

# Vorlesung

## P2P Netzwerke

### 4: Content Addressable Network



Dr. Dominic Battré

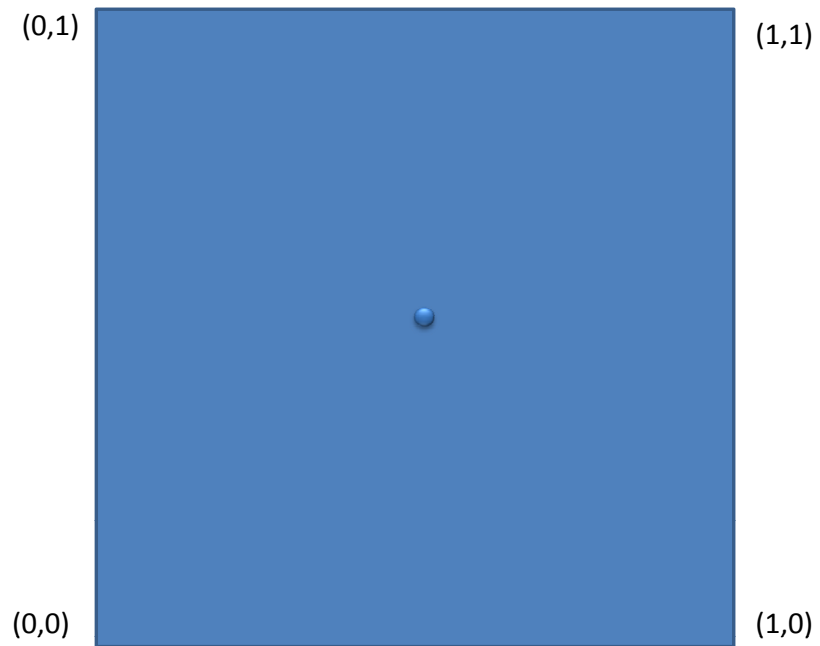
Complex and Distributed IT-Systems

[dominic.battre@tu-berlin.de](mailto:dominic.battre@tu-berlin.de)

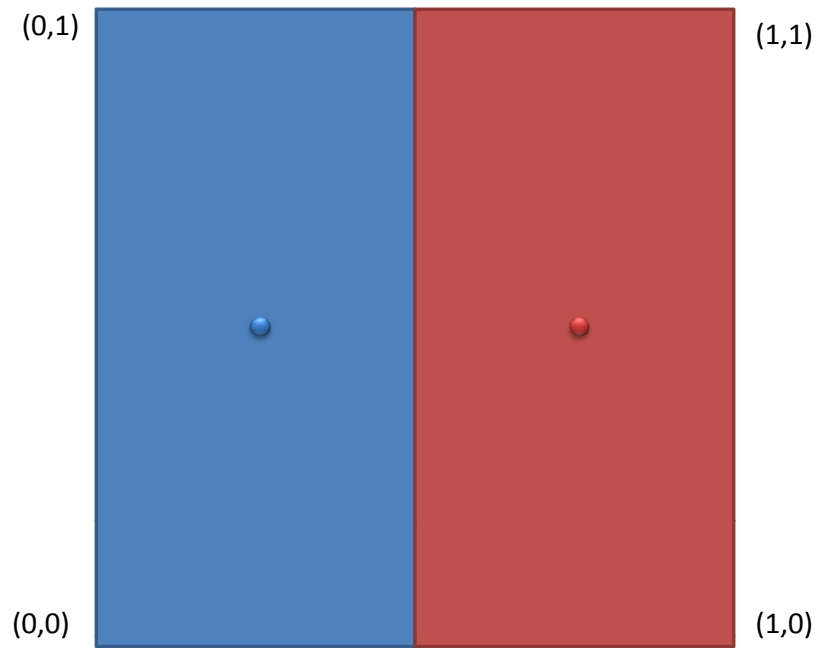
- Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker: "A Scalable Content-Addressable Network", SIGCOMM'01, 2001.
- Alternativer Ansatz:
  - Adresse eines Knotens ist eine 2-dimensionale Koordinate
- Bereiche werden bei Knotenankunft geteilt



