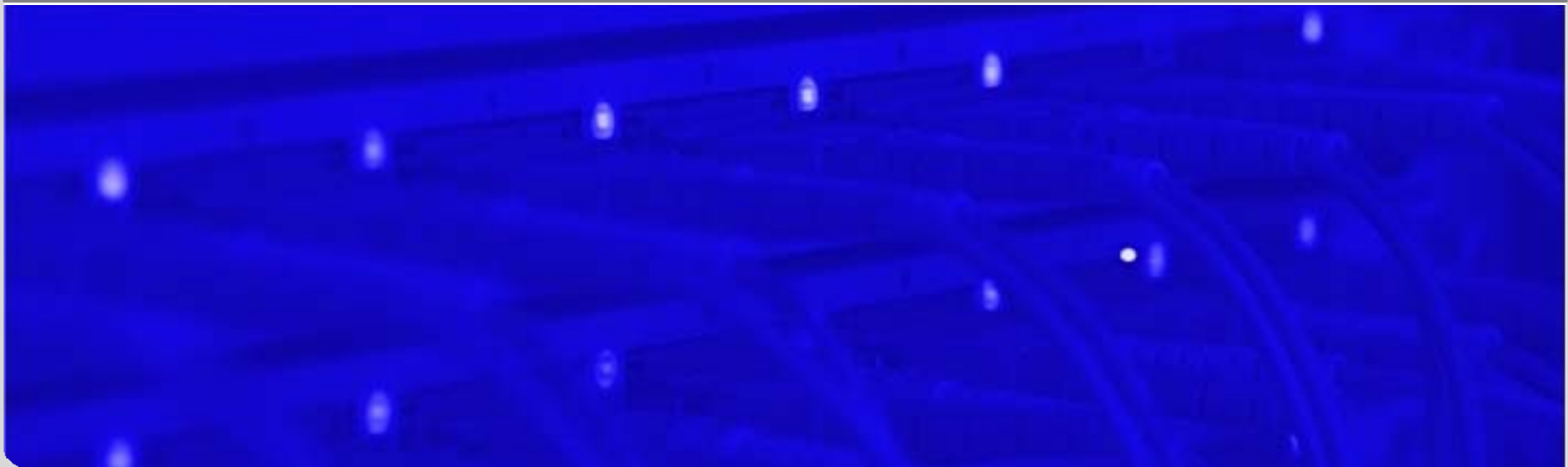


Next Generation Internet

10. Peer-to-Peer (P2P)

INSTITUT FÜR TELEMATIK



Überblick Kapitel 10

I. Einführung

1. Einführung

II. Internet-Architektur

2. Internet-Architektur
3. NAT & IPv6
4. Dienstgüte

III. Multicast

5. Grundlagen
6. Multicast Routing
7. Multicast Transport

IV. Flexible Dienste und Services

8. Neuere Transportprotokolle
9. Flexible Netze
10. Peer-to-Peer

10.1 Motivation und Überblick
10.2 Unstrukturierte Overlays
10.3 Strukturierte Overlays
10.4 Verteilte Hash-Tabellen
10.5 Aktuelle Forschungsarbeiten im Institut
10.6 Dezentrales Bootstrapping mit IPv6

10.1 Motivation und Überblick

- Peer-to-Peer-Paradigma, das ...
 - ... neue Möglichkeiten für das Internet eröffnet
 - ... lange diskutierte Probleme lösen kann
 - ... aber auch viele Fragestellungen aufwirft
 - vor allem bezüglich Sicherheit, sowie Rechtsfragen
- Rein begrifflich: „Peer-to-Peer (P2P)“ im Gegensatz zu „Client/Server“ weist auf gleichberechtigte Partner hin
- Macht sich zu Nutze, dass die Intelligenz in den Endsystemen sitzt (Ende-zu-Ende-Argument)
- Am Netzrand ist Innovation jederzeit möglich!

