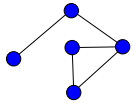
Gliederung

Peer-to-peer Systeme und Datenbanken(SS07)

- Kapitel 1: Einführung
- Kapitel 2: Beispiele
- Kapitel 3: Routing
- Kapitel 4: Schemabasierte p2p-Netzwerke
- Kapitel 5: Integrationsprobleme
  Teil 1:
  Teil 2:
  Teil 3:
- Kapitel 6: Anonymität, Authentifikation
- Kapitel 7: Dienstgüte

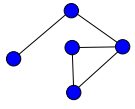
Version vom 11. Mai 2007



Kapitel 3

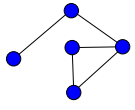
Routing in P2P-Systemen

- Allgemeines
- Hybride Systeme
- Fluten, Iterative Deeping, Directed BFS
- Lokale Indizes
- Verteilte Hashtabellen



Allgemeines

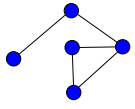
- Zentrale Frage: Wie kann ein Datum in einem reinen P2P-System gefunden werden?
- Wie findet ein Peer effizient Kontakt zu anderen Peers?
- Können datenbanktypische Fragen wie effiziente Suche und konsistentes Einbringen neuer Versionen hier gelöst werden?
- Aus Sicht des P2P-Prinzips ist die Anzahl der Nachrichten, die wegen einer Anfrage ausgelöst werden, interessant (Nutzersicht ist z.T. anders)



Hybride Systeme

typischer Vertreter: Napster, Echolink
Nachschlagen in einem Repository

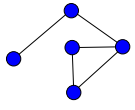
- Wegen des Servers - Suche: 1 Nachrichtenpaar (Anfrage, Antwort), erschöpfend
- Antwort: eine Liste der Peers, die das gesuchte Datum vorhalten.
- Jeder Peer bietet seine Daten an und die Daten, die er schon erhalten hat. (wegen Unsicherheit der Verfügbarkeit des Peers)
- Ist ein Datum einmal abgeholt worden, kann es vom Autor nicht mehr kontrolliert werden: Update-Problem
- An- und Abmelden unproblematisch.
- Erstkontakt unproblematisch
- Server - ein potentieller Flaschenhals? Schlechte Skalierbarkeit, Replizierung der Server ?



Reine P2P-Systeme - Fluten

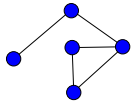
Verteter: Gnutella

- Breitensuche
- Idee: Anfrager sendet seine Anfrage als an alle ihm bekannten Peers (Nachbarknoten), diese wieder an alle ihnen bekannten P. außer dem Anfrager usw.
- Vorteil: Selbstverwaltendes Netz
- Nachteil:
  - Performanz i.a. schlecht, hohe Netzlast.
  - keine Reaktion auf Netzwerkanomalien möglich (gezielte Angriffe auf Knoten $\mapsto$ Ausfall $\mapsto$ Zerstörung der Netztopologie)
  - Knoten evt. mehrfach angefragt, Zyklenbildung möglich



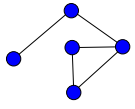
Fluten (2)

- Mechanismen zur Schleifenerkennung: Anfrage-ID (Zufallszahl, Zufallszahl + Host-ID, ...)
  Peer speichert ID eingehender Anfragen und bearbeitet sie nur, wenn er sie vorher nicht kannte.
  Gesamtes aktives Netz durchsucht.
- in der Praxis: Beschränkung der Zahl der bekannten Peers (z.B. 4)
- Verfahren zur Vermeidung der Nachteile des Fluten
  - Ohne Indextabelle (Non-Index-Flooding, Iterative Deeping, Directed BFS)
  - Mit Indextabelle (CRI, HCRI)
  - Verteilte Hashtabellen



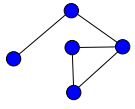
Fluten (3): Non.Indexed.Flooding

- Netzwerk bis zu vorgegebener Tiefe durchsucht.
  Time to live (TTL) - eine Zahl (z.B. 7), die jede Anfrage mit sich trägt und die beinhaltet, wie oft sie noch weitergeleitet werden soll. Bei jeder Weiterleitung um 1 decrementiert. Eine Anfrage mit der TTL=0 wird nicht weitergereicht (beschränkter Horizont)
- Systemlast reduziert, Knotenausfall wegen Überlast ?
  Netzpartitionierung ?
- Keine erschöpfende Suche, ineffizient
  Negatives Suchergebnis $\mapsto$ Grund: nicht vorhanden ODER nicht gefunden, nicht entscheidbar.
- Verbesserungen: Iterative Deeping, Directed BFS, Local Indices
  Yang,B. und Garcia-Molina, H.: *Effizient Search in Peer-to-Peer Networks.* Proc. VLDB 2001



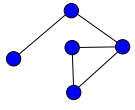
Iterative Deeping

- Ähnlich Non-Indexed-Flooding, aber mit einer streng aufsteigenden Folge $\{t_i, i = 1, ..., k\}$ von Suchtiefen
- Ablauf: Suche startet mit TTL=t_1, bei Nichterfolg oder geringem Erfolg Wiederholung mit TTL=t_2 usf. (*resend message* mit neuer Tiefe)
- Routing: Ein Knoten kann eine schon einmal bearbeitete Anfrage erkennen, bearbeitet sie nicht, sondern sendet sofort weiter. Damit liefern nur die Knoten der Tiefen $[t_i + 1, t_{i+1}]$ ihre Ergebnisse.
- Vorteile: Geringere Netz- u. Knotenlast bei erfolgreicher Suche. Nachteil: Bei erfolgloser Suche erhöhte Zeitdauer, da vor jedem Versuch Warten auf das Ergebnis des vorangehenden.



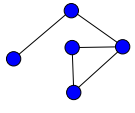
Iterative Deeping (2) - Variante

- Idee:
  Die Knoten, die die Anfrage mit einer TTT=0 erhalten, senden mit ihrer Anwort ihre Node-ID zurück. Wenn die Gesamtantwort beim Anfrager negativ ist, kann er diese Nodes zu neuen Nachbarn erklären und diese mit der ursprünglichen TTL direkt anfragen. Diese leiten dann die Anfrage als bekannt wie im Originalprotokoll sofort weiter.
- Vorteil:
  - Das Durchleiten der *resend messages* durch schon angefragte Netzknoten wird vermieden
  - Das Netz wird dichter.



