Lookup Performanz von Verteilten Hashtabellen

Vortrag von Martin Christian <martin_at_christianix_dot_de>

1. **Verteilte Hashtabellen**
2. **Routing-Strategien**
3. **Lookup-Strategien**
4. **Replikationsstrategien**
5. **Zusammenfassung**

Peer-to-Peer Netze

Hybride P2P-Netze:

- zentralistisch: zentraler Server verwaltet Peers und Ressourcen,
  z. B.: Napster
- hierarchisch: hierarchische Struktur der Server,
  z. B.: DNS

Reine P2P-Netze:

- unstrukturiert: zwischen Nachbarn gibt es keine Unterschiede,
  z. B.: Gnutella
- strukturiert: Nachbarn stehen in einem Verhältnis zueinander,
  z. B.: DHTs

Verteilte Hashtabellen

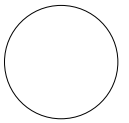
engl. Distributed Hashtables (DHT)

Vorteile:

- Definieren einfache Schnittstelle: lookup(key)
- Keine Hierarchien => reines P2P Netz
- Skalierbar => Nachrichten werden nur ausgewählten Nachbarn gesendet

Beispiele für Chord und Kademlia:

- 4-Bit Schlüssel
- Knoten: 0x1 (1), 0x3 (3), 0xA (10), 0xD (13), 0xE (14)
- Daten: 0x3 (3), 0x8 (8), 0xC (12)



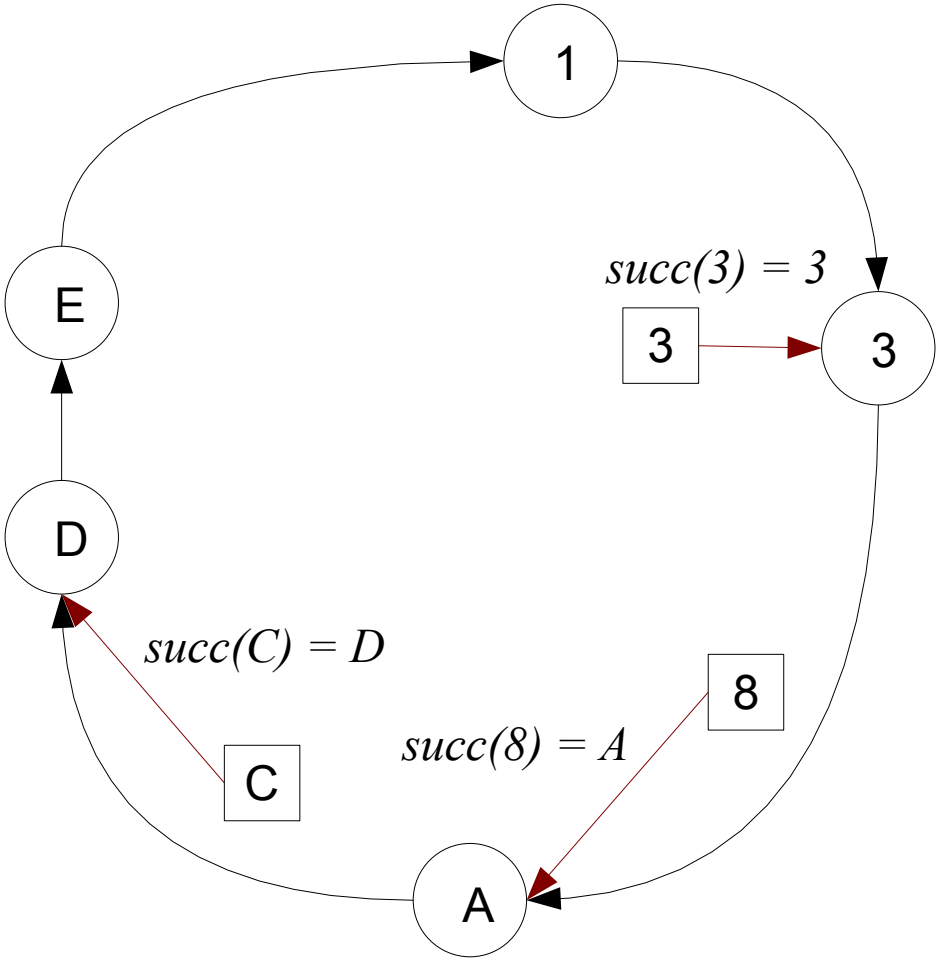
Chord

Eigenschaften:

1.Schlüsselraum: $Restklassenring\ \mathbb{Z}/2^m$

2.Distanzmaß: $d(x,y) = 2^m - (x-y) \bmod 2^m$

3.Routing: Sprungliste



$succ(k) =$ der Knoten n mit n ≥ k und es ist

kein anderer Knoten dazwischen

Sprungtabelle:

$1 \le i \le m$	$finger[i] = (n + 2^{i-1}) \bmod 2^m$	$succ(finger[i])$

Chord

Lookup:

Knoten 0x1: succ(0xC) = ?