Client/Server-Paradigma

■ Traditionell: Client/Server

■ Verkehrsaufkommen

- Konzentration am Server; Unterlast in anderen Netzteilen
- Asymmetrischer Verkehr
- Erhöhung der Bandbreiten nutzt nur an wenigen Stellen im Netz
- Problem der Skalierbarkeit (Speicherplatz, Rechenleistung)

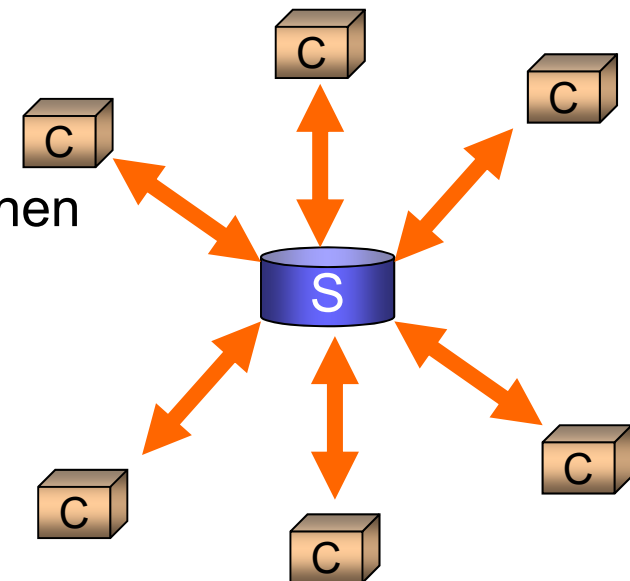
■ Information im Netz

- World Wide Web
Request/Response-Protokoll: HTTP
- Inhalt nicht komplett von Suchmaschinen erfasst bzw. erfassbar

■ Ungenutzte Ressourcen

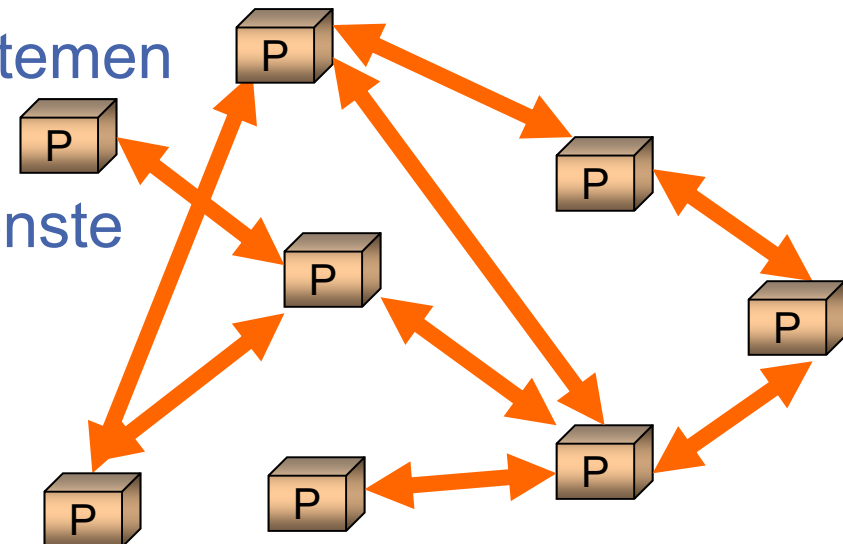
- CPU, Speicherplatz, Information

■ Server ist Single-Point-of-Failure



Definition von Peer-to-Peer-Systemen

- Peer-to-Peer-System = Verteiltes System bzw. Anwendung charakterisiert durch
 - Interaktion von (gleichberechtigten) Endsystemen (Peers)
 - Gemeinsame Nutzung von Ressourcen in Endsystemen
 - Keine zentrale Kontrolle oder Nutzung zentraler Dienste
 - Gleichberechtigte und autonome Systeme
 - Selbstorganisation des Systems



Eigenschaften von Peer-to-Peer-Systemen (1)

■ Interaktion von Endsystemen

- Dienstleistung durch direkte Kooperation der Endsysteme
- Minimale Anforderungen an Infrastruktur (Netzwerk, Dienste)
- Einfache Umsetzung

