Wie designt man robuste Netzwerke?
Connect to the Seniors!

Randomisierte Algorithmen für Netzwerke

Eine „Exkursion“

Heute Spezialprogramm

- Prof. Scheideler auf Rhodos

- Nächste Woche geht‘s normal weiter!

Populäre Netzwerke heute

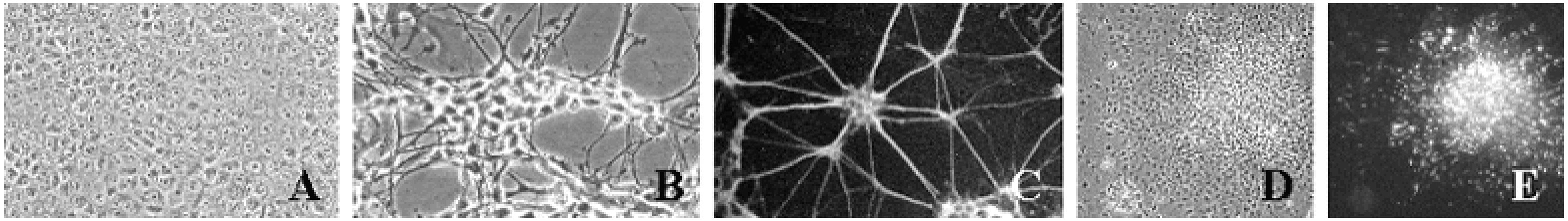
Soziale Netzwerke...

... Peer-to-Peer Netzwerke ...

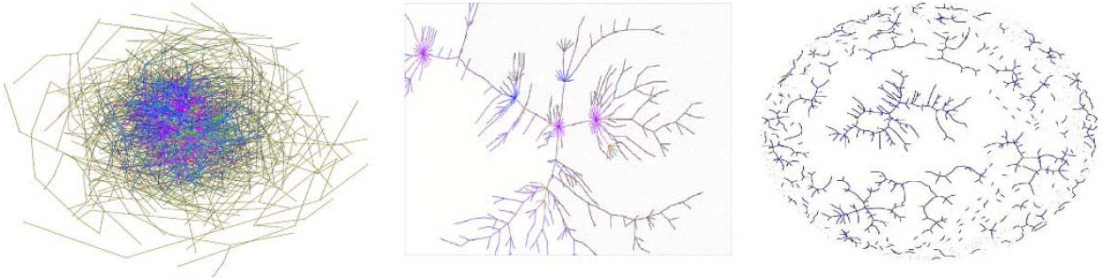
etc.

Viele interessante Eigenschaften

- Z.B. das „**Kleine Welt Phänomen**“
  - Jeder kennt jeden über kurze Ketten von Bekannten
  - Aber z.B. auch: Nervennetz von gewissen Tieren



- Z.B. topologische Eigenschaften von Gnutella
  - Toleriert zufällige, aber nicht schlimmstmögliche Ausfälle



Eine weitere, nützliche Eigenschaft? (1)

- Häufig sind **ältere Teilnehmer** im Netzwerk zuverlässiger!

- Z.B. Messungen in Peer-to-Peer Netzwerken:

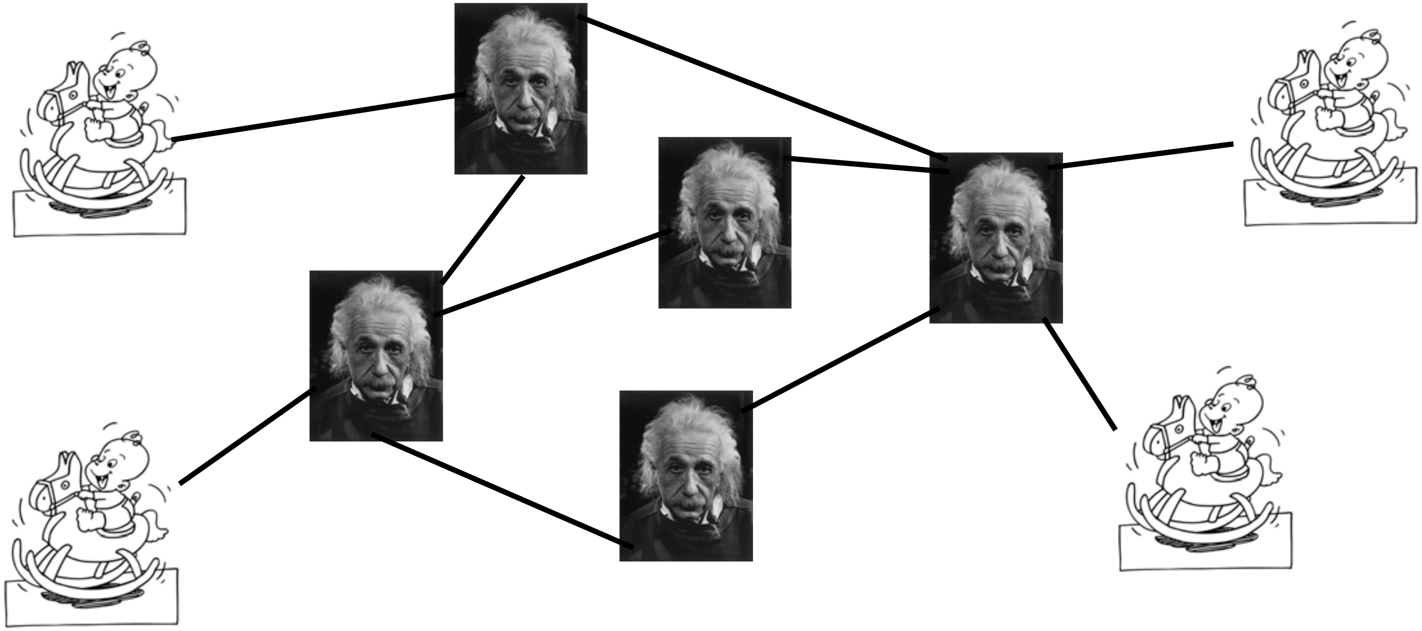
  - Wer schon länger „online“ ist, bleibt typischerweise auch länger
  - Ältere Nachbarn sind also typischerweise stabiler

- Z.B. soziale Netzwerke?

  - Wer schon lange „gute“ Wikipedia Beiträge geschrieben hat, wird es mit höherer Wahrscheinlichkeit auch in Zukunft tun?