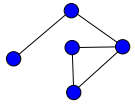
Directed Breadth First Search (Directed BFS)

- Breitensuche
- **Anfrage nur an ausgewählte Nachbarn** geschickt
  Statistiken über Nachbarn zur Auswahl geführt und zur Auswahl ausgewertet
- **Auswahl- / Qualitätskriterien** für Nachbarn:
  - Anzahl erfolgreich beantworteter Anfragen
  - Anzahl der Knoten, über die geroutet wird (hop count)
  - Anzahl der Nachrichten, die Nachbar verschickt ($\uparrow$ = gute Anbindung)
  - Länge der Warteschlange ($\uparrow$ = überlastet oder ausgefallen)



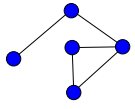
Local Indizes

- Idee: Jeder Knoten hält Index seiner Resourcen und der aller Nachbarn bis vorgegebener Tiefe
- Ablauf: Neue Anfrage - lokalen Index durchsuchen,
  bei positivem Resultat Suche beenden,
  bei negativem Resultat Anfrage weiterleiten an Nachbarn, diese routen sofort weiter an einen Ihrer Nachbarn, wenn sie schon im lokalen Index des Bearbeiters waren.
  Andernfalls Durchsuchen entsprechenden lokalen Index u. weiter wie oben
  Kombination mit TTL
- Vorteile: reduzierte Netzlast, geringere Rechenleistung nötig
  Nachteile / Aufwand:
  - Erhöhter Speicheraufwand - Betritt oder verläßt ein Peer das Netz, müssen alle betroffenen Indizes aktualisiert werden (*join, leave, update*)



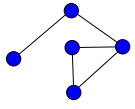
Routing Indizes

- Routing Indizes - keine direkten Referenzen auf Resourcen,
  Basis für Vorschläge für Weiterroutig
- Erhält ein Knoten eine Suchanfrage, die er weiterleiten muß, sucht unter Benutzung seiner Routing Indizes einen für die Anfrage geeignetzen Nachbarn und leitet die Anfrage an diesen weiter.
- Vorteile (gegenüber lokalen Indizes): veringerter Speicherplatz für Indizes, weniger Netzlast bei Veränderung der Netztopologie
- Methoden zur Bestimmung der geeigneten Nachbarn:
  Crespo, A. und Garcia-Molina, H.: Routing Indices For Peer-to-Peer Systems. Proc. Int. Conf. on Distributed Computing Systems (ICDCS). Juli 2002.
  Mehrere Vorschläge, u.a.
  CRI (Compound Routing Indices), HCRI (Hop-Count Routing Indices)



Compound Routing Indices CRI

- Routing Indices, in denen Ressourcen in Kategorien zusammengefaßt, (Extern: Erstellen der Kategorien, Verschlagwortung)
- Knoten: Index mit lokalen Resourcen, Routingtabelle (Nachbarknoten, Anzahl der dortigen Resourcen pro Kategorie).
- Kann eine Suchanfrage nicht mit lok. Index pos. beantwortet werden, routing an den Nachbarn mit der besten Güte bezügl. dieser Anfrage, dieser analog.
- Gütebewertung:
  - Anfrage $\mapsto$ **betroffene Kategorien** $S = \{s_l, l = 1, ..., k\}$ (mehrere möglich)
  - Bewertung des Knotens i: $G_i = (\sharp(Res_i))^{-(k-1)} \times \prod_l (\sharp_i(s_l))$



CRI (2)

- Vorteile:

  - reduzierter Platzbedarf für Routing-Tabellen

  - weniger Anfragen in Richtungen, wo keine / wenige Resultate erwartet

  - Performanz (Simulation) gegenüber Flooding: 10 ... 100, 1,5 ...2 gegenüber Zufallsauswahl

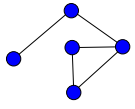
  - Nachrichtenzahl $\sim$ Resultatmenge

- Nachteile:

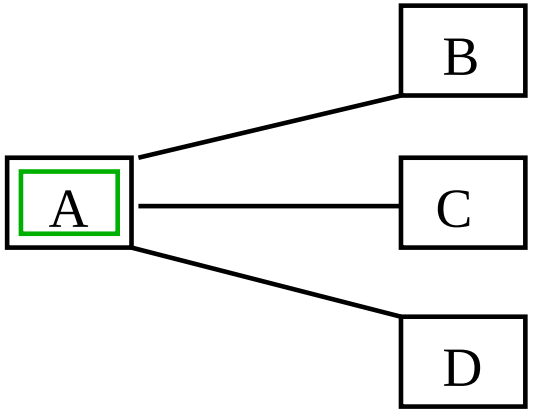
  - keine Gewichtung der Entfernung zu Resourcen in den Tabellen - Netzlast durch Erstellen und Aktualisierung der CRI, Änderungen einer Anzahl wird überall wirksam

- Neue Verbindung zwischen Knoten K und L: K aggregiert seine CRI und sendet Ergebnisvektor an L, dieser in seiner Tabelle neue Zeile mit Node-ID(K) und aggregierten Werten von K; L analog an K.

- Auch andere Bewertungen außer Abzählen möglich



CRI - Anfrage



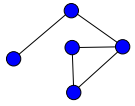
Routing Index bei A:

	#	r	s	t
A	70	0	30	10
B	100	10	20	0
C	100	50	30	40
D	200	100	30	50

Anfrage bei A, betrifft die Themen r und t:

Güteberechnung

	#	r	s	t
G_B	= 100*(	10/100) *		(0/100) = 0
G_C	= 100*(	50/100) *		(40/100) = 20
G_D	= 200*(100/200) *			(50/100) = 25



CRI - Tabellenaufbau

Knoten unverbunden

A

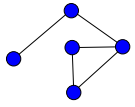
Node	#	r	s	t
A	1000	30	100	0

B

Node	#	r	s	t
B	500	20	10	100

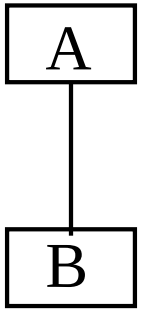
C

Node	#	r	s	t
C	100	0	20	0



CRI - Tabellenaufbau (2)