■ Gemeinsame Nutzung der Ressourcen in Endsystemen

- Umgehung des Flaschenhalses einer zentralen Realisierung
- Redundanz durch Verteilung und/oder Replikation

Eigenschaften von Peer-to-Peer-Systemen (2)

■ Keine Nutzung zentraler Dienste

- Ausfallsicherheit, Skalierbarkeit
- Heimanwender werden selbst zu Dienstanbieter

■ Alle Peers sind gleichberechtigt

- Diese Aussage trifft nicht immer zu
- Es gibt Peers mit besonderen bzw. speziellen Aufgaben

■ Besser formuliert:

- Dynamische Aufteilung der Aufgaben zwischen Peers
- bei gemeinsamer, verteilter Diensterbringung

■ Warum?

- Server sind “Single point of failure”, außerdem oft ungenutzte Ressourcen in Endsystemen

Definition von Peer-to-Peer-Systemen

■ Selbstorganisation des Systems

- zentrale Eigenschaft von Peer-to-Peer-Systemen
- Impliziert durch vorherige Eigenschaften

■ Definition eines Peer-to-Peer-Systems:

Selbstorganisierendes System gleichberechtigter, autonomer Systeme ohne Nutzung zentraler Dienste auf der Basis eines unzuverlässigen Netzwerks.

Folgen der Definition (Beispiele)

■ Keine echten P2P-Systeme:

■ Napster

- Zwar Nutzung verteilter Ressourcen, aber zentraler Server

■ SETI@Home

- Nutzung verteilter Rechenleistung, aber sonst Client-Server

■ Echte P2P-Systeme:

■ Bittorrent

- Aktuelle Filesharing-Applikation (trackerless)

■ Gnutella

- Vollständig selbstorganisierend (leider nicht skalierbar)

■ Distributed Hash Tables – DHT

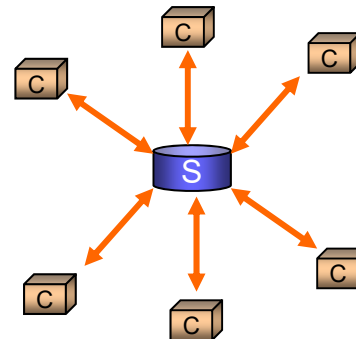
■ Aber auch IP-Routing

- IP-Knoten als Endsysteme, Links als Nachrichtendienst

Klassifikation von Peer-to-Peer-Systemen

■ Client-Server

- Klassische Rollenverteilung
- Keine Interaktion zwischen Clients

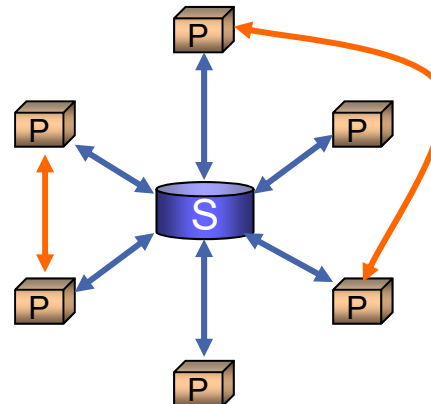


Beispiele:

- WWW
- DNS

■ Hybrides P2P-System

- Gemeinsame Nutzung verteilter Ressourcen
- Interaktion zwischen Peers
- Einsatz von Servern zur Koordination

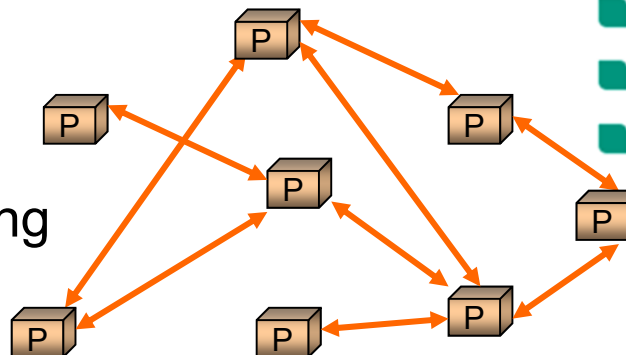


- Seti@Home
- Distributed.net

- Napster
- ICQ
- AIM (AOL)

■ Reines P2P-System

- Vollständige dezentrale Organisation und Nutzung der Ressourcen



- Gnutella
- Freenet
- DHT-Anwendungen (Past, i3, Ocean-Store, ...)