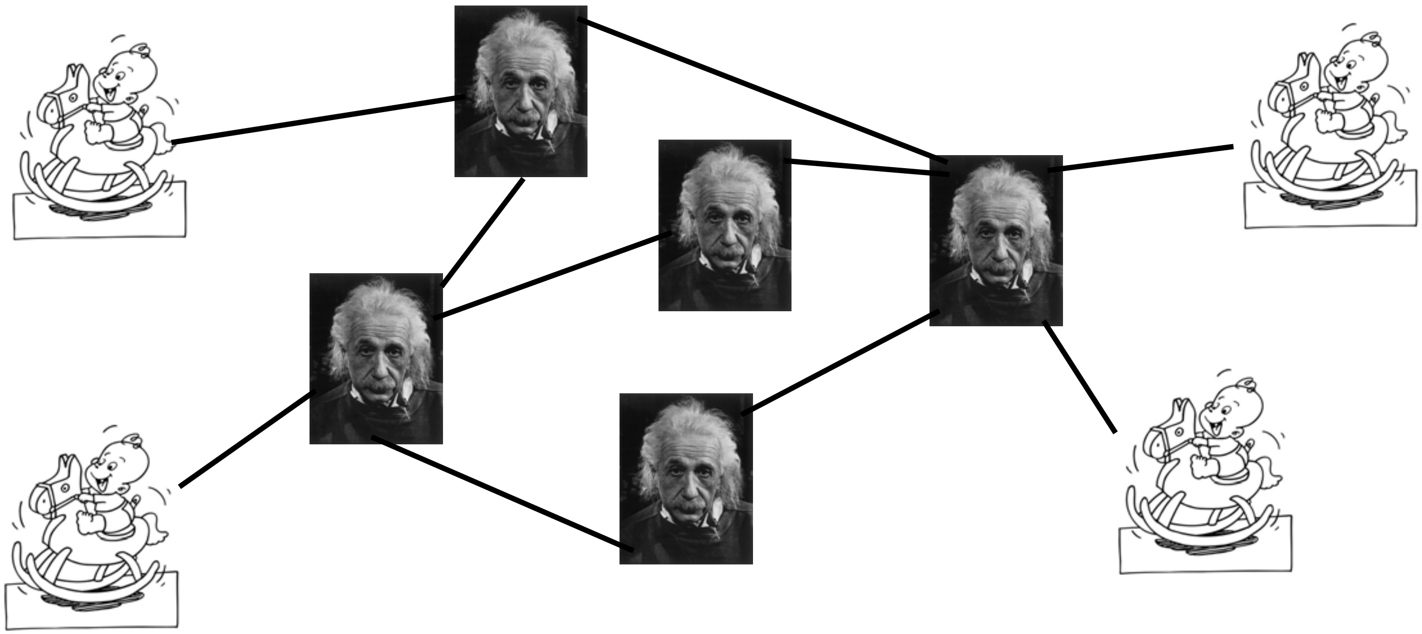
Eine weitere, nützliche Eigenschaft? (2)

- Idee, wenn jeder nur zu älteren Teilnehmern verbinden würde:
  - ... hätte man stabilere Nachbarschaften!
  - ... wäre man geschützt gegen Angriffe durch „junge Störefriede“



Eine weitere, nützliche Eigenschaft? (3)

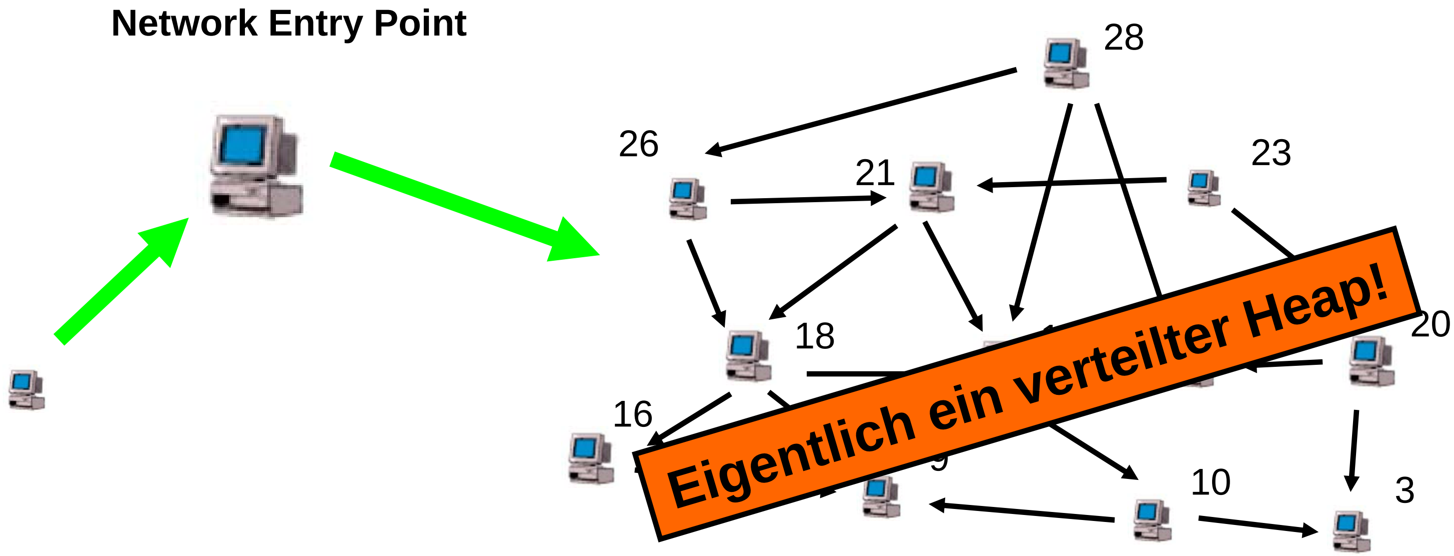
- Implikationen:
  - Kommunikationspfade der „Seniors“ gehen nie über Jüngere
  - Jüngere können Netzwerk nicht überfluten (Rate Control vor „Core Network“)



Modell

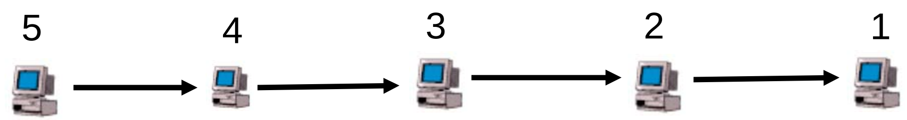
- Wie kann man ein solches Netzwerk designen?

- Idee: Ein zentraler Server weist bei „Join“ jedem Teilnehmer seinen Rang zu
  - Knoten verbinden nur zu früher angekommenen (kleinerer Rang)



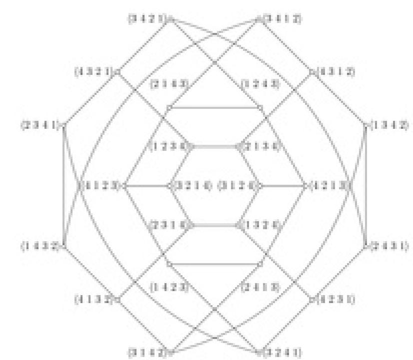
Eine naive Lösung

- Unser Ziel lässt sich leicht erreichen:



- Problem: Skalierbarkeit
  - Grosser Durchmesser, nicht robust bei Join/Leave, etc.!

- Bessere Topologien: Hyperwürfel, de Bruijn Graphen, Pancake Graphen, ...