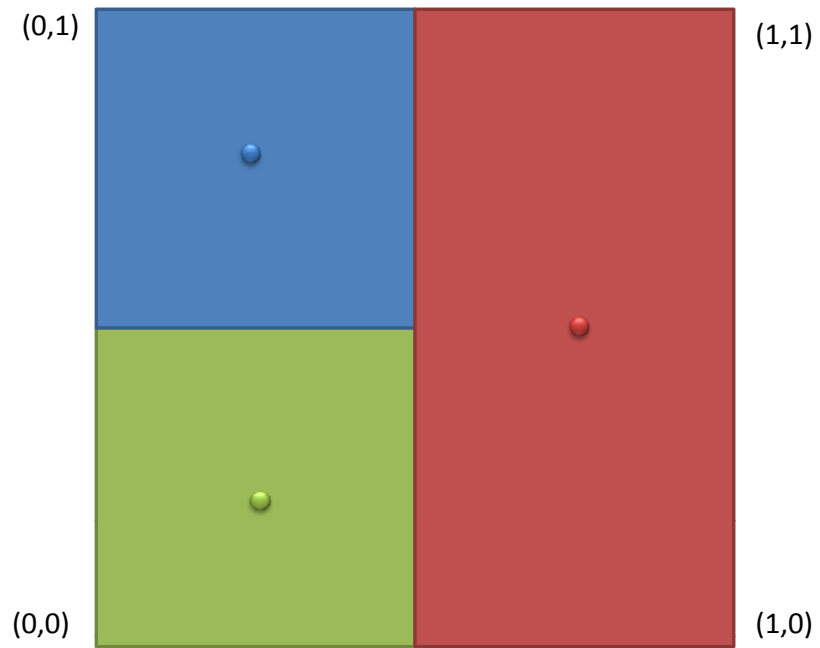
# CAN



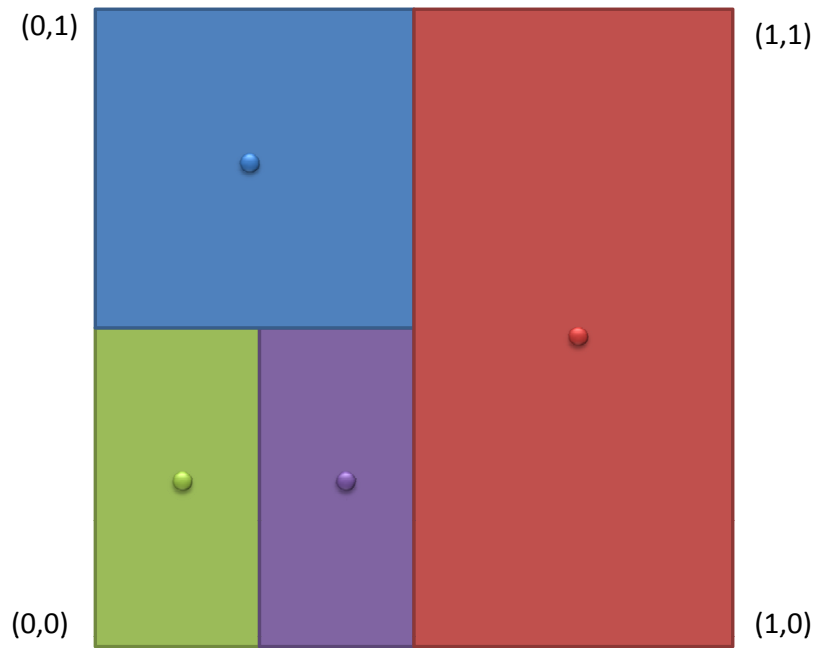
# CAN



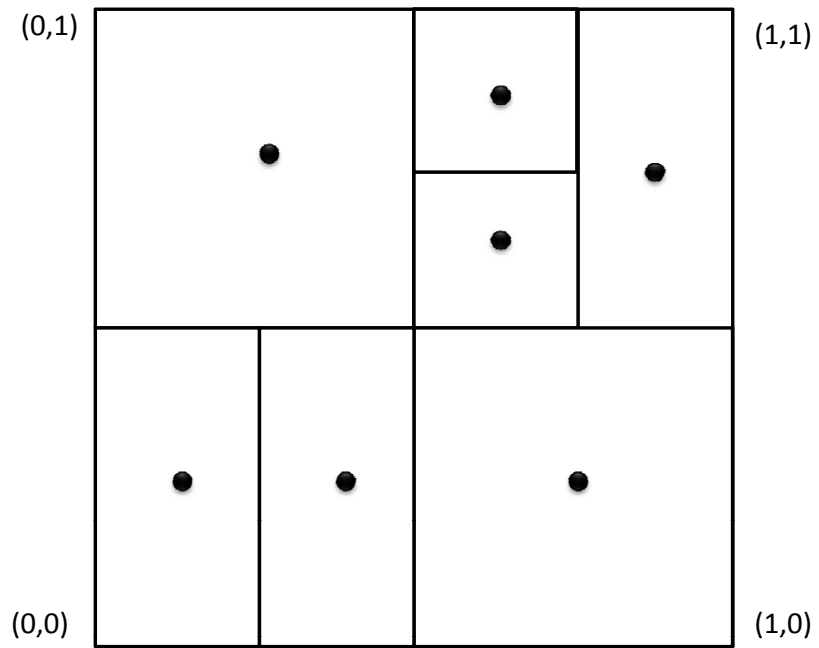
# CAN



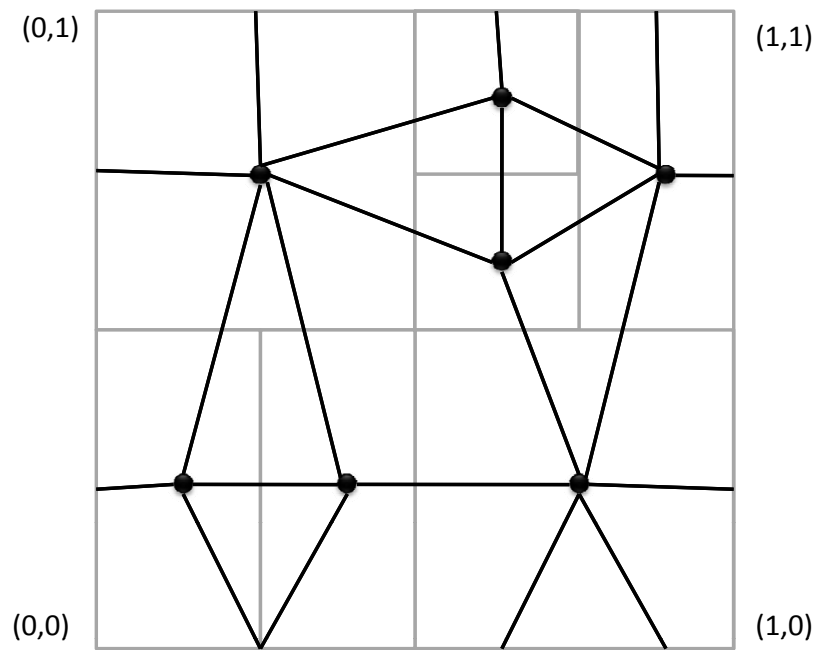
# CAN



# CAN



Achtung:  
*d*-torus





- ID-Raum ist 2-Dimensional kontinuierlich  $[0,1) \times [0,1)$
- Routing:
  - Wenn Ziel in meine Zone fällt, bin ich fertig
  - Sonst: Schicke Nachricht zu *einem* Nachbarknoten, der eine kleinere (euklidische) Distanz zum Ziel hat
- Knotenankunft:
  - Wähle zufällige ID aus  $[0,1) \times [0,1)$
  - Route Nachricht an diesen Punkt
  - Gebiet des Ziel-Peers wird geteilt
  - Baue Verbindungen zu allen Nachbarn des alten Peers auf

- Idealisiert: 2D-Gitter
- Nötige Routing-Schritte
  - $O(\sqrt{N})$
- Anzahl Nachbarn
  - 4 Nachbarn, also  $O(1)$
- Vergleiche mit Chord:
  - beides  $O(\log N)$
- → Routing langsamer, aber viel weniger Verbindungen und damit weniger Verwaltungsaufwand!
- Trotzdem robustes Netzwerk



