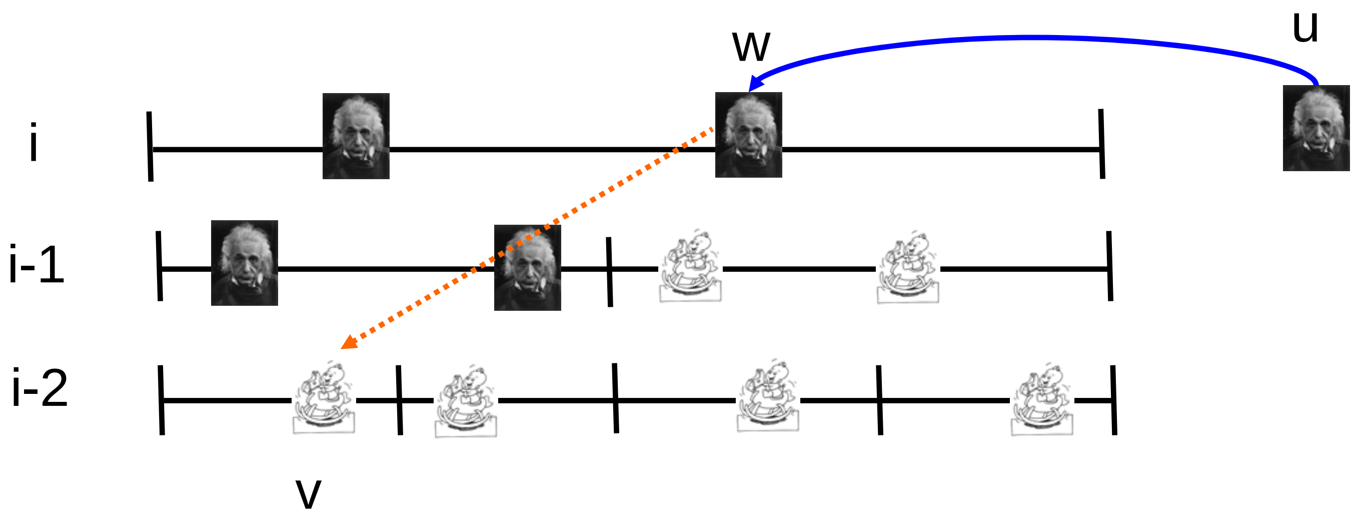


- Goal: Send down the right level of v, the node there must know v!



Routing: Phase 2 (3)





- One can show: w always has an edge to a node in an interval closer to v. By this „binary search“ v can be reached in logarithmic time (and low congestion).

Join / Leave

- Another feature: efficient Join and Leave

- If many nodes leave, some intervals need to be merged

- Result (w/o proof):

Join in time O(log n) and affects at most $O(\log^2 n)$ edges.

Leave in time O(1) and with $O(\log^2 n)$ many edge changes.

Estimation of n_v (1)

- Nodes must estimate n_v locally, to find level
  - Recall: We want at least c log n_v alive older nodes in an interval
  - Problem: n_v is a global variable!

- Idea: Sampling

Locally observed!

Let B(j) be the number of older alive nodes in level-j interval;
Then increase j if **j>B(j)/c – log B(j)**, and
decrease j if **j<B(j)/c – log B(j)**.

- The level with i = B(i)/c – log B(i) is „good“, almost as if one knew n_v!

Estimation of n_v (2)

Let B(j) be the number of older alive nodes in level-j interval;
Then increase j if $\mathbf{j > B(j)/c - \log B(j)}$, and
decrease j if $\mathbf{j < B(j)/c - \log B(j)}$.