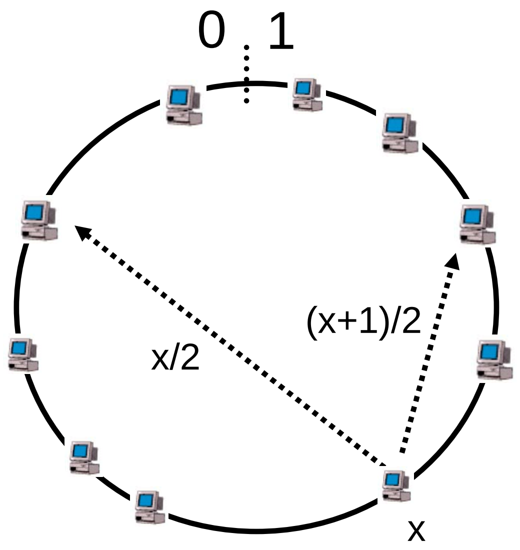


Einfacher Ansatz für „gute“ Peer-to-Peer Topologien

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

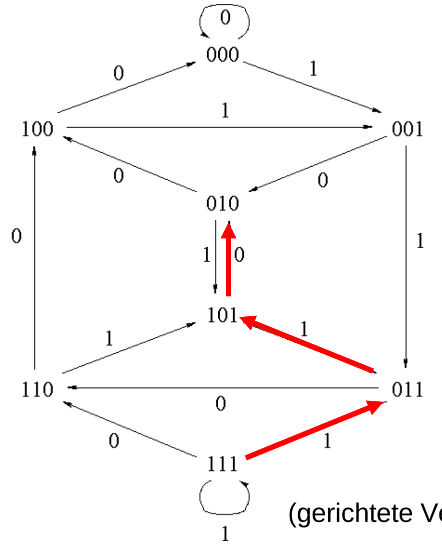
- Vereinfachte Version:
  - Jeder Knote hat eine Position in **[0,1)**
  - Verbindet zu Position **x/2** und **(1+x)/2**
  - Da dort ev. kein Knote ist, muss man noch runden („continuous => discrete“)
  - Details hier weniger interessant

- Es resultiert eine Art de Bruijn Graph
  - konstanter Grad (Grad 2)
  - logarithmischer Durchmesser
  - einfaches Routing



Routing

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



(gerichtete Version!)

- Knote u an binärer Position (**0....**)

  u = **11010111**

  zu Knote

  v = **01000101**

- Idealer Weg über Punkte:

  111010111

  011101011

  101110101

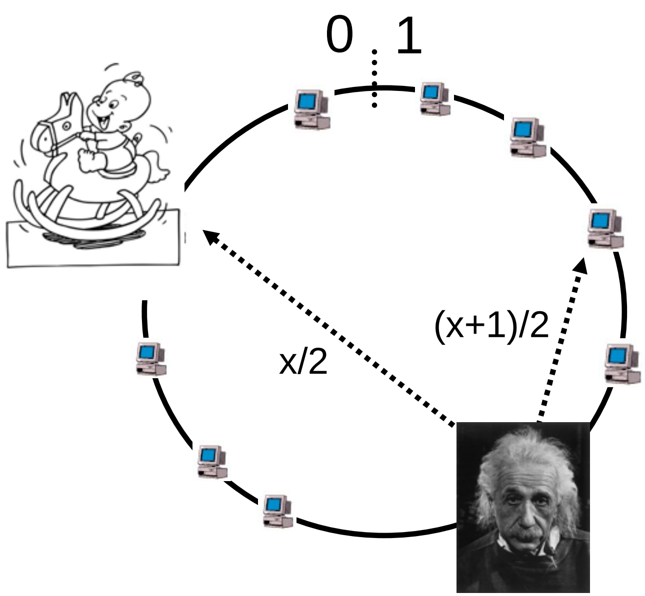
  ...

  01000101

Annahme: Alle Knoten k = log n Bit Positionen, korrigiere in jedem Schritt ein Bit („nächsthinterstes“ von v).

Idee und Problem

- *Network Entry Point* weist jedem Knoten zufällige Position [0,1) zu...
- ... dann bilde Topologie gemäss *Continuous-Discrete Approach*!
- Problem?



An jener Position könnte ein jüngerer Knote sein!

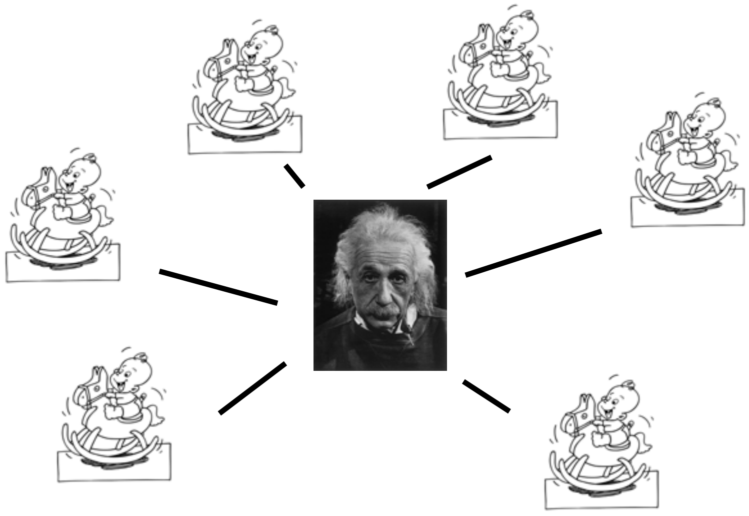
Lösungsidee?

Lösung und noch ein Problem

- Verbinde zu entsprechendem, älterem Knoten nahe an jenem Punkt

- Alles gelöst nun? Mögliche Probleme?

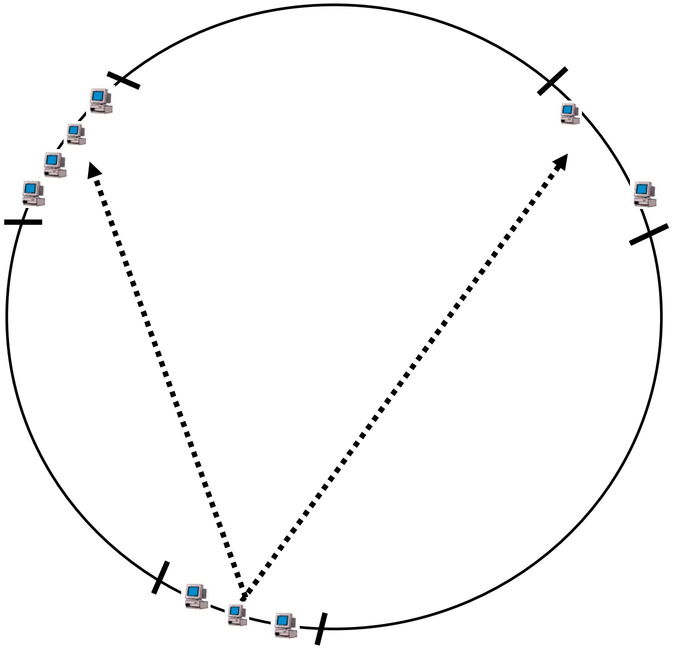
- Analyse zeigt, dass bei älteren Knoten Staus auftreten können (z.B. beim Ältesten insgesamt), da alle eher zu ihnen verbinden!



Idee?