■ Peer-to-Peer-Networking $\neq$ Overlay-Netze

■ Definition Overlay-Netz:

- “Zwei Anwendungen kommunizieren über einen anderen Pfad, als durch die Netzwerkschicht vorgegeben wird”

- **Logisches Netz** über unterliegender Topologie

■ Eigene Wegewahl, oft **eigene Adressierung**

■ Organisationsform: **strukturierte/unstrukturierte Overlays**

■ P2P-Anwendungen benutzen meistens Overlay-Netze

- DHTs, Gnutella, ...

■ P2P-Anwendungen ohne Overlay:

- Ad-hoc-Netze

■ Overlay-Netze, die nicht P2P sind:

- VPNs, MPLS, DiffServ, etc.

Ist P2P neu?

- P2P war die **ursprüngliche Idee des Internet**
 - Vernetzung gleichwertiger Teilnehmer/Endsysteme
 - Jedes Endsystem stellt Dienste bereit
 - Unzuverlässige Infrastruktur (früher waren Ausfälle problematisch – heute eher Attacken, Mobilität, DHCP)
 - Vollständige Selbstorganisation
 - **Internet-Routing (BGP)** arbeitet nach **Peer-to-Peer-Prinzip**
 - Router organisieren selbstständig Erreichbarkeit
 - Router sind autonom und weder Client noch Server
 - Internet war/ist „Overlay“ über Telefonnetz
 - Beispiele
 - E-Mail
 - Usenet
 - keine zentrale Kontrolle
 - Kopieren von Dateien (Mail, News, usw.) zwischen Computern
 - DNS (P2P mit hierarchischer Organisation)
- aber dann kam erst einmal Client/Server (Siegeszug des WWW)...

P2P-Anwendungen

■ File Sharing

- BitTorrent
- eDonkey (eMule)
- Gnutella
- FastTrack (KaZaA, Morpheus, ...)

■ Distributed Computation

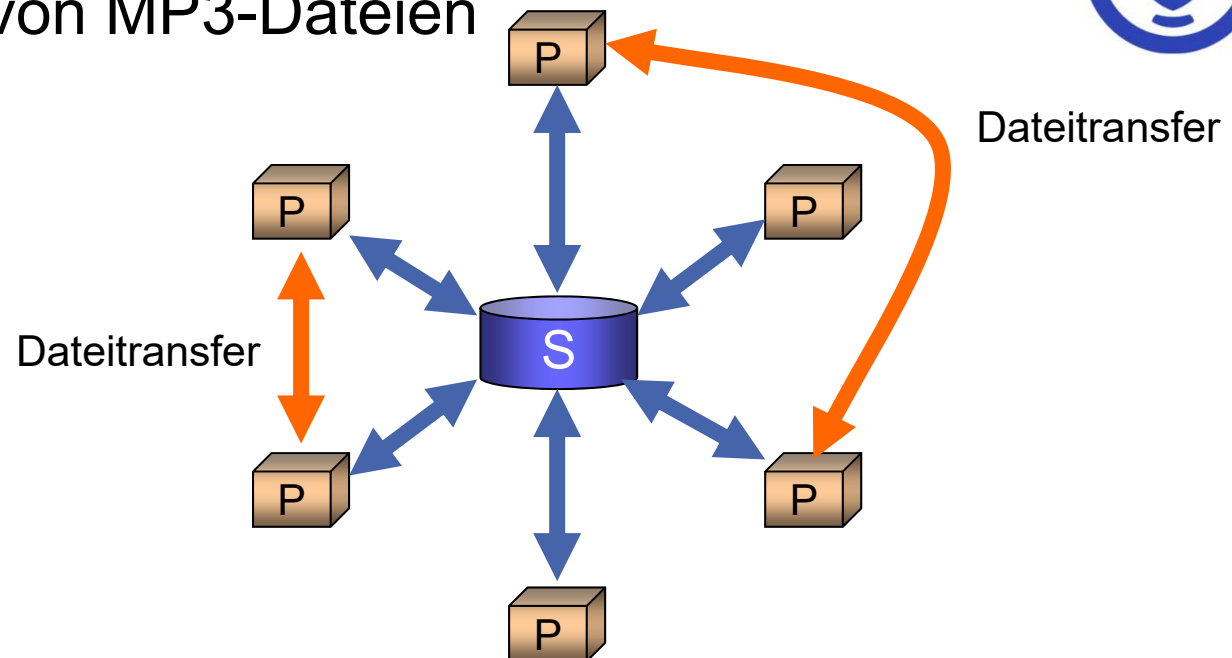
- Popular Power
- SETI@home
- Distributed.net