- Why is $\mathbf{i = B(i)/c - \log B(i)}$ good?
- In „the ideal case“ each level j contains the same number of nodes (assumption uniform failures!): $\mathbf{B(j) = n_v/2^j \Rightarrow n_v = B(j)\, 2^j}$
- Let $\mathbf{B(j) = \alpha\, c \log n_v}$ for a α.
- Then also $\mathbf{B(j)/\alpha = c \log n_v = c \log(2^j\, B(j))}$
  and hence:
  $$\mathbf{j = B(j)/(\alpha\, c) - \log B(j)}$$

This function has a unique extremal value => search possible!

Schätzung von n_v (3)

Let B(j) be the number of older alive nodes in level-j interval;
Then increase j if **j>B(j)/c – log B(j)**, and
decrease j if **j<B(j)/c – log B(j)**.

- But: World not ideal!

  - Intervals have not the same number of nodes

  - Variations in binomial distributed random variables:

    $$\mathbf{B(j) = (1 \pm \delta)\, \alpha\, c \log n_v}$$

- Thus:

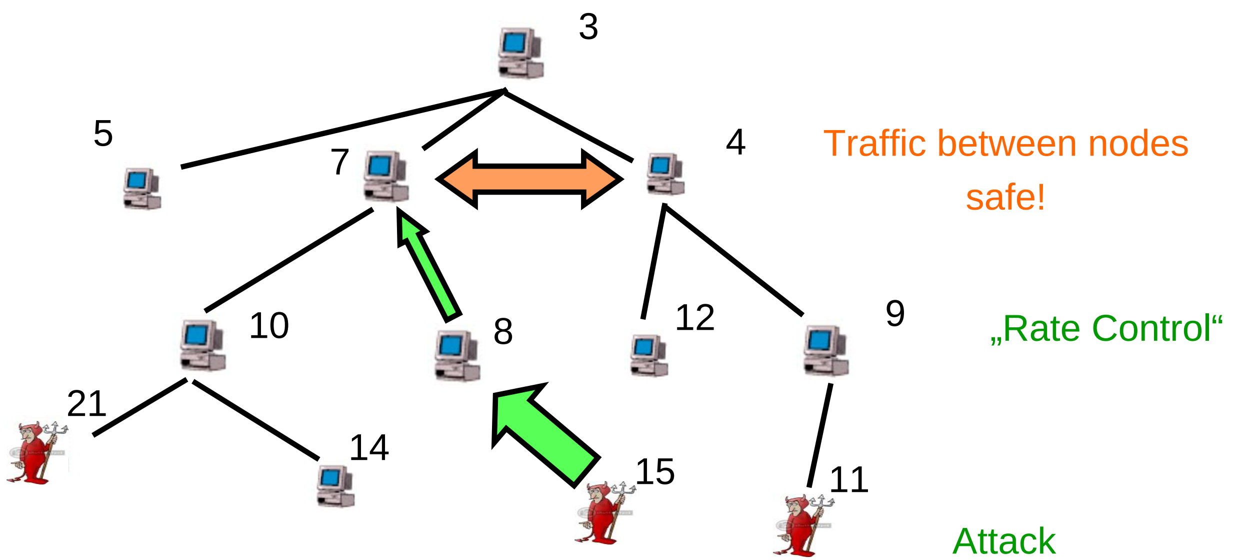
$$n_v = \frac{B(j)}{1 \pm \delta} \cdot 2^j$$

Schätzung von n_v (3)

Let B(j) be the number of older alive nodes in level-j interval;
Then increase j if **j>B(j)/c – log B(j)**, and
decrease j if **j<B(j)/c – log B(j)**.