Redundanz

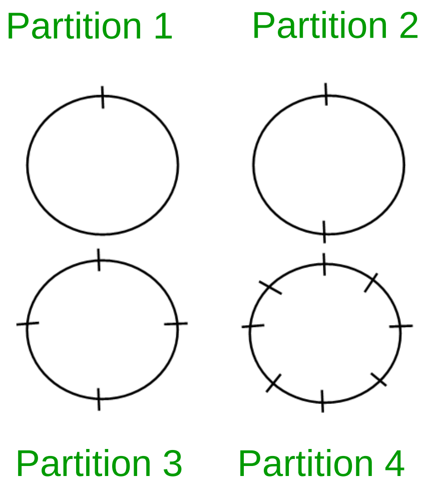
- Lösung: wir verbinden zu mehr als nur einem Knoten!
- Erlaubt „Load-Balancing“ gegen Congestion!



Der Algorithmus (1)

- Nehme an, jeder Knote u kennt n_u = aktuelle Anzahl älterer Knoten (kann man schätzen, siehe später)

- Teile [0,1) Kreis in fixe Intervalle / Levels exponentiell kleinerer Grössen



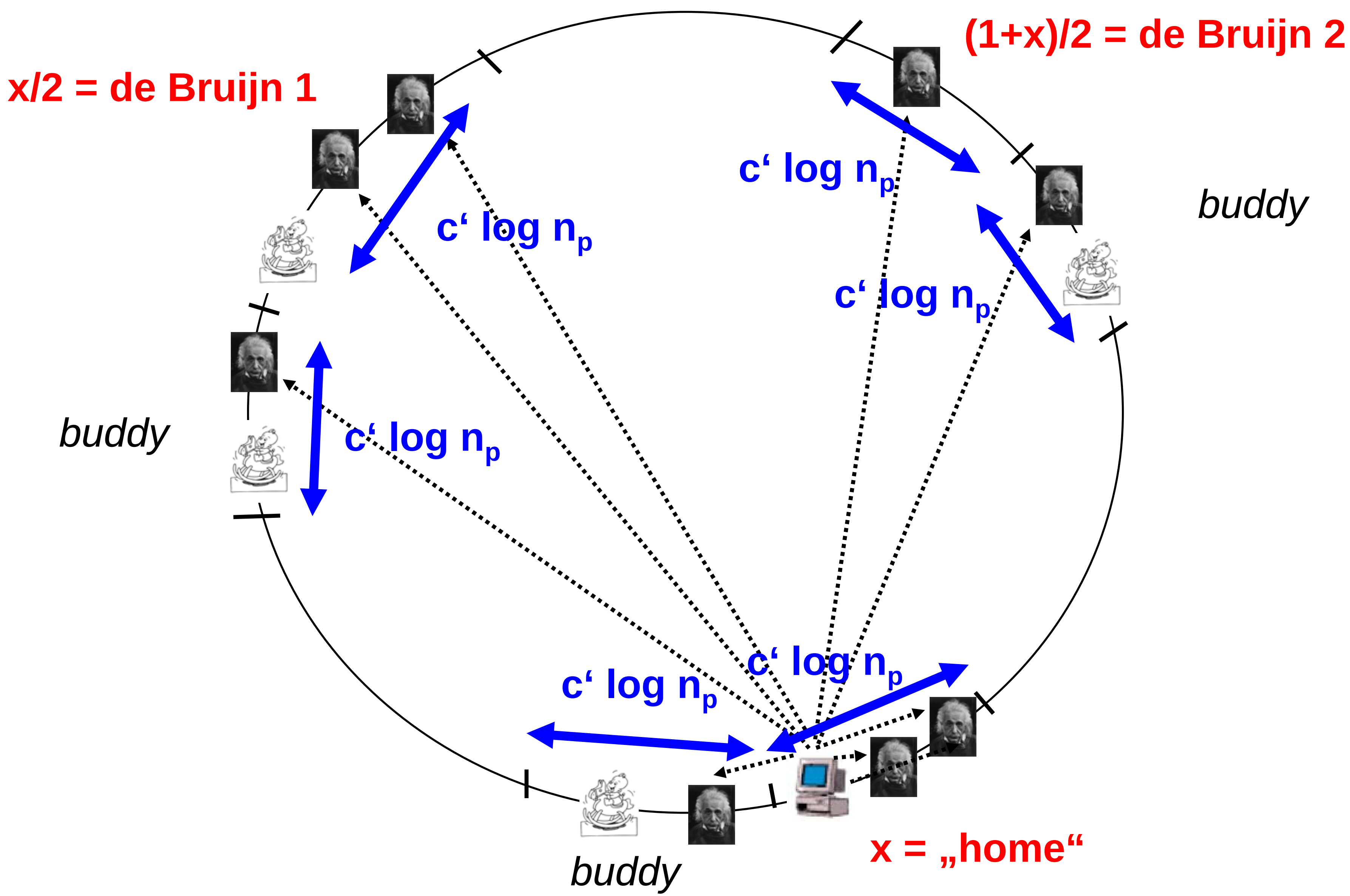
Der Algorithmus (2)

- Knote v verbindet sich zu drei Intervallen
  - $\mathbf{I_{v,0}}$ & buddy: Home-interval mit Position x
    plus andere Hälfte des (i-1)-Intervalls
  - $\mathbf{I_{v,1}}$ & buddy: Intervall mit Position x/2, plus Buddy
  - $\mathbf{I_{v,2}}$ & buddy: Intervall mit Position (1+x)/2, plus Buddy

- Das Intervall wird so gewählt, dass es mindestens **c log n_v** ältere Knoten (c = const.) drin hat! (Falls nicht möglich, setze Level auf 0.)

- Zu diesen Älteren, mache Forward-Kanten. Speichere alle eingehenden Kanten als Backward-Kanten!

Überblick „Forward Edges“



... was hat das bloss mit
„randomisierten Algorithmen“ zu tun?!

Viel!

- Topologie ist zufällig!

  - Recall: Knotenposition ist uniform zufällig aus [0,1) gewählt!

- „Durch Zufall“ lassen sich einfach und „lokal“ gute Eigenschaften erreichen

  - keine aufwendige „Buchhaltung“ am Network Entry Point

- Routing/Congestion, etc.

- Beachte auch: Unsere verteilte Heapstruktur ist oblivious

  - Knotenposition unabhängig von Join-Leave-Historie in der Vergangenheit

  - Das ist ein grosser Vorteil in dynamischen Systemen: effizientes Join/Leave möglich!

... wie effizient ist das System?!

Der Forward-Knotengrad

- Anzahl Forward Edges eines Knoten?
- Intervall so gewählt dass mind. $c \log n_v$ ältere Knoten drin sind
- Die Anzahl Knoten in einem Intervall ist binomialverteilt
- Total gibt's 3 Intervalle mit 3 Buddies, also 6 Intervalle
- Mit Chernoff folgt einfach, dass $\mathbf{O(\log n_v)}$ ältere Knoten total, w.h.p.

Der Forward Grad ist logarithmisch bezgl. Anzahl älterer Knoten aktuell im Netzwerk, w.h.p.!