Knoten A mit B verbunden



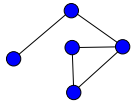
	Node	#	r	s	t
✓	A	1000	30	100	0
	B	500	20	10	100

	Node	#	r	s	t
✓	B	500	20	10	100
	A	1000	30	100	0

C

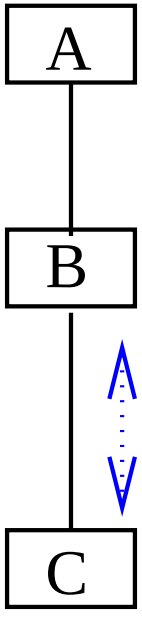
	Node	#	r	s	t
✓	C	100	0	20	0

✓ Lok. Index



CRI - Tabellenaufbau (3)

Knoten C kontaktiert B, 1.Schritt

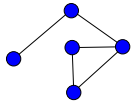


	Node	#	r	s	t
✓	A	1000	30	100	0
	B	500	20	10	100

	Node	#	r	s	t
✓	B	500	20	10	100
	A	1000	30	100	0
	C	100	0	20	0

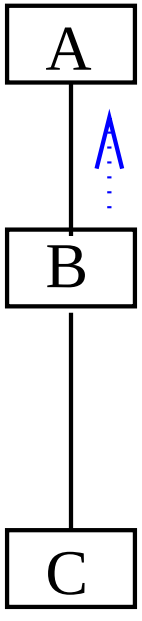
	Node	#	r	s	t
✓	C	100	0	20	0
	B	1500	50	110	100

✓ Lok. Index



CRI - Tabellenaufbau (4)

Knoten C kontaktiert B, 2.Schritt

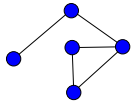


	Node	#	r	s	t
✓	A	1000	30	100	0
	B	600	20	30	100

	Node	#	r	s	t
✓	B	500	20	10	100
	A	1000	30	100	0
	C	100	0	20	0

	Node	#	r	s	t
✓	C	100	0	20	0
	B	1500	50	110	100

✓ Lok. Index

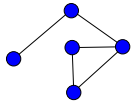


Hope-Count Routing Indizes HCRI

- Verbesserung des CRI durch Einbeziehung der Entfernung zu den Resourcen und Einführung eines Horizonts $h \in \mathbf{N}$
- Entfernung(K,L) = minimale Anzahl der Kanten im Netzgraphen bei Pfaden von K nach L.
  Der Knoten K speichert einen CRI für jede Entfernung von 1 bis zum Horizontentfernung h
- Einbeziehung der Entfernung in die Bewertung eines Knotens:
  K bewertet L bei Anfrage A:
  $Q_K(L, A) =$

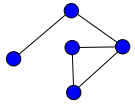
$$\frac{\text{Anzahl der Dokumente, die L bei A liefert}}{\text{Anzahl der Nachrichten, die noetig sind, diese Dokumente zu erhalten}}$$

- Fanout F (Jeder Knoten ist exakt mit F+1 Knoten verbunden):
  Bewertung berechenbar, als Schätzung für andere Fälle nutzbar.
- Nachteil: Horizont = Grenze



Verteilte Hashtabellen

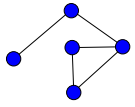
- Idee: von Resourcen (Inhalt, Namen, ...) werden Hashwerte erzeugt, über die die Lokalisierung stattfindet.
- Jedem Knoten wird ein Bereich aus dem Wertevorrat der Hashfunktion zugeordnet, Knoten „verwaltet“die zugeh. Ressourcen (bzw. Verweise).
- Lokalisation (Suche, Einbringen) - Suche nach dem den entsprechenden Hashwert verwaltenden Knoten.
- Vorteile:
  Lokalisation lokal, dezentral berechenbar, Auflösung von Clustern.
  Skalierbarkeit, Lastbalanzierung, Verfügbarkeit, Behebung von Namenskonflikten
- Nachteil (aller Hashverfahren): keine Unterstützung von Gebietsanfragen.