1. Suche größten Vorgänger von 0xC in Sprungtabelle von Knoten 0x1:

i	Schlüssel	Knoten
1	0x2	0x3
2	0x3	0x3
3	0x5	0xA
4	0x9	0xA

2. Der Schlüssel 0xC steht in der Spungtabelle von Knoten 0xA:

i	Schlüssel	Knoten
1	0xB	0xD
2	0xC	0xD
3	0xE	0xE
4	0x2	0x3

Antwort: succ(0xC) = 0xD

CAN

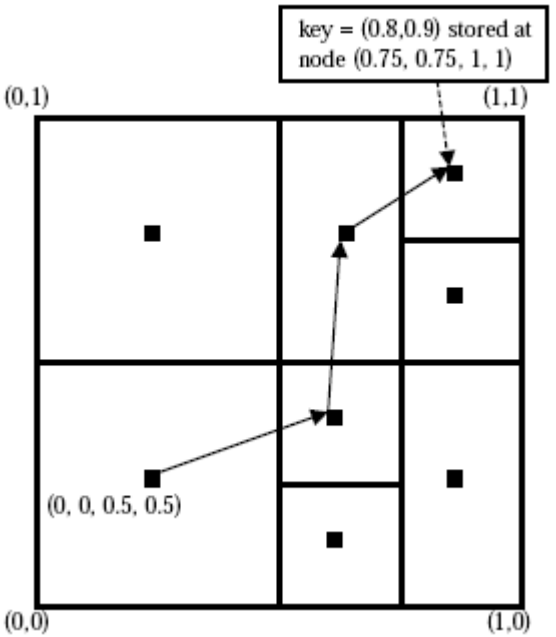
Eigenschaften:

1.Schlüsselraum: $[0,1]^d \subset \mathbb{R}^d$

2.Distanzmaß: $d(x,y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$

3.Routing: Nachbarzone mit dem geringsten Abstand zum gesuchten Schlüssel

Lookup:



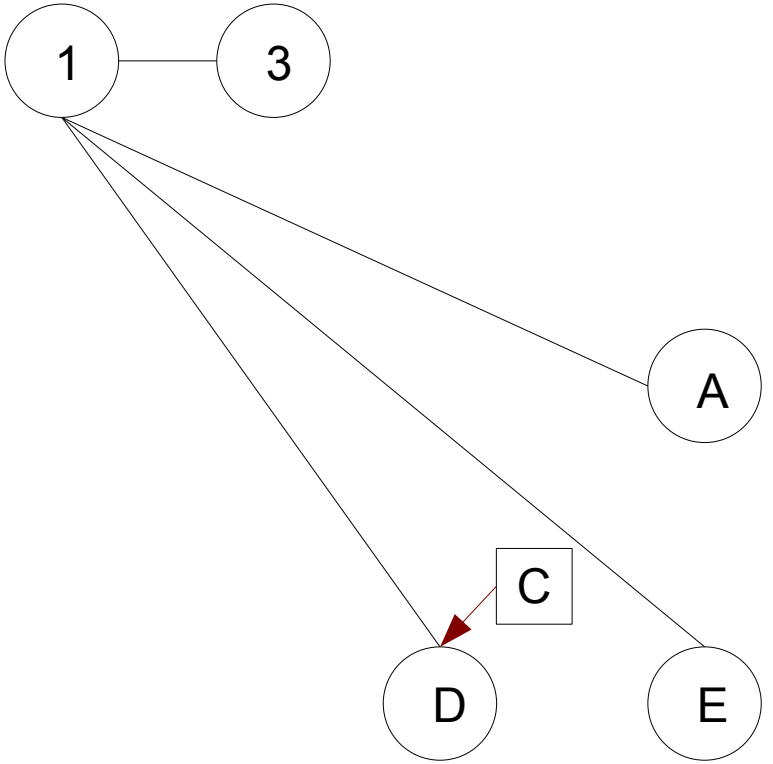
Kademlia

Eigenschaften:

1.Schlüsselraum: Symmetrische Maschen mit 160 Bit Schlüsseln

2.Distanzmaß: d(x,y) = x xor y

3.Routing: Präfixbaum



Distanztabelle:

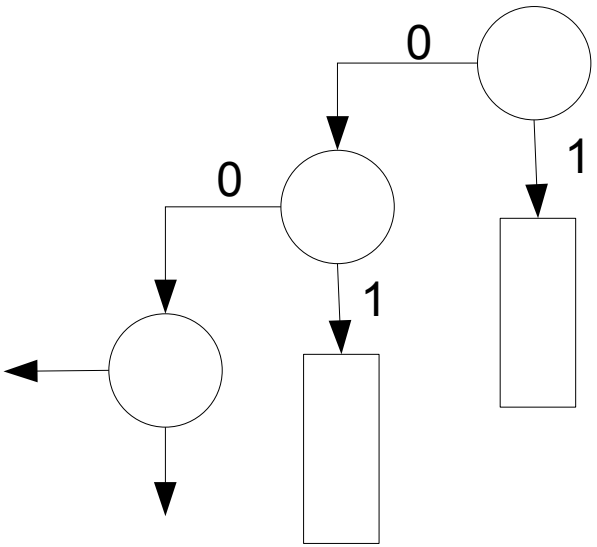
Knoten	0x1	0x3	0xA	0xD	0xE
0x1	0x0	0x2	0xB	0xC	0xF
0x3		0x0	0x9	0xE	0xD
0xA			0x0	0x7	0x4
0xD				0x0	0x3

Kademlia

Routingtabelle:

Intervall	k-Bucket
$[2^0 - 2^1[$	1..k Knoten
$[2^1 - 2^2[$	1..k Knoten
...	...
$[2^i - 2^{i+1}[$	1..k Knoten

Routingbaum:



Subnetzmaske:

- Terminologie vom Internetprotokoll (IP) „geliehen“
- Bezeichnet gleiche Anzahl höherwertiger Bits:

Schlüssel A:	1	0	1	1
Schlüssel B:	1	0	0	1

➡ Schlüssel A und B sind im gleichen /2 Subnetz

Kademlia

Lookup:

Knoten 0x1: Wo ist 0xC gespeichert?

1. Berechne Distanz:: d(0x1, 0xC) = 0xD

2. Suche nächsten Knoten in Routingtabelle:

Maske	Bereich	Knoten
3	0x1	-
2	0x2 – 0x3	0x3
1	0x4 – 0x7	-
0	0x8 – 0xF	0xA,0xD,0xE

3. Frage 0xA nach Schlüssel 0xC:

   3.1. Berechne Distanz: d(0xA, 0xC) = 0x6

Kademlia

Lookup (Fortsetzung):

3.2. Suche nächsten Knoten in Routingtabelle:

Maske	Bereich	Knoten
3	0x1	-
2	0x2 – 0x3	-
1	0x4 – 0x7	0xD, 0xE
0	0x8 – 0xF	0x1, 0x3

3.3. Antwort: Frage 0xD.

4. Frage 0xD nach Schlüssel 0xC: Gefunden!

Knoten	Distanz zu 0xC
0x1	0xD
0x3	0xF
0xA	0x6
0xD	0x1
0xE	0x2

Kademlia

Weitere Eigenschaften:

- Redundanz durch k-Buckets
- Replikation der Daten auf benachbarten Knoten
- k-Buckets werden durch Arbeitskontakte aktuell gehalten
- k-Buckets können nicht „vergiftet werden weil nur inaktive Knoten gelöscht werden

Lookup Effizienz

Motivation:

- Bisher DHTs nur im Labor untersucht
- Bisher nur Worst-Case Betrachtung der Hop Anzahl
- Hier: Kad Netz von eMule mit nahezu 1 Million Nutzer

Zielsetzung:

- Theoretisches Modell zur Abschätzung der mittleren Anzahl Hops
- Neue Werkzeuge (-> Zusammenfassung)
- Empirischer Vergleich von Routing-Strategien
- Empirischer Vergleich von Lookup-Strategien
- Empirische Vergleich von Replikationsstrategien
- Verbesserungsvorschläge für Präfixbasierte DHTs