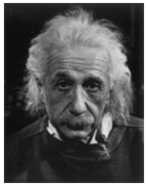
Der Backward-Knotengrad (1)

- Knoten speichern auch eingehende Forward-Kanten...
- Anzahl Backward Edges eines Knoten?
- Betrachte Knoten mit höchstem erwarteten Backward Grad! Nämlich?
- Lebender Knote v mit kleinstem Rang (aller ältester)...
  Wie berechnen?
- Wahrscheinlichkeit dass ein anderer Knote w eine Verbindung zu v hat?

w

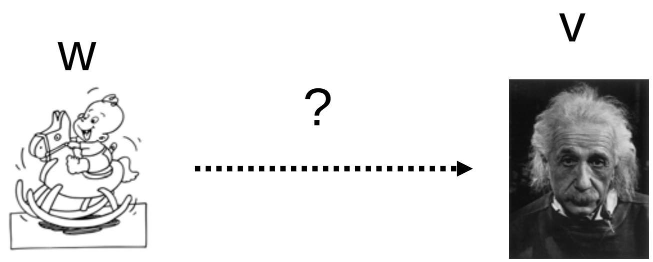
?

v



Der Backward-Knotengrad (2)

- Wahrscheinlichkeit dass ein anderer Knote w eine Verbindung zu v hat?



- Knote w verbindet zu Intervall der Grösse höchstens

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

w.h.p.

Da w zu 6 Intervallen (inkl. Buddies) dieser Grösse verbindet, ist Wahrscheinlichkeit

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

Der Backward-Knotengrad (3)

- Also insgesamt?

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

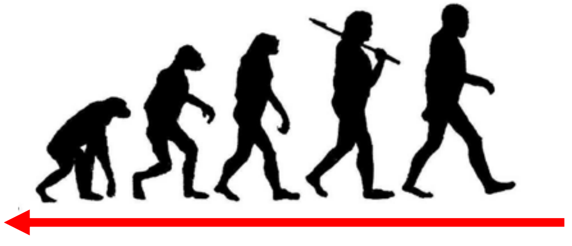
Der Backward Grad / In-degree ist in $O(\log^2 n)$ w.h.p.!

Routing

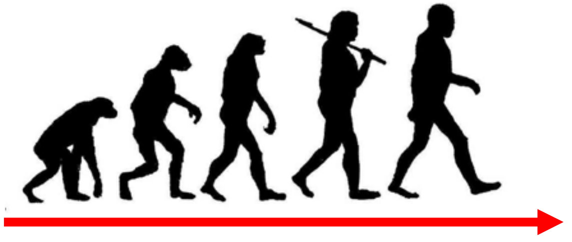
- Ziel: Routing „wie normal“ bei de Bruijn Graphen („Bits fixen“)
- Maxime: Solange wie möglich nur Forward Kanten benutzen:
  Dadurch unabhängig von jüngeren Knoten!

Idee

Phase 1: Entlang „Forward Edges“ zu älteren



Phase 2 (falls Ziel noch nicht erreicht):
„Abstieg“ zu jüngeren Knoten



Routing: Phase 1 (1)

- Recall: de Bruijn Routing

- Knote u an binärer Position (**0**....)

  u = **11010111**

  zu Knote

  v = **01000101**

- Idealer Weg über Punkte:

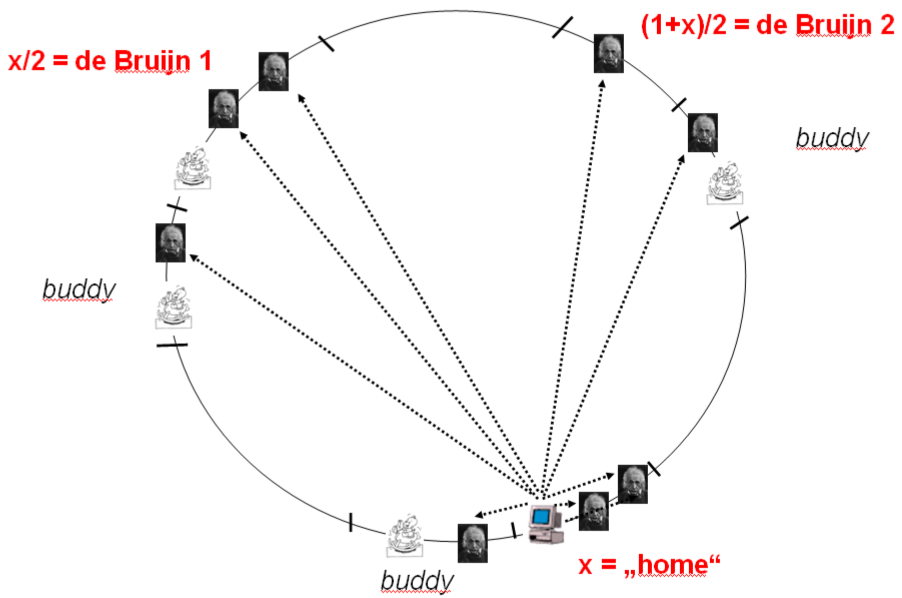
  $\mathbf{z_1}$ **= 111010111**

  $\mathbf{z_2}$ **= 011101011**

  $\mathbf{z_3}$ **= 101110101**

  ...

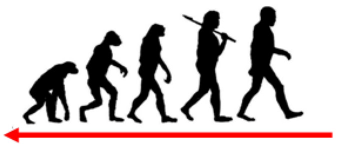
  $\mathbf{z_t}$ **= 01000101**



Wir haben Verbindungen zu ganzen Intervallen! Zum Load-Balancing verwende folgende Strategie im Schritt i:

Forwarde Nachricht zum jüngsten mit Forward Kante erreichbaren Knoten, dessen Home Interval z_i enthält.

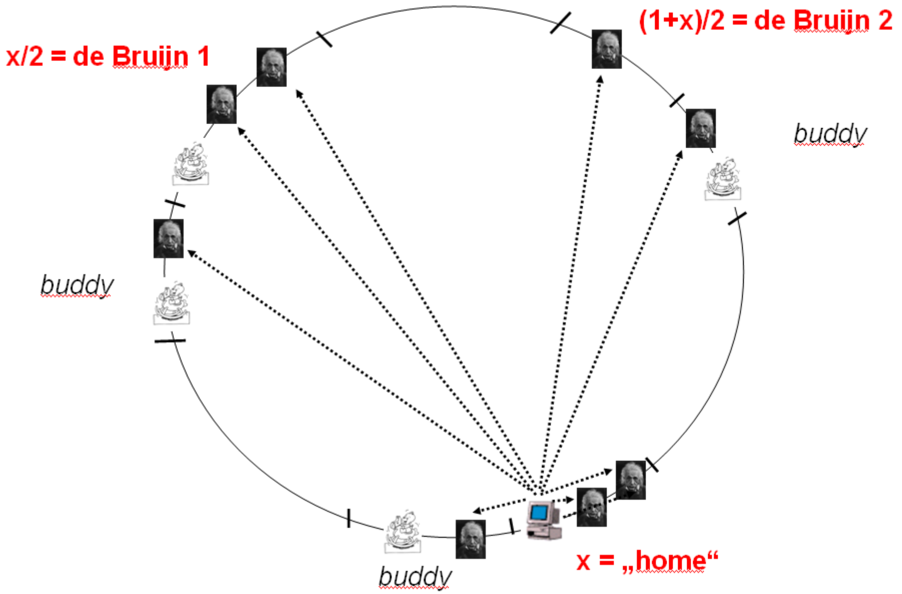
Routing: Phase 1 (2)



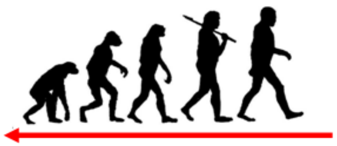
Mit anderen Worten, ein Knote sendet die Nachricht an seinen *jüngsten älteren Nachbarn*, dessen Intervall die theoretische de Bruijn Position enthält („Emulation“).
Dadurch werden ältere Knoten nicht überlastet!