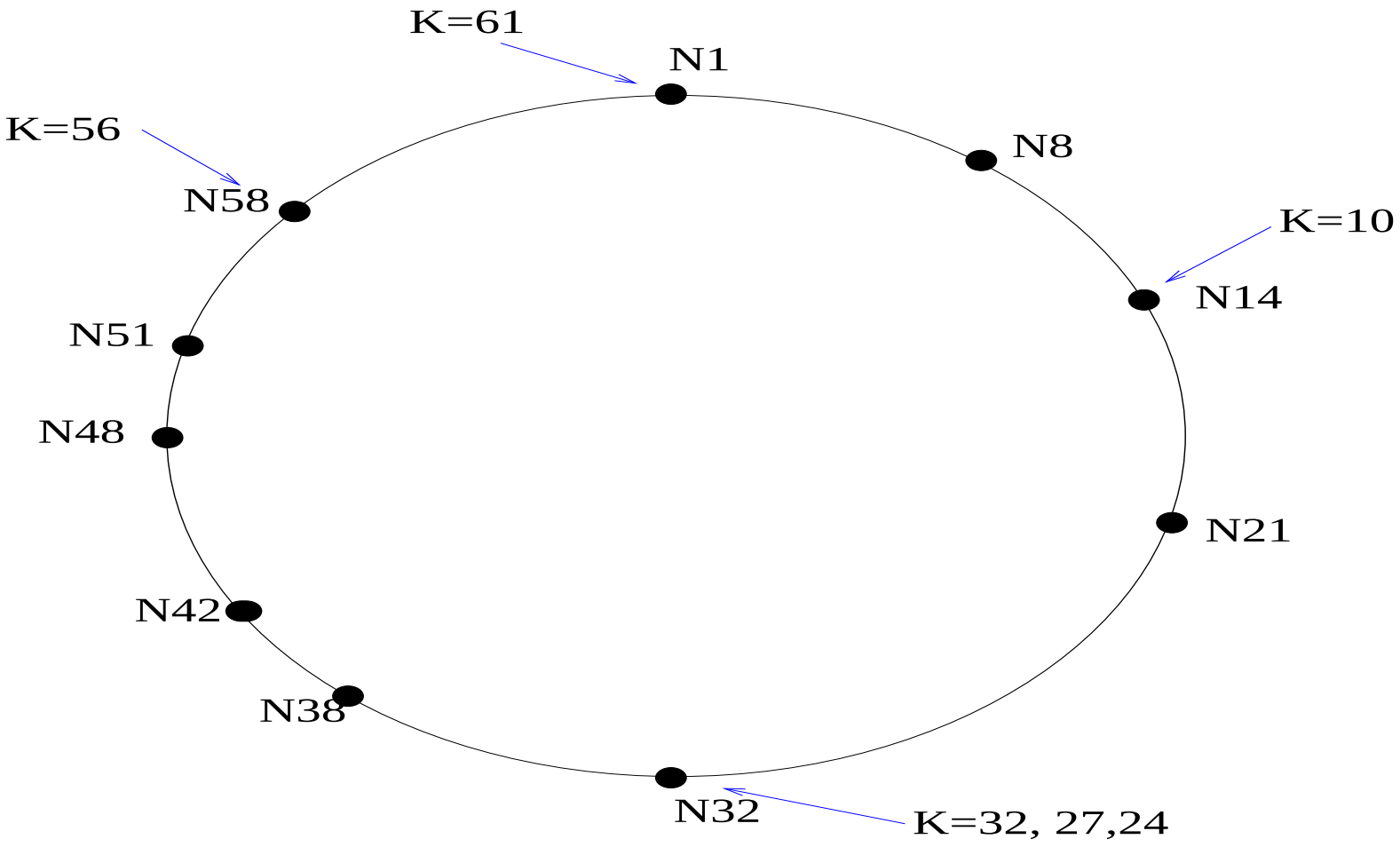
Verteilte Hashtabellen - CHORD

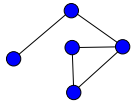
Literaratur: Stoica, I. u.a.: *Chord - A Scalable Peer-to-Peer Lookup Service for Internet Applications.* Proc. of the 2001 conf. on applications, technologies, architecture, and protocols for computer communications.

- Idee: Die Netzknoten bilden logischen Ring (, die Nummer i eines Knotens ist der Hashwert (m Bit) seiner IP-Addr. in einem hinr. gr. Restklassenring $R(2^m)$; Kollisionen ?)
  Mit gleicher Hashfunktion aus jedem Datum (Schlüssel k) Hashwert $h(k)$ abgeleitet (konsitenter Hash). Datum /Verweis auf dem Knoten j mit $j = \min_{(i \geq k)} i$ gespeichert (successor(h).
  Jeder Knoten kennt seinen Nachfolger.
  Betritt / Verläßt ein Knoten den Ring mit N Knoten $\mapsto 1/N$ der Routing-Informationen umspeichern



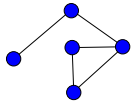
CHORD (2)





CHORD (3) Verteilung, Suche

- Satz (über die Schlüsselverteilung): Für jeden Satz von Knoten N und jeden Satz von Schlüsseln K gilt *mit hoher Wahrscheinlichkeit*:
  (1) Jeder Knoten ist zuständig für höchstens $(1+\epsilon)|K|/|N|$ Schlüssel.
  (2) Wenn bei N Knoten einer hizukommt oder das Netzwerk verläßt, sind $O(|K|/|N|)$ betroffen, die alle beim eintretenden oder verlassenden Knoten liegen.
- Wesentlich ist die gleichverteilende Wirkung der Hash-Funktion, auch Cluster verteilend. (z.B. SHA-1). Wenn die Hashfunktion gewählt ist, Angriffe gegen die Gleichverteilung möglich.
- Einfache /langsame Suche: Kann ein Knoten nicht beantworten, prüft er (rekursiv), ob der Nachfolger den Schlüssel kennt.
- Beispiel (letzte Abbildung): Knoten 8 sucht nach K=56

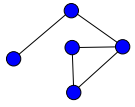


CHORD (4) Suche (2)

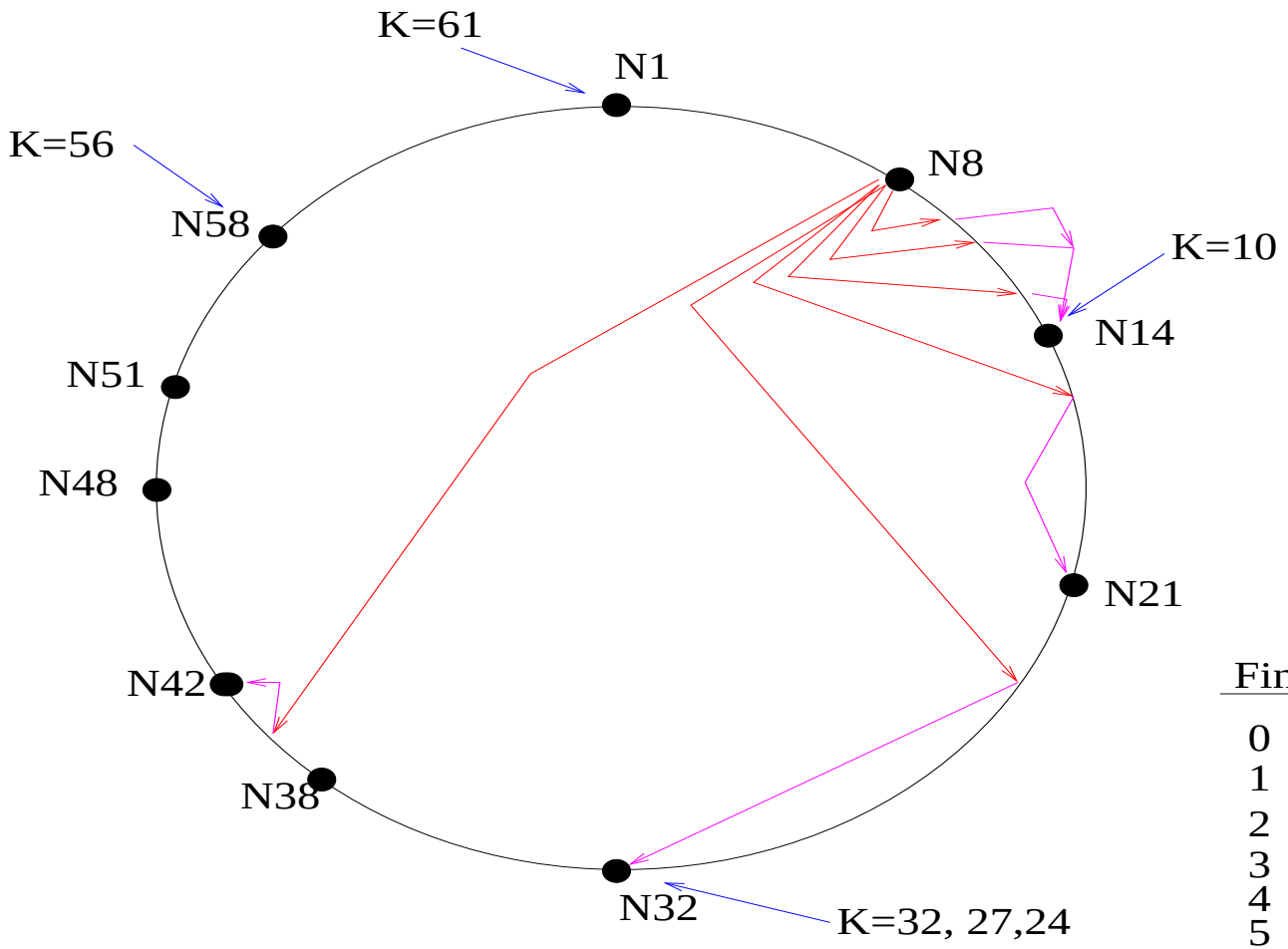
- Pseudocode: sucht in Knoten n, ob der Nachfolger zuständig ist, bei Nichterfolg startet er die Suche beim Nachfloger rekursiv:

```
n.find-successor(id)
  if(id in (n.successor) return (successor)
  else return (successor.find-successor(id)
```