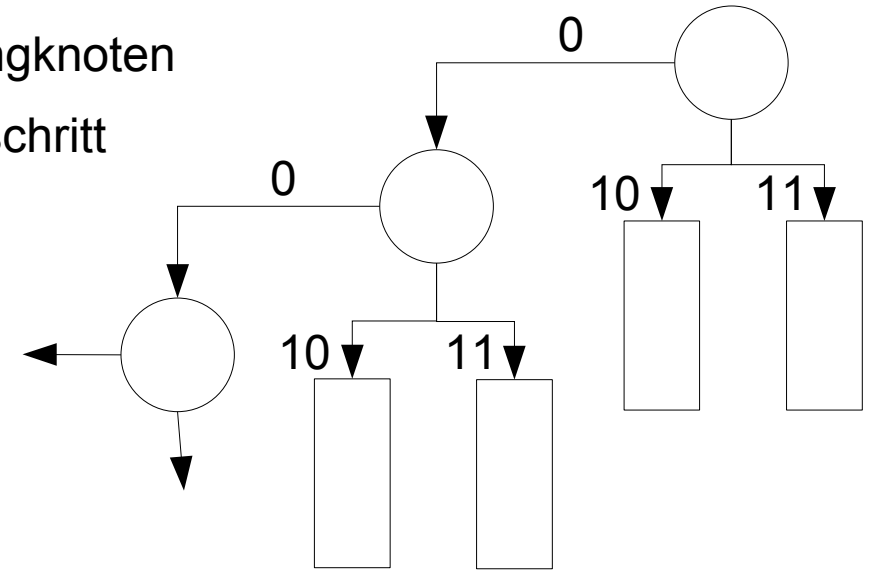
Ideen

Ideen:

- Anzahl Routingeinträge erhöhen
- Anzahl Einträge in den k-Buckets erhöhen

Begriffe:

- Symbolgröße: Anzahl k-Buckets pro Routingknoten
- Auflösung: Anzahl Bits die pro Routingschritt verglichen werden



Redundanz bringt Perfomanz

Theorie

Die Wahrscheinlichkeit, mindestens δ Bit näher ans Ziel zu kommen:

$$Pr[X \geq \delta] = \frac{1}{2^{\delta}}$$

Verteilungsfunktion:

$$F(\delta, r, k) = Pr[X \geq \delta] = 1 - (1 - \frac{1}{2^{r \cdot \delta}})^{k}$$

Wahrscheinlichkeitsfunktion:

$$f(\delta, r, k) = Pr(X = \delta) = F(\delta, r, k) - F(\delta + 1, r, k)$$

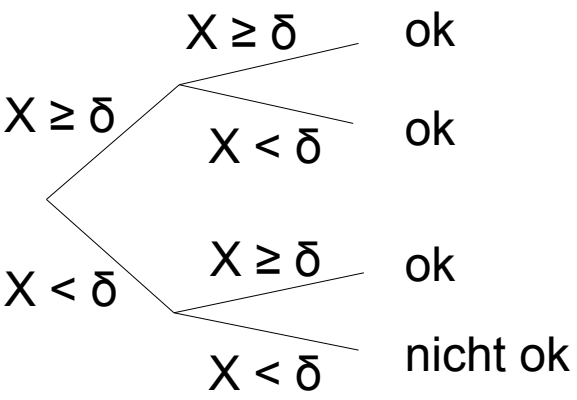
Beispiele

/2 Subnetz von Knoten 0000:

0000
0001
0010
0011

$$Pr[X \geq 1] = \frac{1}{2^{1}}$$

Sei k=2:



Redundanz bring Permormanz

Zufällige Bitverbesserung pro Schritt (Erwartungswert):

$$m(r,k)=r\cdot\sum_{\delta=0}^{\infty}\delta\cdot f(\delta,r,k)$$

Gesamtzahl der Bitverbesserung pro Schritt:

$$t(b,r,k)=b+m(r,k)$$

(garantierte + zufällige Bitverbesserung)

Erwartete Anzahl Schritte für einen Lookup:

$$steps_{avg}=\frac{\log_2(n)}{t(b,r,k)}$$

Verschiedene Symbolgrößen

Systembeschreibung:

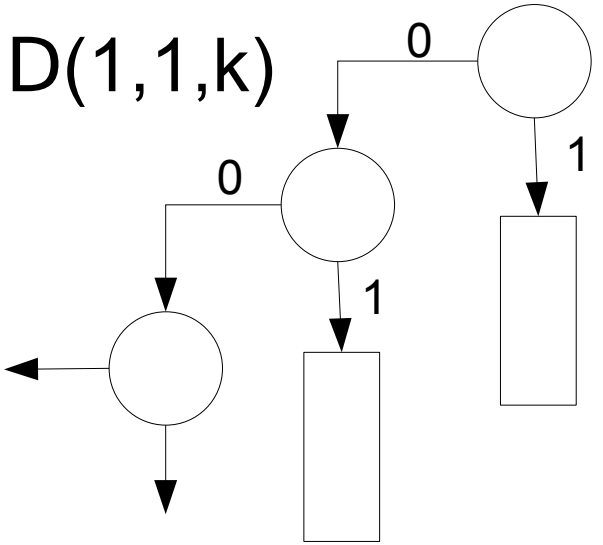
D(b, r, k) mit

b = Symbolgröße

r = Auflösung der inneren Knoten

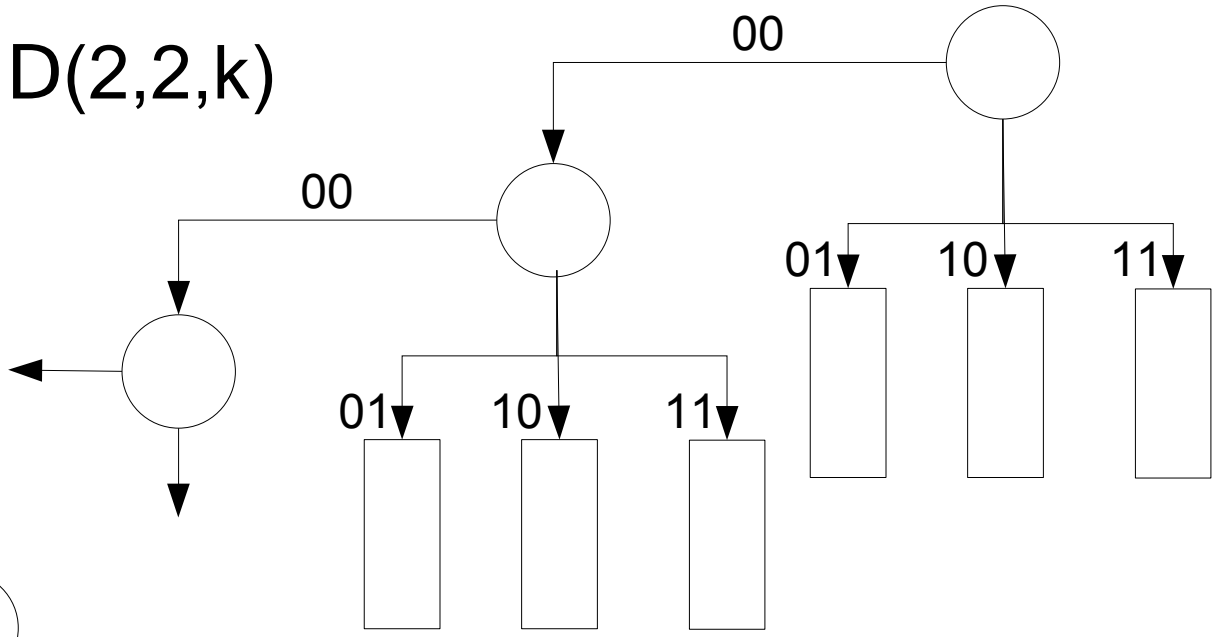
k = Bucket Größe

Standard Kademlia:

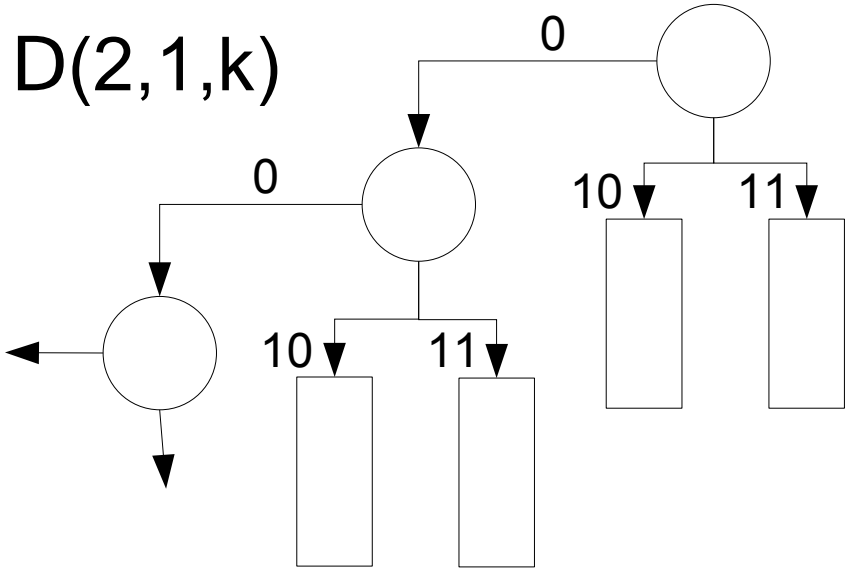


Verschiedene Symbolgrößen

Discrete Symbols:



Split Symbols:



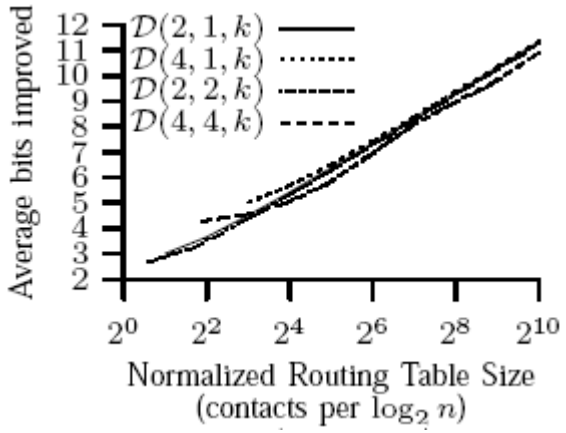
Symbolgröße und k-Buckets

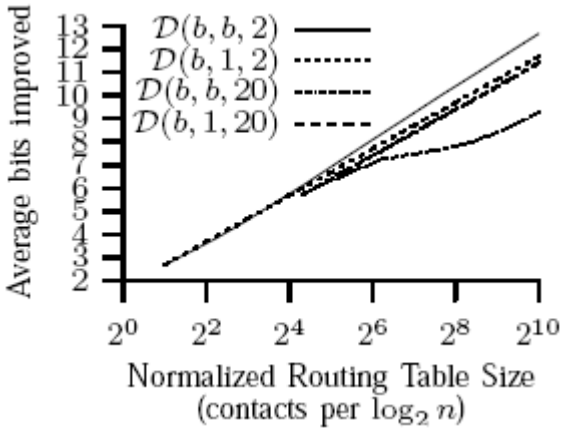
Bit-Gewinn durch komplexere Routingtabellen:

Routingtabellengröße: $k \cdot (2^b - 2^{b-r}) \cdot \log_{2^r} n$

Normalisierte Routingtabellengröße: $k \cdot \frac{2^b - 2^{b-r}}{r}$

: $\log_2 n$



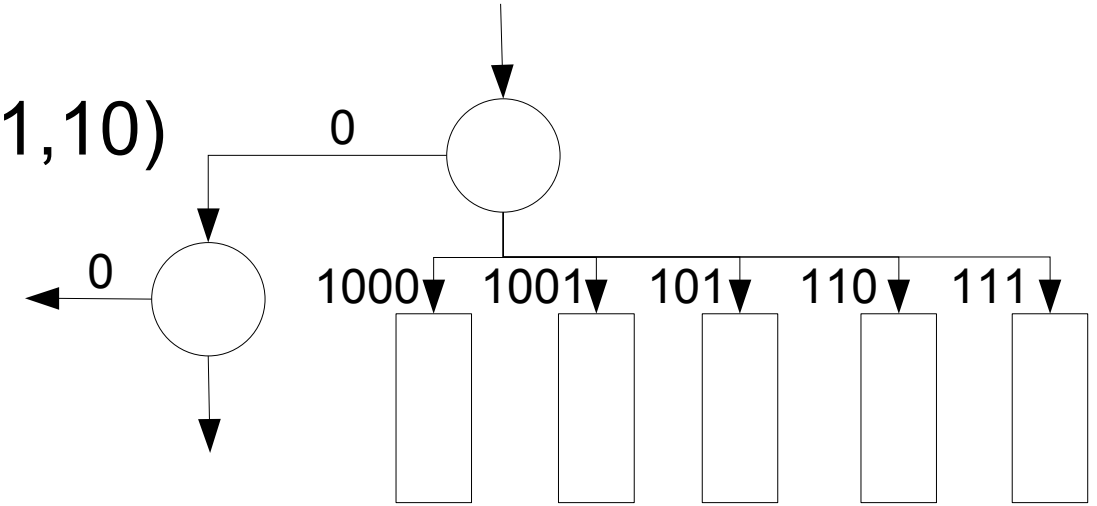


Ergebnisse:

- Komplexere Routingtabellen bringen eine Verbesserung im Worst-Case
- k-Buckets bringen eine Verbesserung im Average-Case
- k-Buckets und komplexe Routingtabellen bringen zusammen keinen Mehrgewinn

Kad

D(3.25,1,10)



Bitverbesserung pro Schritt: $t(3.25,1,10)=3.25+m(1,10)=6.98$

Der Wurzelknoten hat eine vollständige 4-Bit Verästelung.