Aus de Bruijn Theorie folgt:
Phase 1 braucht maximal log Hops!

... but what about congestion?!



Routing: Phase 1 (3)



Behauptung: In Phase 1, ein Paket von einem Knoten u endet bei einem Knoten dessen Ordnung nicht grösser als $n_u/2$ ist!

Wie wir sehen werden folgt daraus, dass die Congestion klein ist...

Sei δ_i die Differenz der Ordnung vom Knoten u und dem Knoten nach dem i-ten Hop.

Was ist also die Wahrscheinlichkeit, dass der erste erreichte Knoten Ordnung $\mathbf{n_u - \delta_1}$ hat?

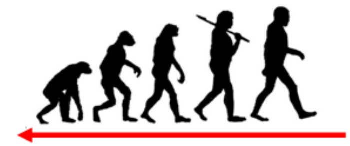
WSK dass alle jüngeren älteren Knoten nicht im entsprechenden Intervall sind!

Konstanter Faktor weil Intervallgrössen sich unterscheiden können (Zufallsprozess!)

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

WSK dass Knote dieser Ordnung im entsprechenden Intervall

Routing: Phase 1 (4)



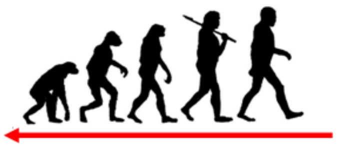
Behauptung: In Phase 1, ein Paket von einem Knoten u endet bei einem Knoten dessen Ordnung nicht grösser als $n_u/2$ ist!

Für allgemeine i?

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

Analog....

Routing: Phase 1 (4)



Behauptung: In Phase 1, ein Paket von einem Knoten u endet bei einem Knoten dessen Ordnung nicht grösser als $n_u/2$ ist!

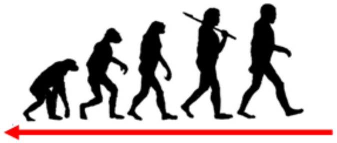
Wir wissen, insgesamt hat Phase 1 höchstens logarithmisch viele Schritte.
Wie gross ist die Wahrscheinlichkeit, dass es ein **$\delta_k > n_u/2$** gibt?
(Dies wäre ein Widerspruch zur Behauptung oben!)
Sei δ_k das erste δ_i, das das erfüllt.

Der Pfad hat maximal log n_u viele Hops.

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

Alle möglichen Aufteilungen...

Wahrscheinlichkeit einer gewissen
\delta_i Folge!



Behauptung: In Phase 1, ein Paket von einem Knoten u endet bei einem Knoten dessen Ordnung nicht grösser als $n_u/2$ ist!

Wieviele Möglichkeiten gibt es, die δ_i‘s zu wählen?

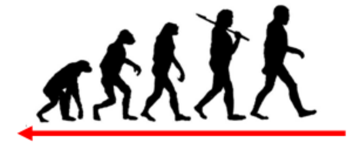
(Die ersten k-1 sind also kleiner als $n_u/2$...)

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

ein grösseres...

k-1 kleinere...

Routing: Phase 1 (6)



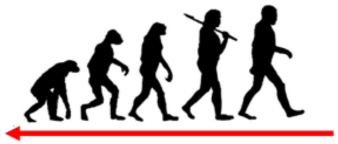
Behauptung: In Phase 1, ein Paket von einem Knoten u endet bei einem Knoten dessen Ordnung nicht grösser als $n_u/2$ ist!

Lange Rechnungen und Abschätzungen zeigen.....

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

Mit hoher WSK ist das nicht der Fall! Also folgt der Claim!

Routing: Phase 1 (7)



Wieso ist das gut für die Congestion?

Messen der Congestion?

Wir definieren das **Random Routing Problem**:
Jeder Knote möchte *einem zufälligen anderen* eine Nachricht schicken.
Congestion = Wieviele Nachrichten gehen dabei durch einen Knoten?
(Im Erwartungswert oder w.h.p.)

Routing: Phase 1 (8)

Wieso ist das gut für die Congestion?

Erwartete Anzahl Nachrichten durch v?

Sei $s = c \log n_v / n_v$ die Home Intervall Grösse. Auf einem fixen de Bruijn Pfad der Länge k werden Intervalle $I_0, I_1, \ldots, I_k$ besucht. Für wieviele Knoten u geht dieser Pfad durch v?

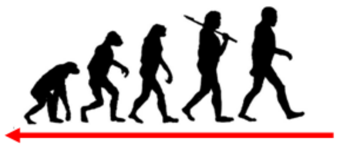
Wann ist Pfad $u = u_0, u_1, \ldots, u_k = v$ gültig?

Wenn es in jedem Intervall I_i keinen Knoten der Ordnung zwischen u_{i-1} und u_i hat!
Sei m = n_u – n_v, dann gehen höchstens wieviele Pfade durch v?

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

Über alle Ordnungsunterschiede, zähle die Möglichkeiten wo die k (Hop-)Knoten im richtigen Intervall und die „Störefriede“ nicht!

Routing: Phase 1 (9)



Wieso ist das gut für die Congestion?

Erwartete Anzahl Nachrichten durch v?

Man kann zeigen:

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

Für konstant viele! ☺

Daraus lässt sich zeigen, dass die erwartete Anzahl höchstens logarithmisch ist: Ein zufälliger de Brujin Pfad hat **WSK 1/2^{k}**, und wird von **konstant** vielen Knoten benutzt (s. oben); ein Pfad hat **logarithmisch** viele Hops.

Routing: Phase 1 (10)

Wieso ist das gut für die Congestion?

Interessanter: Anzahl Nachrichten durch v *mit hoher Wahrscheinlichkeit*?

Welche Knoten schicken Nachricht über Knoten v?

Wir wissen: w.h.p. nur solche, die Ordnung zwischen 2 n_v und n_v haben!
Wie lange sind diese Pfade?

Wege de Bruijn Strategie: $\log(2\, n_v)$, über Intervalle der Grösse maximal $2\, c \log n_v / n_v$, w.h.p.
Die Wahrscheinlichkeit dass man im i-ten Hop über ein Intervall von v kommt ist auch höchstens $2\, c \log n_v / n_v$. Also ist die erwartete Anzahl:

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

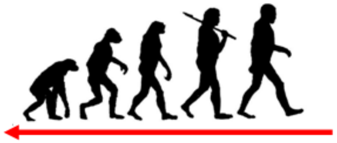
Anzahl „Absender"

Pfadlänge

WSK im i-ten Hop

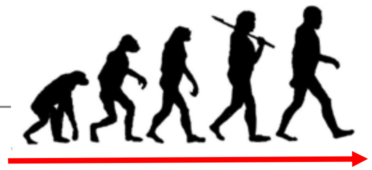
Das gilt nun alles auch w.h.p.!