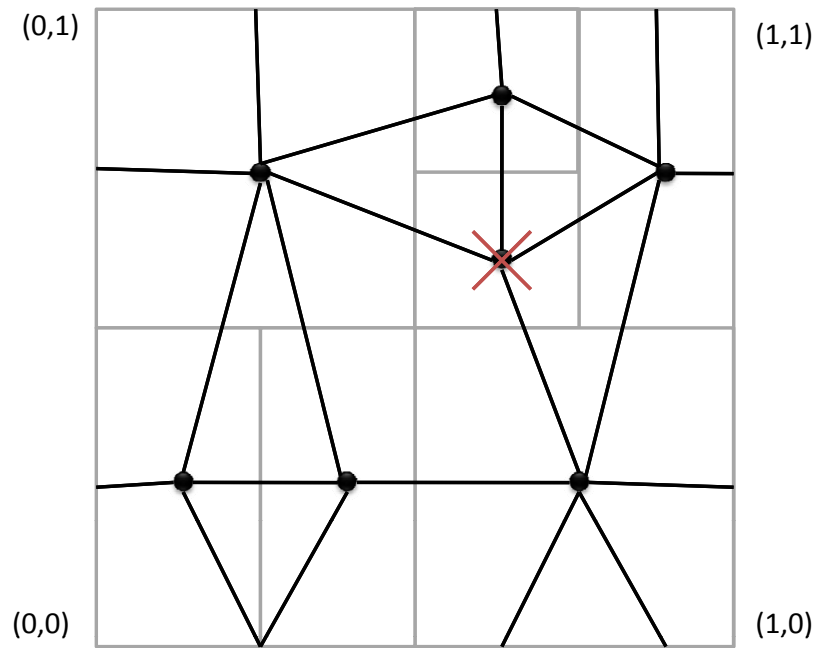
# Wie groß werden die Zonen?

---

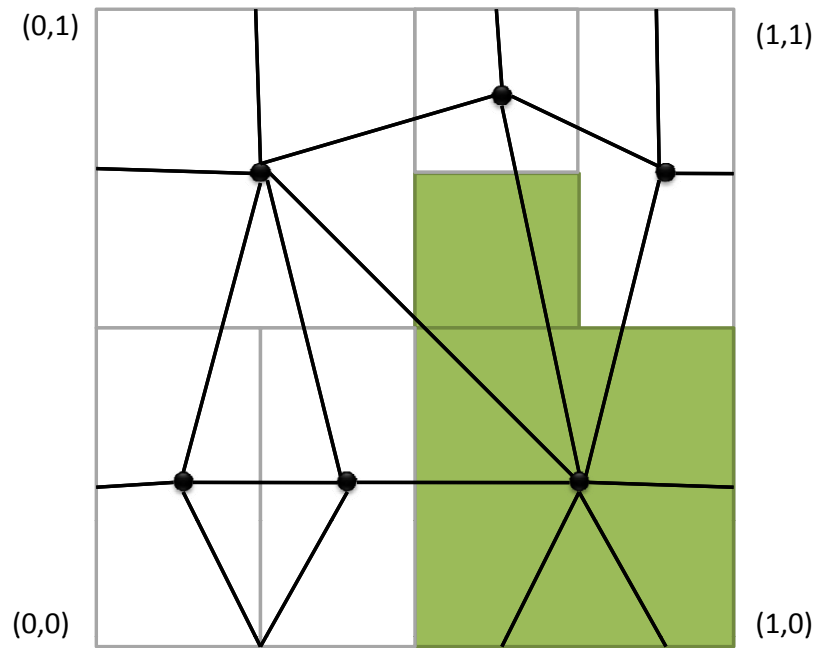
- → Übungsaufgabe

- Die Nachbarn entdecken, dass der Peer ausgefallen ist
- Ziel:
  - der Nachbar mit der kleinsten Zone soll den ausgefallenen Peer übernehmen
- Daher:
  - Jeder Knoten startet einen Timer proportional zur Größe der eigenen Zone
  - Wenn der Timer abgelaufen ist, sendet er eine Nachricht an alle anderen Nachbarn
  - Wer die Nachricht empfängt, stoppt seinen Timer
  - Danach übernimmt er die Zone des ausgefallenen Peers
- Kann zu **Fragmentierung** führen