=> Bitverbesserung für den ersten Schritt:

$$t(4,1,10)=4+m(1,10)=7.73$$

Mittlere Anzahl Suchschritte: $steps_{avg}=1+\frac{\log_2(n)-t(4,1,10)}{t(3.25,1,10)} = 1+\frac{\log_2(n)-7.73}{6.98}$

Kad

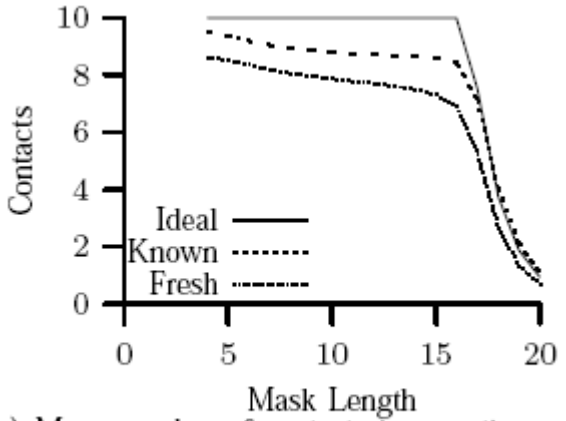
Netzgröße bestimmen:

- Genaue Bestimmung von n nicht möglich, wegen der Dynamik von P2P-Netzen
- Abschätzung von n durch Stichprobenziehung in Subnetzen
- Ergebnis: $n \approx 980{,}000$
- Laufzeit für Lookups im Mittel: 2.7 (mit perfekten Routingtabellen)

Qualität von Routingtabellen

Vollständigkeit:

Verhältnis von IST- zu SOLL-Einträgen in den k-Buckets:



(a) Mean number of contacts in a routing table bucket

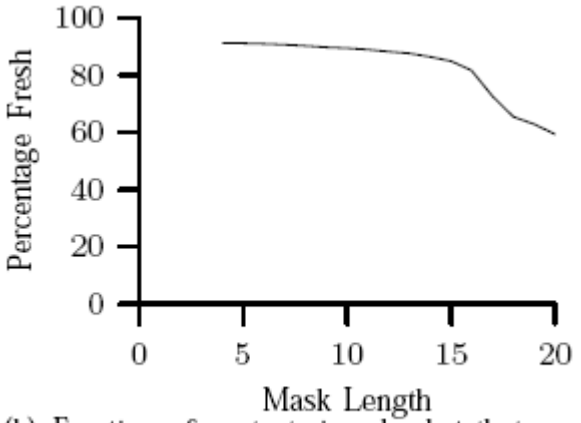
Sei x die Anzahl Bits in der Netzmaske und n die Netzgröße, dann ergeben sich die SOLL-Einträge als:

$$Ideal(x) = min(k, \frac{n}{2^x})$$

Qualität von Routingtabellen

Aktualität:

Anteil von aktiven Knoten im k-Bucket:



(b) Fraction of contacts in a bucket that are fresh (i.e., valid)

Ergebnis:

- 1.5 Plätze sind im Durchschnitt unbesetzt
- 1 Eintrag ist im Durchschnitt nicht mehr aktiv
- Neue Erkenntnisse ins Modell integrieren: $k = 10 - 1.5 - 1 = 7.5$

→ $\frac{\log_2(n) - 7.33}{6.58} + 1 = 2.91\ Hops$ (empirisches Ergebnis: maximal 3.2 Hops)

Lookup-Strategien

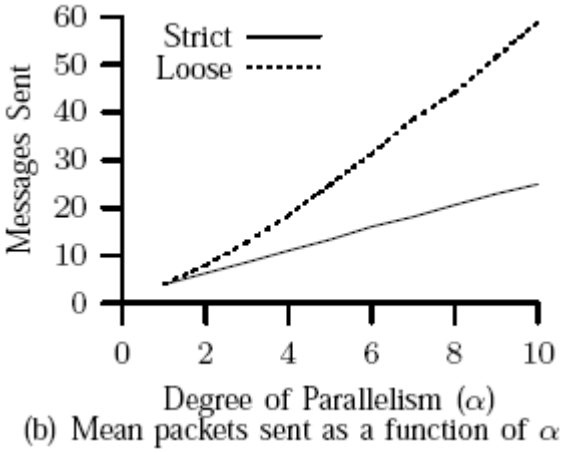
Unabhängige Parameter:

- Parallelisierungsgrad: α
- Strategien:
  - Strikte Parallelität: „A new request is issued only when a pending request times out or a response is received.“ (D. Stutzbach, R. Rejaie)
  - Lockere Parallelität: „[...] a client can issue a lookup request to a contact that is among the top α contacts as soon as such a contact is identified [...]“ (D. Stutzbach, R. Rejaie)