- Komplexität: $O(|N|)$,
- Verbesserung: Fingertabelle: (Sprungsuche) Knoten n hat Tabelle, welcher Knoten (ID, IP-Adresse) an Position $n.finger\,[i] = n + 2^i, i = 0, 1, 2, ..., m - 1$, m Hashwertlänge im Ring. Fehlt der Knoten mit dieser ID, dann nächstfolgender real existierender Knoten.
  Komplexität $O(\log |N|)$, $|N|$ Anzahl der Knoten im Netz

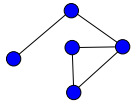


CHORD (5) Fingertabellen

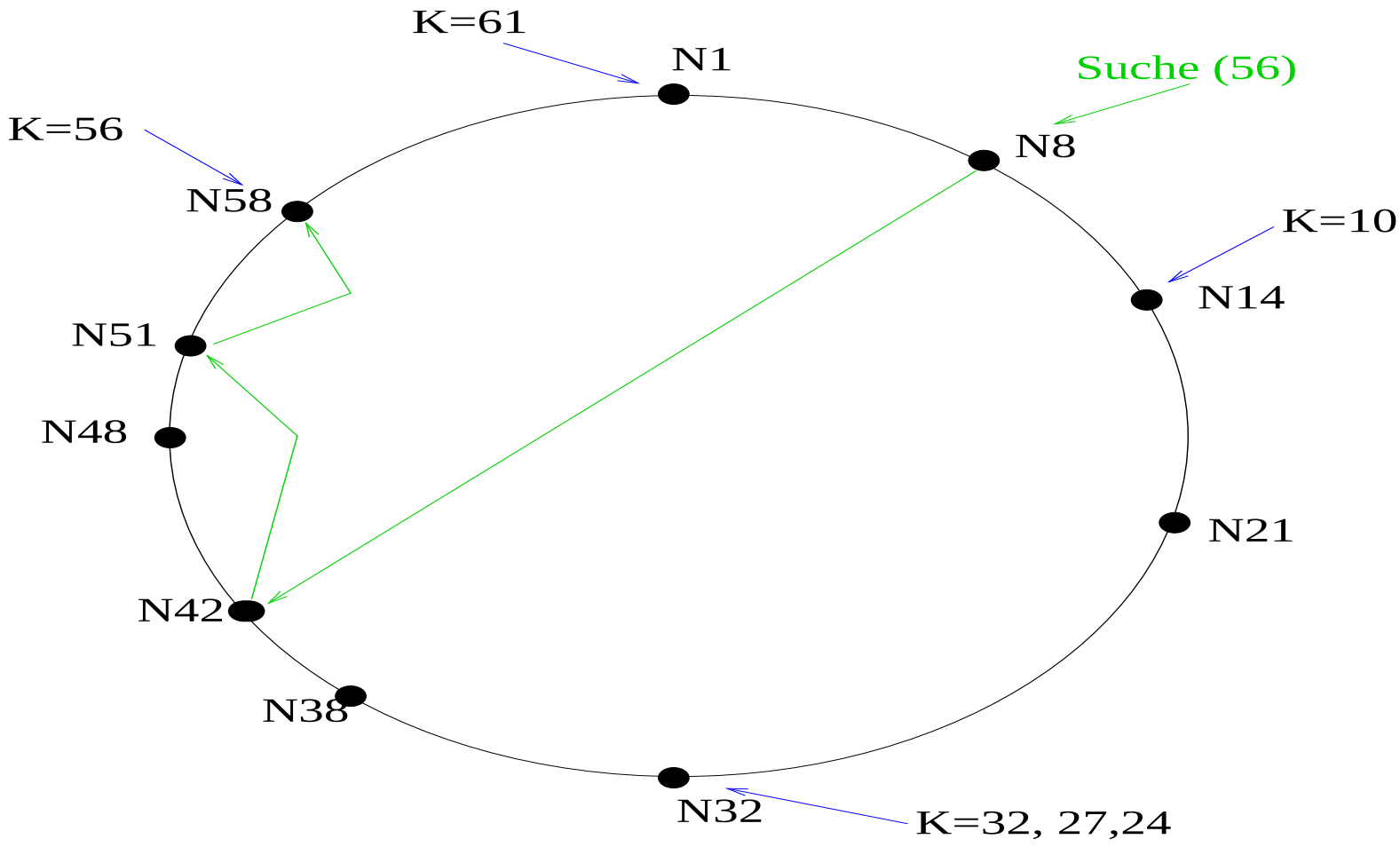


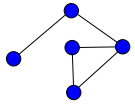
Fingertabelle N8

0	1	N14
1	2	N14
2	4	N14
3	8	N21
4	16	N32
5	32	N42



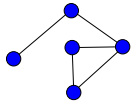
CHORD (6) Fingersuche



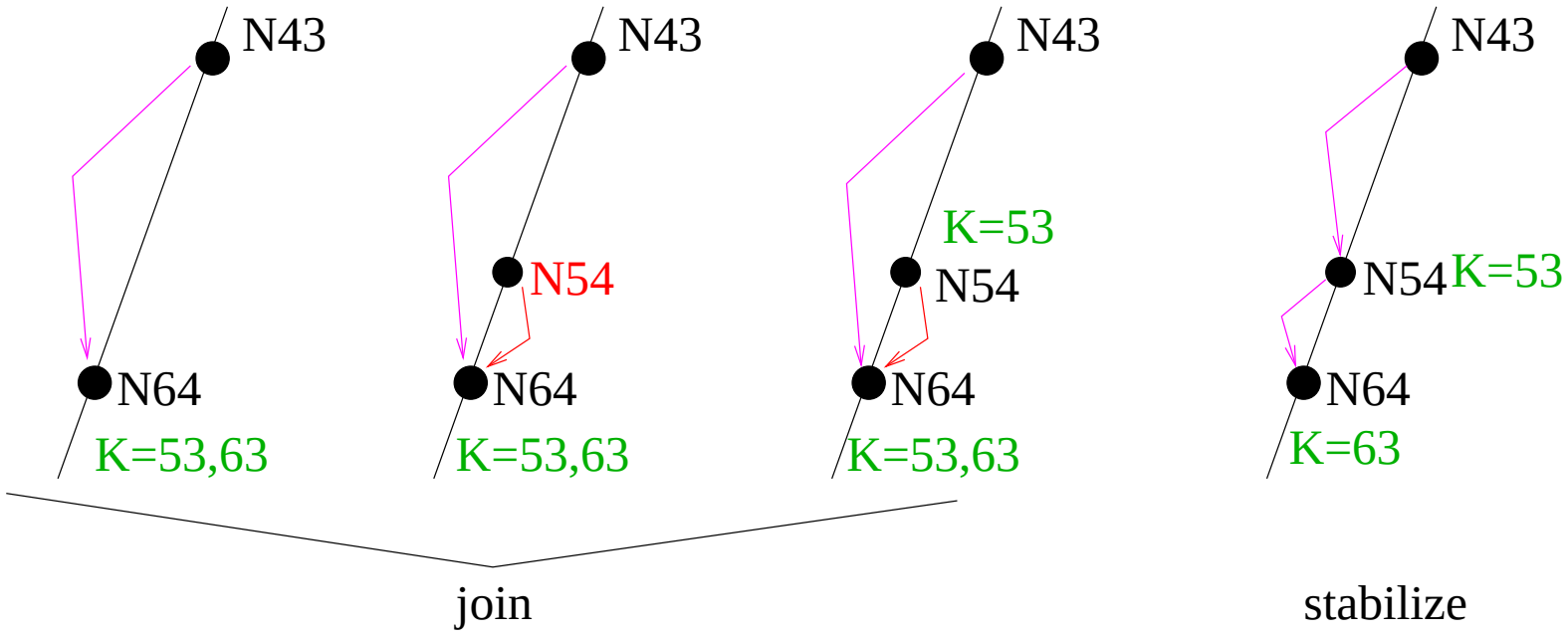


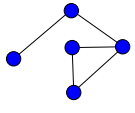
CHORD (7) - Anmelden, Stabilität

- ein Knoten n_0 bekannt: ($n.join(n_0)$) ordnet n an der richtigen Stelle ein und liefert den Nachfolger