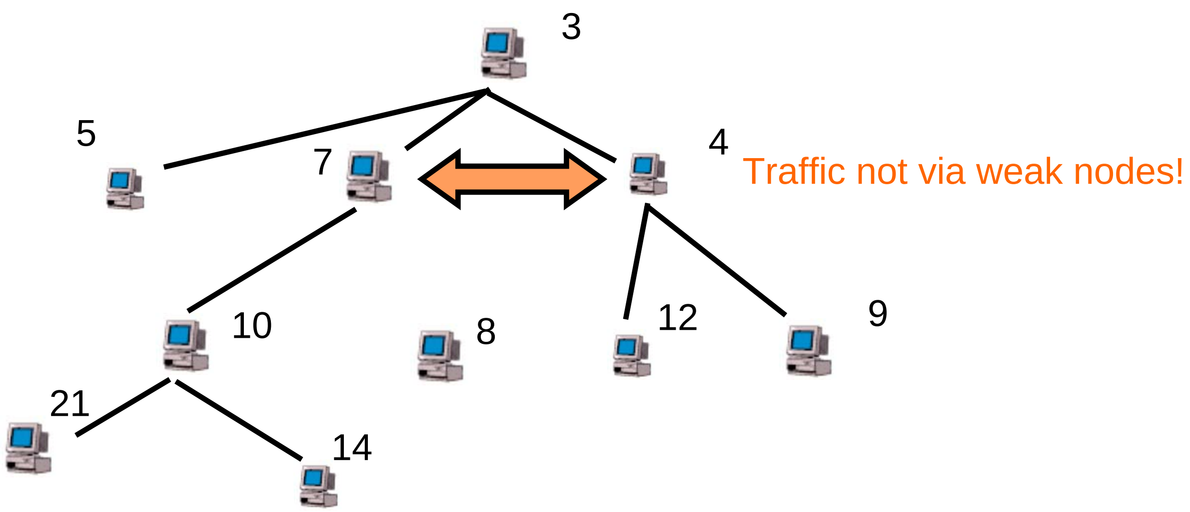
- Thus $$\frac{1}{\alpha(1\pm\delta)}B(j) = c\log n_v = c(j+\log(B(j)/(1\pm\delta))$$
- And hence $$j = \frac{B(j)}{(1\pm\delta)\alpha c} - \log(B(j)/(1\pm\delta))$$
- Since δ is an arbitrarily small constant according to Chernoff, level j is at most by 1 away from the ideal level!
- Details: see paper!

Applications: Sybil Attacks



Applications: Heterogeneous Systems

Idea: Order = Inverse of quality of Internet connection



More Literature about the SHELL System

TUM

INSTITUT FÜR INFORMATIK

A Distributed and Oblivious Heap

Christian Scheideler, Stefan Schmid

TUM-I0908
April 09

http://www.cs.uni-paderborn.de/fachgebiete/fg-ti/personen/schmiste.html

Extra Slides

Routing: Phase 1 (1)

Why is this good for congestion?

Expected number of messages through v?

Let $s = c \log n_v / n_v$ be the size of the home interval. On a given de Bruijn path of length k intervals $I_0, I_1, ..., I_k$ are visited. For how many nodes u does this path lead through v?

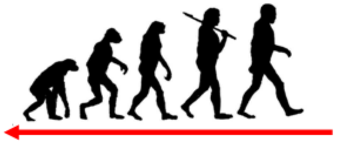
When is the path $u = u_0, u_1, ..., u_k = v$ valid?

If in an interval I_i there is no node between order u_{i-1} and u_i!
Let $m = n_u – n_v$, then how many paths go through v?

$$\sum_{m \geq k} \binom{m-1}{k-1} s^k (1-s)^{m-k}$$

Over all order differences,
count the number of possibilities that
the k (hop-)nodes are in the right
Interval and the „unwanted nodes“ not!

Routing: Phase 1 (2)



Why is this good for congestion?

Expected number of messages through v?

One can show:

$$\sum_{m>k} \binom{m-1}{k-1} s^k (1-s)^{m-k} = O\left(\frac{s^k}{(1-s)^k (k-1)!} \cdot \frac{(k-1)!}{s^k} e^{-s(k-1)}\right) = O(1)$$

Thus: for a constant number only! ☺

From this, we can show that the expected number is at most logarithmic:
A random de Brujin path has probability **WSK $1/2^k$**, and is used by a constant number of nodes (see above); a path has a **logarithmic** number of hopes.