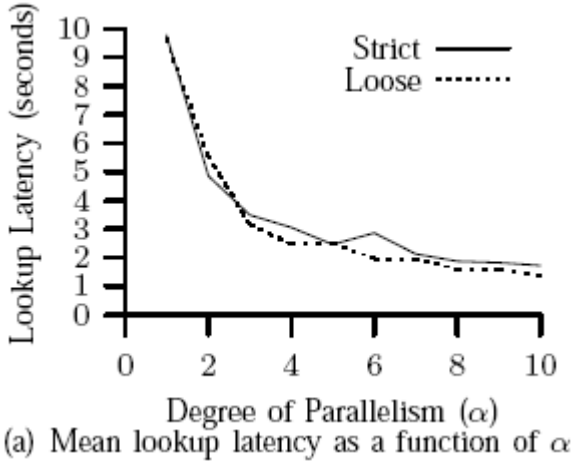
Abhängige Parameter:

- Latenzzeit: Ergebniszeit – Startzeit
- Overhead: Anzahl gesendeter Nachrichten
- Hops: Anzahl gefragter Knoten

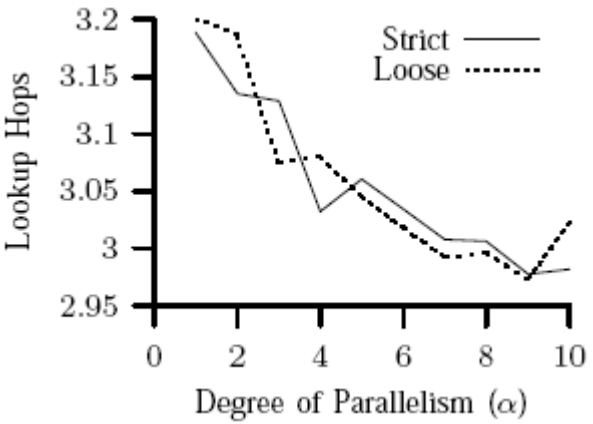
Lookup-Strategien



(b) Mean packets sent as a function of α



(a) Mean lookup latency as a function of α



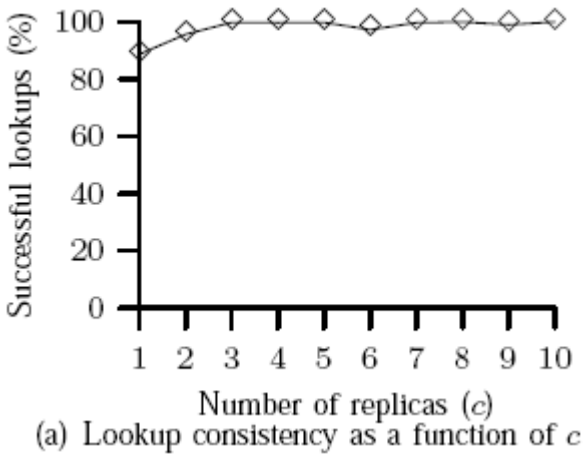
Ergebnisse:

- Strikte Parallelisierung leistet das Gleiche wie lockere, erzeugt aber weniger Overhead.
- Der optimale Parallelisierungsgrad liegt bei α = 3.
- Bei α = 1 erreicht ein Lookup nach 3.2 Hops den Zielknoten, bei α > 1 in weniger Hops.

Konsistenz von Suchanfragen

Fragestellung:

Was ist der optimale Replikationsparameter c, um eine Zuverlässigkeit von mindestens p zu garantieren?



(a) Lookup consistency as a function of c

Ergebnis:

Ab c=3 erreicht man eine Zuverlässigkeit von über 99,9 %

Zusammenfassung

Ergebnisse:

- k-Buckets bringen Redundanz UND Performanz
- k-Buckets und komplexere Routingtabellen bringen keinen Vorteil
- Anzahl Hops: SOLL = 2.91
- Anzahl Hops bei α = 1: IST = 3.2
- Anzahl Hops bei anderen Autoren: SOLL = 6.3
- Effizienteste Lookup-Strategie: Strikte Parallelität mit α = 3.
- p = 99,9 % wird erreicht mit: c = 3

eMule:

- Lookup Strategie: lockere Parallelität mit α = 3
- Routing Strategie: k-Buckets und Split Symbols
- Replikationsfaktor: c = 19 (im Durchschnitt)

Zusammenfassung

Neue Werkzeuge:

- cruiser: Liefert für ein gegebenes DHT-Subnetz die Anzahl der aktiven Peers.
- kFetch: Lädt von einem Peer die Routingtabelle herunter und testet die Einträge.
- kLookup: Führt von einer beliebigen Quelle einen Lookup nach einem Ziel aus. Dabei nutzt es kFetch, um die Routingtabelle der Quelle zu holen.