Routing: Phase 1 (11)

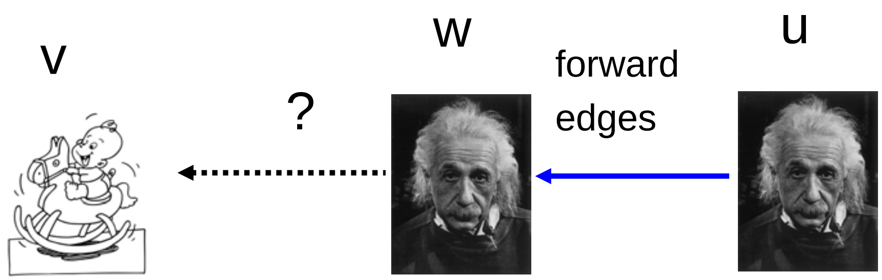


Die Congestion in Phase 1 ist also höchstens $O(\log^2 n)$!

Routing: Phase 2 (1)

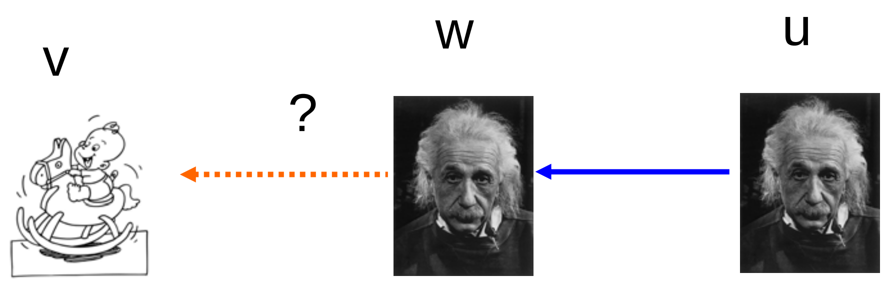


- Was aber nun, wenn z.B. ein alter Knote wirklich mal einem jüngeren etwas schicken will?

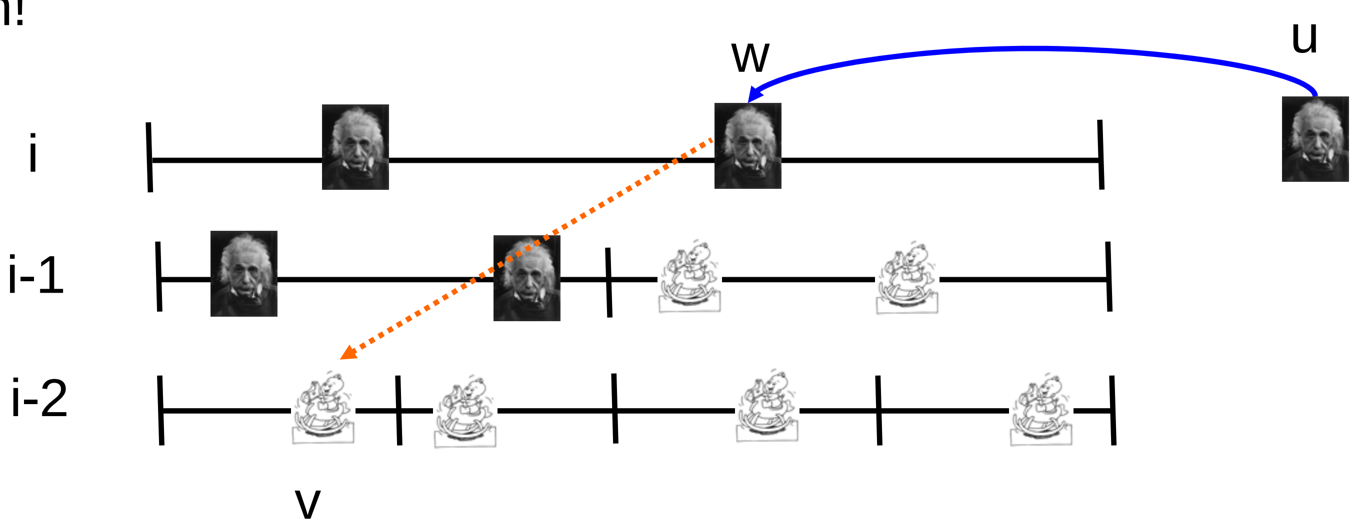


- Zweite Routing Phase
  - Phase 1: solange entlang Forward Edges bis an einem Knoten der im gleichen Intervall ist wie v.
  - Dann Phase 2: Backward Edges erlaubt (gebe Invariante auf)!

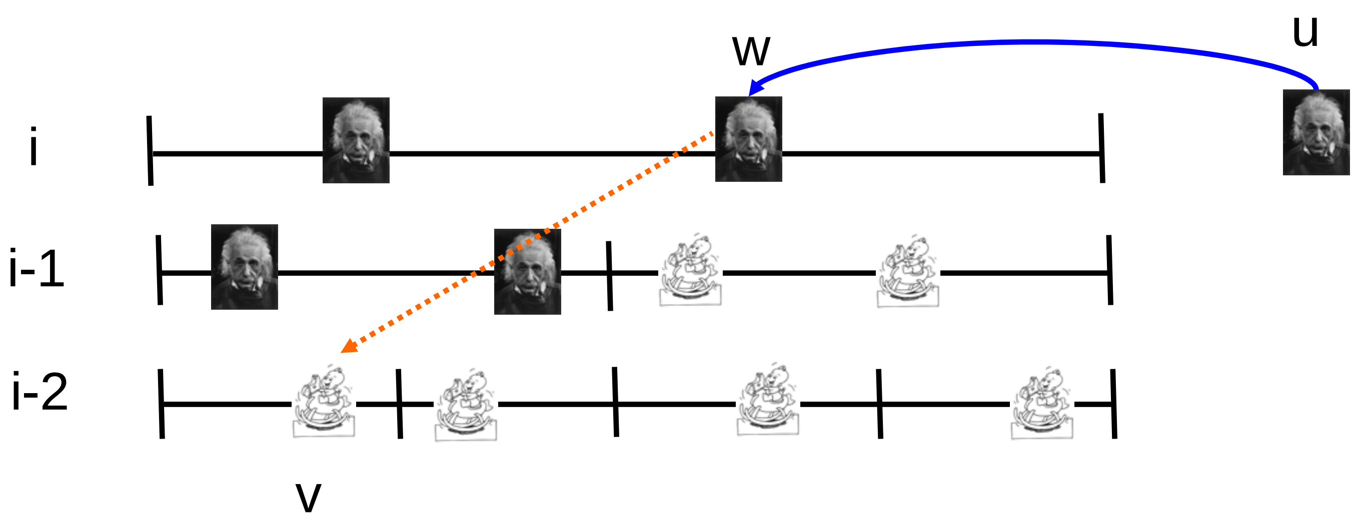
Routing: Phase 2 (2)



- Ziel: Sende ins richtige Level von v hinunter, der Knote dort muss v kennen!