# Fragmentierung

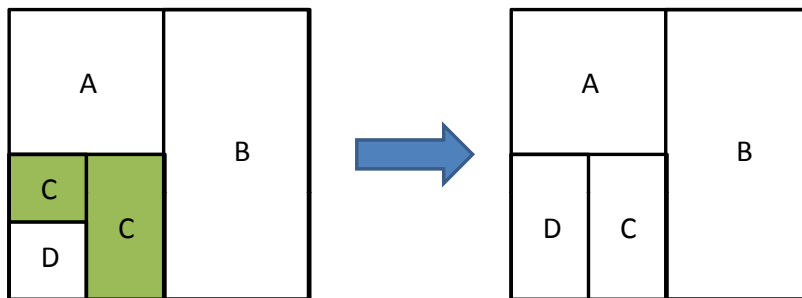


# Fragmentierung

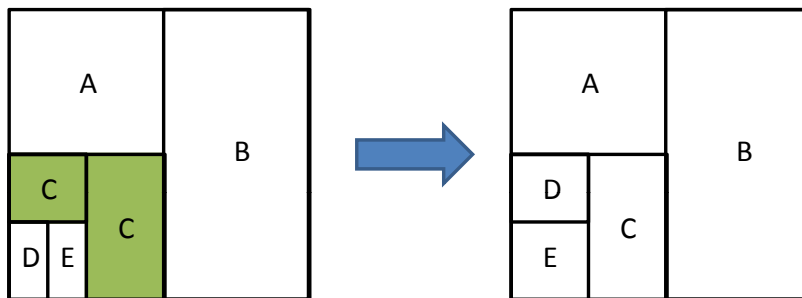


## Defragmentierung Teil 1

- Wenn ein Peer mehr als eine Zone hat:
  - Versuche, kleinste Zone loszuwerden
- Wenn die Nachbarzone ungeteilt ist (beide Knoten Blätter im Aufteilungsbaum):
  - Übergib die Zone dem Nachbar



- Wenn die Nachbarzone aufgeteilt ist:
  - Tiefensuche im Baum, bis zwei benachbarte Blätter gefunden wurden
  - Fasse diese Blätter zusammen
  - der freigewordene Peer übernimmt das Gebiet





- Analoge Konstruktion, aber d-Dimensionaler Schlüsselraum
- Bewirkt:
  - Kürzere Routing-Zeiten
  - Mehr Nachbarn
- Genauer:
  - $O(n^{1/d})$  Hops
  - 2d Nachbarn, also  $O(d)$  Nachbarn

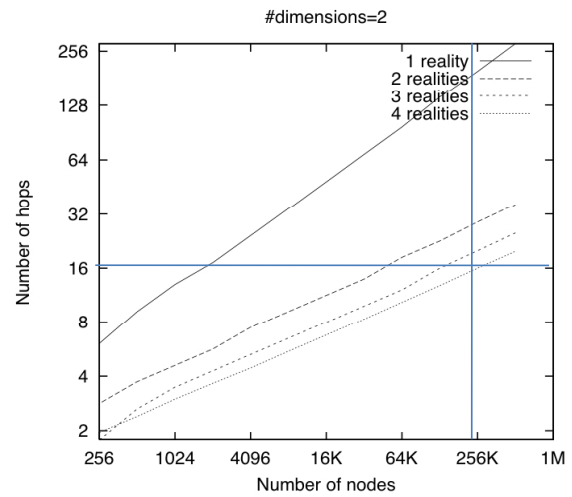
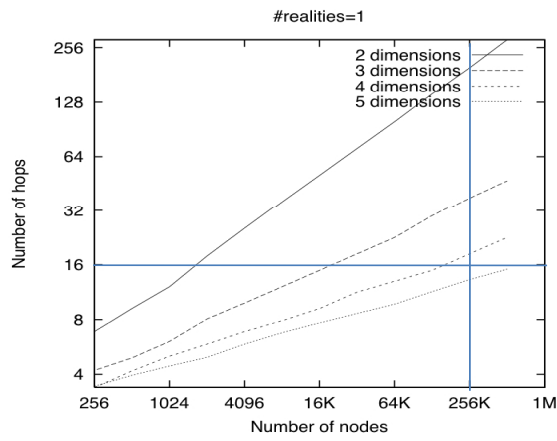
# Mehrere Realitäten