  kein Knoten bekann: $n.create()$ eröffnet neuen Ring
- Wegen des dynam Verhaltens kennt jeder Knoten auch seinen Vorgänger.
  Jeder Knoten in regelmäßigen Abständen- $stabilize()$ - fragt Nachfolger nach dessen Vorgänger, ändert dann evt. sein $n.finger\,[1]$
- Es gibt allg. Aussagen über Qualität, diese können verbessert werden, denn Knoten eine geordnete Abmeldung realisieren.
- NodeID kennen das physische Netz nicht, der log. nächste Knoten evt. physisch weit entfernt (Latenzzeit). Abhilfe: z.B. mehrer Nachfolger und Laufzeitstatistiken oder Abstandsfunktion im Netz $\mapsto$ bester Nachfolger gewählt.



CHORD (8) - Anmelden, Stabilität (2)

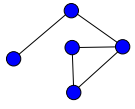




Verteilte Hashtabellen - CAN

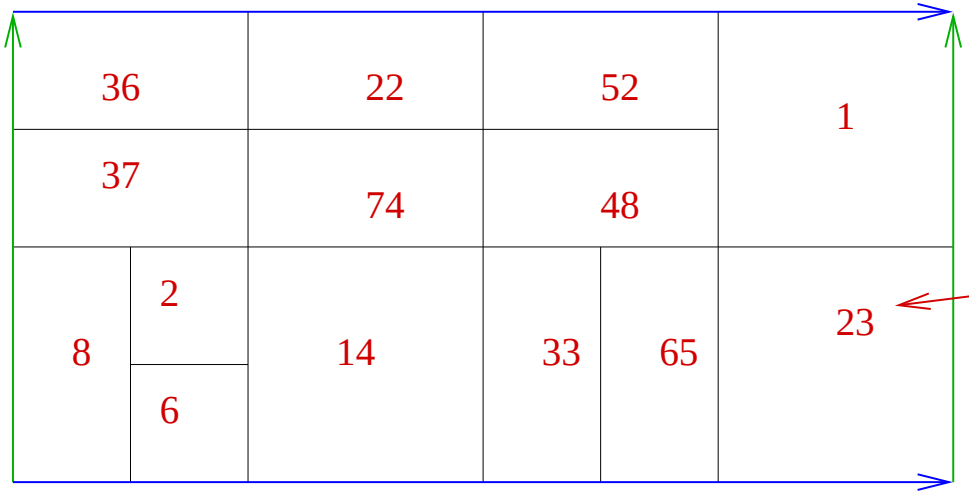
Literatur: Sylvia Ratnasamy: *A Scalable Content-Addressable Network.* Proc. ACM SIGCOMM. 2001, pp. 161 - 172.
Ideen:

- Daten (Schlüssel) und Adressen der Knoten gehasht, determinist. Hashwerte als Datum auf einem d-dimendionalen Torus interpretiert.
  Kein Bezug zu phys. Daten oder Adressen.
- Jeder Knoten verwaltet die Daten in einer Region (der Hash-Tabelle), in der er liegt.
- Neuer Knoten teilt die Region, in die er kommt (in zwei gleiche Teile)
- Routing einer Anfrage: Weiterleitung einer Anfrage an den /einen besser geeigneten Nachbarn. (Nachbar auf dem Torus). Es gibt i.A. mehrere Wege zwischen zwei Knoten - Ausfallsicherheit $\uparrow$
- Knoten kennen das phys. Netzwerk nicht, Latenzprobleme s.o.



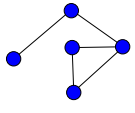
CAN (2): Topologie des Schlüsselraums

- Definition: Zwei Knoten sind **benachbart**, wenn es eine Koordinatenachse gibt, hinsichtlich der sich die von ihnen verwalteten Bereiche überlappen
- 2-dim. Beispiel: Torus

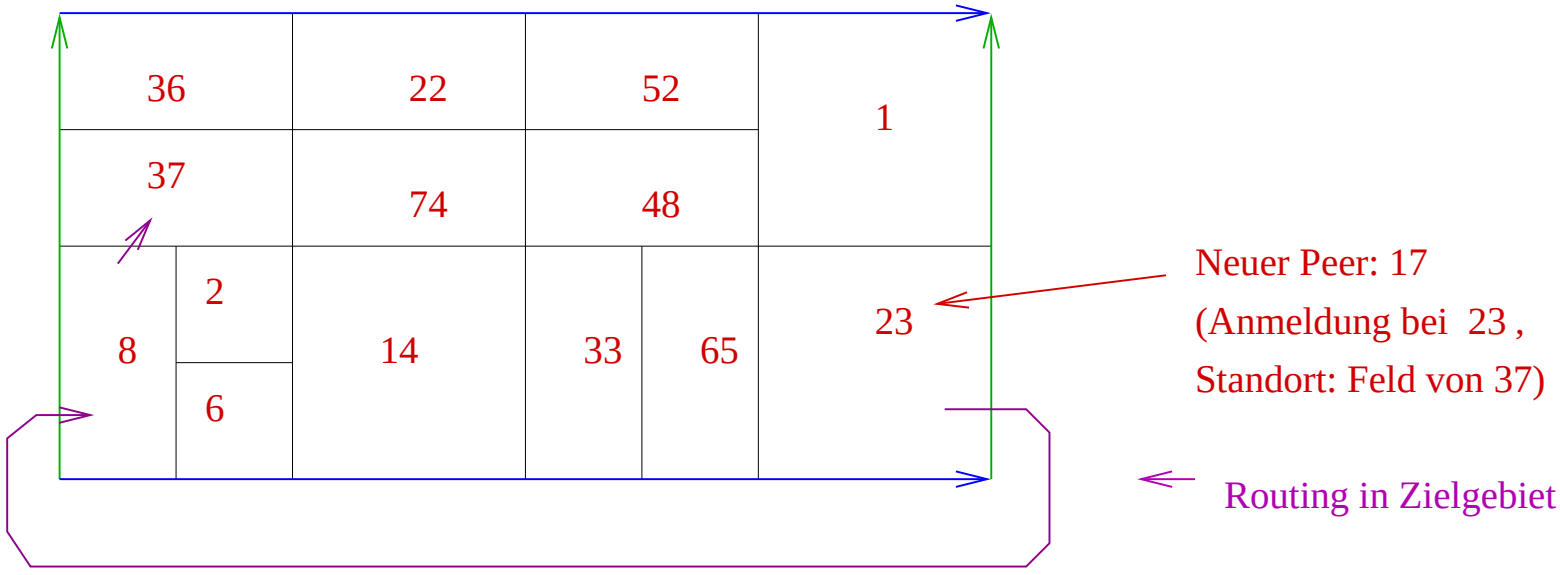


Nachbar(22):74, 36, 52,14

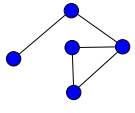
Neuer Peer: 17
(Anmeldung bei 23 ,
Standort: Feld von 37)



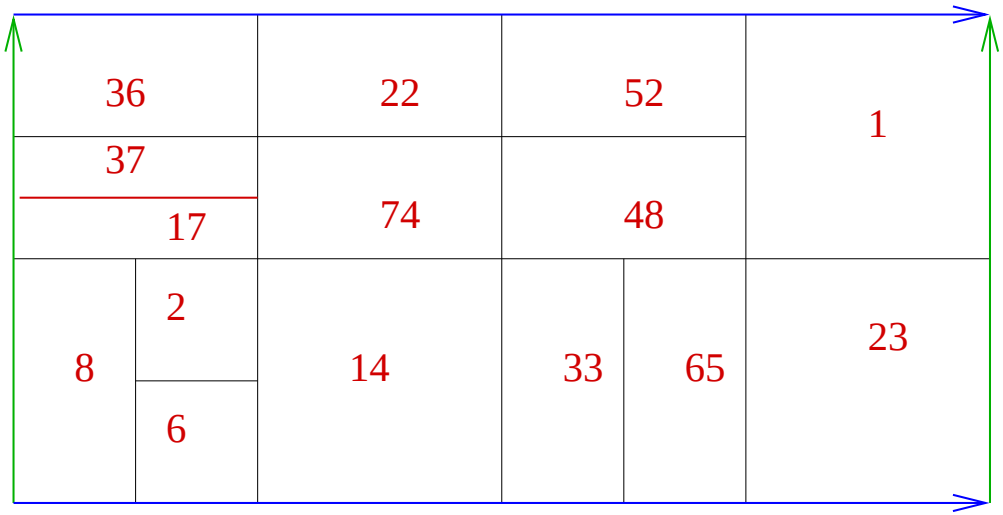
CAN (3): Anmelden



- Routing: Jeder Knoten führt Nachbarschaftstabelle entlang aller d Koordinatenrichtungen: (Hashwert, IP-Adresse)
  Ist ein Knoten nicht zuständig, gibt er Anfrage weiter an den Nachbarknoten, der dem Anfrageziel am ähnlichsten. Ahnlichkeitsmaß ?