Routing: Phase 2 (3)



- Man kann zeigen: w hat immer eine Kante zu einem Knoten in einem Intervall näher zu v. Durch diese „binäre Suche" kann v erreicht werden in logarithmischer Zeit (und geringer Congestion).

Join / Leave

- Eine weitere wichtige Eigenschaft: effizientes Join und Leave
- Wenn sehr viele Knoten gehen, müssen z.B. Intervalle vereinigt werden
- Resultat (ohne Beweis):

Join in Zeit O(log n) und betrifft maximal $O(\log^2 n)$ viele Kanten.

Leave in Zeit O(1) und mit $O(\log^2 n)$ vielen Kantenänderungen.

Schätzung von n_v (1)

- Knoten müssen lokal n_v schätzen, um Level zu bestimmen

  - Recall: Wir wollen in einem Intervall mindestens c log n_v viele lebende ältere Knoten

  - Problem: n_v ist eine globale Variable!

- Idee: Sampling

Lokal bestimmbar!

Sei B(j) die Anzahl älterer lebender Knoten im Level-j Intervall.
Dann erhöhe j falls **j>B(j)/c – log B(j)**, und
verringere j falls **j<B(j)/c – log B(j)**.

- Das Level mit i = B(i)/c – log B(i) ist „gut“, fast als ob man n_v kennen würde!

Schätzung von n_v (2)

Sei B(j) die Anzahl älterer Knoten im Level-j Intervall.
Dann erhöhe j falls $\mathbf{j>B(j)/c - \log B(j)}$, und
verringere j falls $\mathbf{j<B(j)/c - \log B(j)}$.

- Wieso ist $\mathbf{i = B(i)/c - \log B(i)}$ gut?

- Im „Idealfall“ hat jedes Level j gleichviele Knoten (Annahme uniforme Ausfälle!): $\mathbf{B(j) = n_v/2^j \Rightarrow n_v = B(j)\, 2^j}$

- Sei $\mathbf{B(j) = \alpha\, c \log n_v}$ für ein α.

- Dann ist also $\mathbf{B(j)/\alpha = c \log n_v = c \log(2^j\, B(j))}$
  und somit:

$$\mathbf{j = B(j)/(\alpha\, c) - \log B(j)}$$

Diese Funktion hat einen eindeutigen Extremalwert => Suche möglich!

Schätzung von n_v (3)

Sei B(j) die Anzahl älterer Knoten im Level-j Intervall.
Dann erhöhe j falls **$j>B(j)/c - \log B(j)$**, und
verringere j falls **$j<B(j)/c - \log B(j)$**.

- Aber: Welt ist nicht ideal!
  - Intervalle haben nicht gleich viele Knoten
  - Schwankungen der binomialverteilten Zufallsvariablen:

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

- Also:

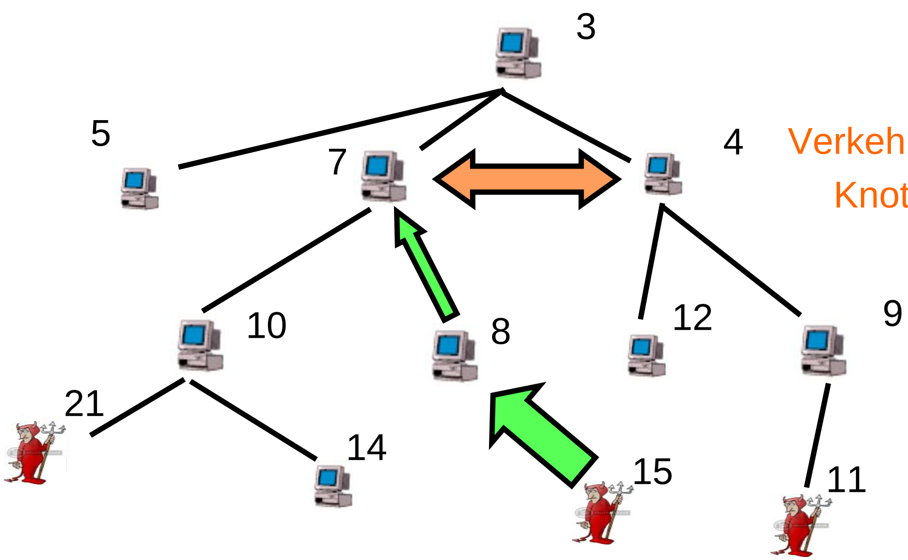
$$n_v = \frac{B(j)}{1 \pm \delta} \cdot 2^j$$

Schätzung von n_v (3)

Sei B(j) die Anzahl älterer Knoten im Level-j Intervall.
Dann erhöhe j falls **j>B(j)/c – log B(j)**, und
verringere j falls **j<B(j)/c – log B(j)**.

- Also $$\frac{1}{\alpha(1\pm\delta)}B(j) = c\log n_v = c(j+\log(B(j)/(1\pm\delta))$$
- Und somit $$j = \frac{B(j)}{(1\pm\delta)\alpha c} - \log(B(j)/(1\pm\delta))$$
- Weil δ gemäss Chernoff eine beliebig kleine Konstante ist, ist das Level j höchstens um 1 vom idealen Level entfernt!
- Details: siehe Paper!

Anwendungen: Sybil Attacke



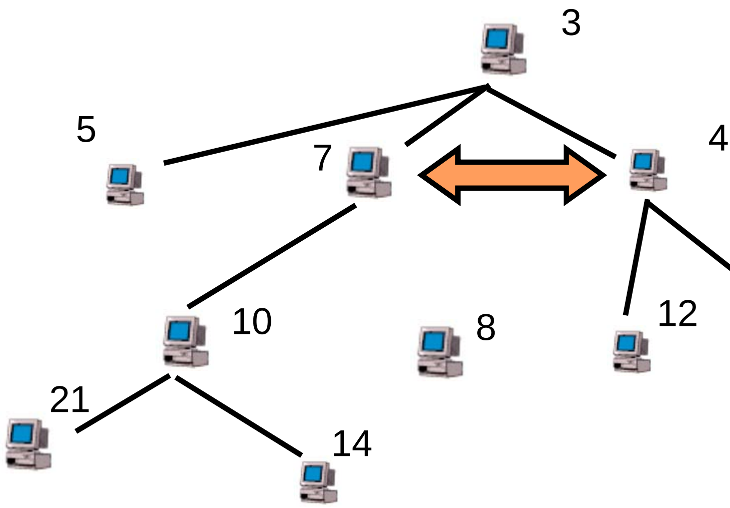
Verkehr zwischen alten Knoten ungestört!

„Rate Control“

Attacke

Anwendungen: Heterogene Systeme

Idee: Ordnung = Inverses der Qualität des Internet Anschlusses



Verkehr zwischen starken geht nicht über schwache!

Weitere Literatur zum sogenannten SHELL System

T U M

INSTITUT FÜR INFORMATIK

A Distributed and Oblivious Heap

Christian Scheideler, Stefan Schmid

TUM-I0908
April 09

http://www14.informatik.tu-muenchen.de/personen/schmiste/SHELL_TR.pdf