- Es werden gleichzeitig mehrere CAN Netzwerke aufgebaut
- Jedes heißt **Realität**
- Beim Routing wird zwischen den Realitäten gesprungen (besonders am Anfang)
- Bewirkt: Schnelleres Routing, Replikation, Robustheit

A	B	C	E
		D	
F	G	I	J
	H		

H	E	G	B
D	F	A	J
I			C

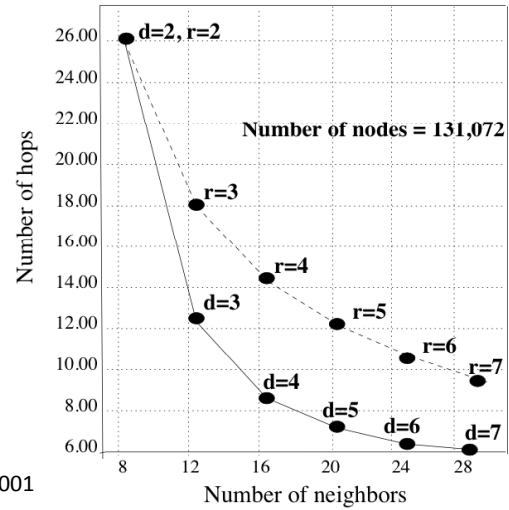
# Dimensionen vs. Realitäten



Quelle: Ratnasamy, Francis, Handley, Karp, Shenker:  
*A Scalable Content-Addressable Network*, SIGCOMM'01, 2001

# Dimensionen vs. Realitäten

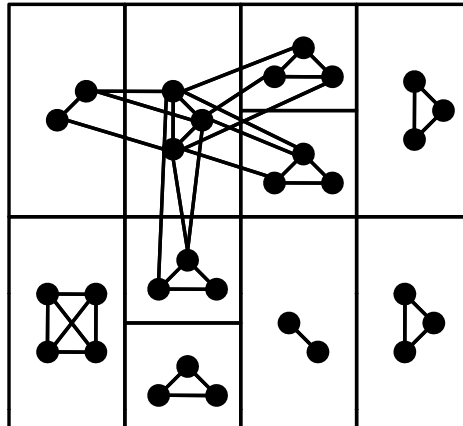
- ———— increasing dimensions, #realities=2
- - - - - - increasing realities, #dimensions=2



Quelle: Ratnasamy, Francis, Handley, Karp, Shenker:  
*A Scalable Content-Addressable Network*, SIGCOMM'01, 2001

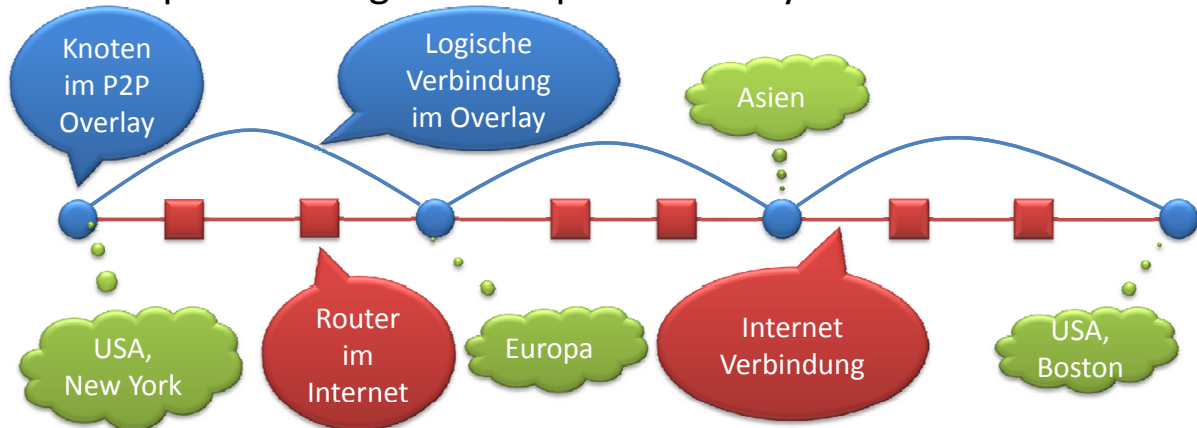
- Daten werden mehrfach abgespeichert
  - indem man  $k$  unterschiedliche Hash-Funktionen nimmt
- Dadurch erhöht sich die Robustheit
- Geringere Entfernungen
  - Lookup nur zu nächster Kopie