CAN (4): Anmelden (2)



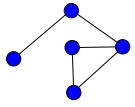
Neuer Peer: 17

(Standort: Feld von 37)

Nachbarn (37):
alt: 8, 2, 74, 36
neu: 17, 74, 36

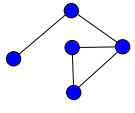
Nachbarn (17):
8, 2, 74, 37

- Neuer und Alter Knoten teilen den Bereich auf und aktualisieren ihre Nachbarschaftstabellen, informieren ihre Nachbarn über Veränderung (z.B. 17 informiert 2, 8, 74, 1) $\mapsto$ nur $O(d)$ Knoten von Änderung betroffen.
- Komplexität: Dimension d, je n Teile(gleichmäßige Partitionierung: $\mapsto$ Mittl. Länge des Routing-Pfads: $(d/4)(n^{1/d})$.



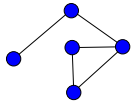
CAN (5) Abmelden, Fehler

- Verlassen: ein Nachbarknoten muss Daten übernehmen,
  Regel: wenn ein passender Knoten existiert (so dass ein reguläres Gebiet entsteht), dann dieser, andernfalls temporär der mit dem kleinsten Volumen.
- Fehler (Knotenausfall): *immediate-takeover*, ein Nachbarknoten übernimmt sofort das Gebiet, aber dabei Datenverlust. Knoten senden periodisch *update-Nachrichten* (Gebietsgrenzen, Nachbarn, deren Gebietsgrenzen). Ausbleiben dieser Nachrichten $\mapsto$ Fehler wird unterstellt. Jeder Nachbar des fehlerhaften Knoten beginnt selbständig takeover mit einer Verzögerung proportional zu Gebietsgöße und sendet Übernahmenachricht an alle Nachbarn. Die Empfänger akzeptieren dies, wenn ihre Verzögerung noch läuft $\mapsto$ Knoten mit kleinsten Gebiet gewinnt.



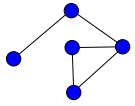
CAN (6) Mehrere Fehler

- Ausfall mehrerer zusammenhängender Nachbarknoten - von deren Nachbarn dann noch weniger als die Hälfte erreichbar sind: Gebietsübernahme kann inkonsist sein. Suche nach existierenden Knoten in expandierenden Ringen, mit diesen dann Neuorganisation der Gebiete wie beschrieben.
- Wiederholte Fehler und temopäre Gebietsübernahme bei einem Gebiet- im Hintergrund laufen Reorganisationen an dieser Stelle, die dafür sorgen, dass die Gebiete zu Quadern im R^d reorganisiert werden.



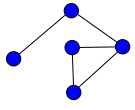
CAN (7) Ausfallsicherheit, Erweiterungen

- Dimension: d kann beliebig sein: Länge des Suchpfades s.o., Anzahl der Nachbarn $2d$, Sicherheit bei Ausfall steigt mit d.
- Mehrere unbhängige Koordinatenräume, mehrere Zeiger auf ein Datum, Ausfallsichertheit $\uparrow$.
- Nachbarschaft auf Anwendungsniveau unabhängig von Internettopologie, Verbesserung: Bewertung des Antwortzeitverhalens der Nachbarn.
- Überladen der Zonen - mehrere Peers verwalten eine Zone (kooperieren alle, kennen aber nur einen Knoten aus der Nachbarzone, werden bei Teilung einer Zone mit aufgeteilt).
  Vorteile: reduzierte Pfadlänge, mehrere Möglichkeiten der Weiterleitung: Lastverteilung, Ausfallsicherheit.
  Nachteil: Hohere Systemkomplexität
- Caching und Replikation zur Lastverteilung:
  Caching: Knoten hält zusätzlich Schüssel, die er jüngst zugegriffen hat.
  Replikation: häufig abgerufene Daten zusätzlich auf Nachbarn repliziert.



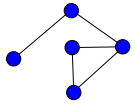
Routing bei Freenet

- Unterscheidung: Node vs. Client - Ein Peer betreibt i.a. einen Node, er kann aber auch den Node eines anderen Peer nutzen (Free Client Protokoll, unverschlüsselt $\mapsto$ Jeder Peer sollte selbst Node betreiben)
- Datum über eine ID identifizeiert.
- Grundsätzlich speichert nicht der Anbieter, jeder Node spezialisiert sich auf einen ID-Bereich, den er bevorzugt speichert.
- Jeder Node kennt die Spezialisierung einiger anderer Nodes
- Einbringen: Autor berechnet aus Inhalt oder aus Kennwort die Daten-ID. Ähnlich Routing algorithmus:Node des Autors sendet die Daten an den Node, dessen Spezialisierung die ID enthält.



Routing bei Freenet (2)

- Anfrageweiterleitung:
  Knoten sucht aus seiner Tabelle bekannter Nachbarn den/die aus, in dessen/deren Interessenbereich die gesuchte ID der Anfrage liegt. Gibt es keinen Knoten mit passendem Interessenbereich, wird der Knoten mit der größten Ähnlichkeit gewählt. (Ähnlichkeitsmaß ?)
- Datenlieferung: Weitergabe von Knoten zu Knoten. Alle Zwischennodes speichern die Daten auch, dort keine Information, ob der Vorgänger Author (Einbringen) /Quelle(Suche) war oder nur Relais. Diese Information nur beim ersten Node.



Zusammenfassung

- - Strategie ohne Kenntnis der Umgebung
  - mit Kenntnis der Inhalte der Nachbarn
  - mit mit globalem Wissen
- Netztopologie beeinflusst routing
- Mit Aufwand steigt Qualität (Ergebnisse, Netzstabilität)
  Mit Blick auf DB-Anwendungen: CHORD- und CAN-artige Mechanismen interessant.