- In jeder Zone werden mehrere Peers platziert
- Alle Peers einer Zone kennen sich untereinander
- Jeder kennt mindestens **einen** aus der Nachbarschaft
- Bewirkt: Routing konstant schneller, Replikation



- Beim Join wird versucht, die Zonengröße besser auszugleichen
- Jeder Peer kennt die Zonen seiner Nachbarn
- Es wird die größte Zone geteilt

- Bisher haben wir Routing-Performance nur in Hop-Distanzen im Overlay-Netzwerk gemessen
- Wie sieht es tatsächlich aus?
- Beispiel: Routing mit 3 Hops im Overlay





- Die effektive Latenz im Routing hängt also von 2 Faktoren ab:
  - Anzahl Hops
  - Internet-Latenzen der einzelnen Hops
- Es kann also evtl. schneller sein, mehr Hops zu machen, die aber dafür lokaler sind
- Dazu muss das Routing oder die Verbindungsstruktur eine gewisse Flexibilität haben
- Bei Chord z.B. sind Struktur und Routen determiniert
- CAN ist hier flexibler

- In CAN 3 Ansätze:
  - Im Standard-CAN den nächsten Hop mit bestem Verhältnis ID-Distanz zu Latenz auswählen (statt maximaler ID-Distanz)  
**Proximity Route Selection (PRS)**
  - Beim Überladen von Zonen Verbindung mit dem "dichtesten" Knoten in der Nachbarzone eingehen
    - ➔ regelmäßig die anderen Kandidaten prüfen, evtl. wechseln**Proximity Neighbor Selection (PNS)**
  - ID nicht zufällig wählen, sondern anhand einer Lokalitäts-Metrik Position im Netzwerk wählen  
**Proximity Neighbor Selection (PNS)**



# Netzwerk an Internet-Topologie anpassen

---

- $m$  spezielle Peers dienen als "Landmarken"
- Latenzzeiten zu diesen Landmarken werden gemessen
- Liste der Latenzzeiten zu den Landmarken wird sortiert
- Diese Sortierung bestimmt Position im CAN
- Dadurch werden nahe Peers auch in CAN nah einsortiert
- Vorteile:
  - Gute Verringerung der Latenzzeiten
- Probleme
  - Wie wähle ich die Landmarken aus?
  - Lastungleichgewichte
  - Gefahr von Partitionierung

Parameter	bare bones CAN	knobs on full CAN			
Dim.	2	10			
Realitäten	1	1			
Peer / Zone	0	4			
Hashfkt.	1	1			
Latenz-Optimiertes Routing	Aus	An			
Uniform Partitioning	Aus	An			
Landmark Ordering	Aus	Aus			
			Metrik	bare bones CAN	knobs on full CAN
			Pfadlänge	198	5
			Grad	4,57	27,1
			Peers	0	2,95
			IP Latenz	115,9 ms	82,4 ms
			Pfad Latenz	23,008 sec.	135,29 ms

2<sup>18</sup> Knoten, Transit-Stub Topologie

- Vorteile
  - Einfaches Verfahren
  - Balanciert die Datenmenge
  - Kleiner Grad
  - Netzwerk ist stark zusammenhängend, dadurch robust
  - Kennt verschiedene Wege zum Ziel und kann dadurch Routen optimieren
- Nachteile
  - Lange Wege (polynomiell lang)
  - Stabilität durch geringe Nachbarzahl gefährdet