■ auch Instant Messaging, verteilte Verzeichnis- oder Collaborationsdienste

P2P-Anwendungen – Napster

- Erste „P2P“-Killer-Applikation (1999–2001)

- Austausch von MP3-Dateien



- Zentraler Verzeichnissserver – also kein echtes P2P

- Verwaltet Adressen und Dateilisten der Peers
 - Speichert Dateien nicht selbst

10.2 Unstrukturiertes Overlay BitTorrent

■ BitTorrent (Datensturzflut)

- Aktuell sehr weit verbreitete File-Sharing-Anwendung (bis zu 50% am P2P-Datenaufkommen)

■ Sprung in die Legalität

- Softwareverteilung (Red Hat, Novell, Blizzard)
- Legale Medieninhalte (Videos)
- Aber auch noch jede Menge illegale Inhalte

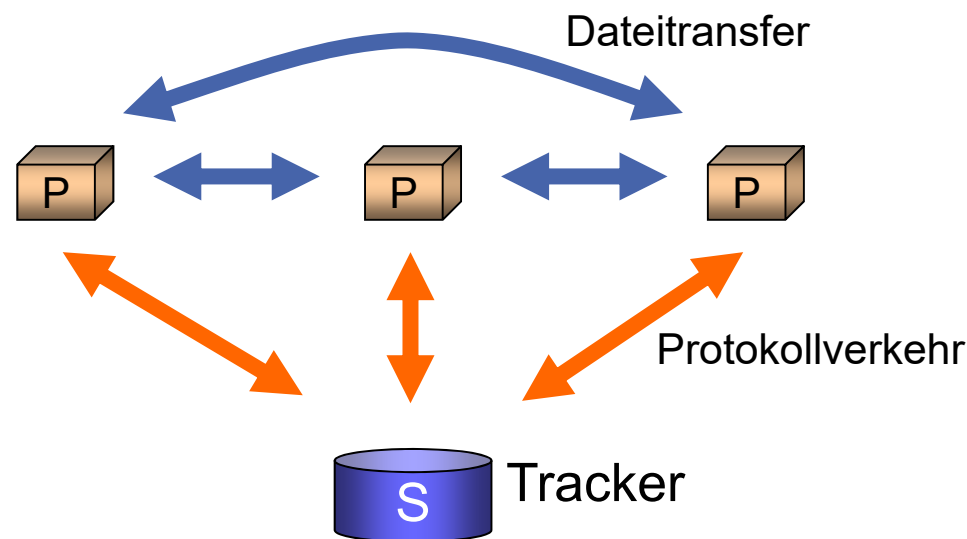
■ Lokalisierung einer Datei über .torrent-Datei

- Kann z.B. auf Webseite veröffentlicht werden
- Enthält Informationen über zugehörigen Tracker

Tracker Mode

■ Hybride Variante

- Indexverwaltung auf zentralen **Server (Tracker)**
- Tracker verantwortlich für alle Peers (**Schwarm**), die Interesse an einer bestimmten Datei haben
- Konzept sehr ähnlich zu Napster, nur effizienter (Datei ist bei allen Mitgliedern des Schwarms verfügbar)



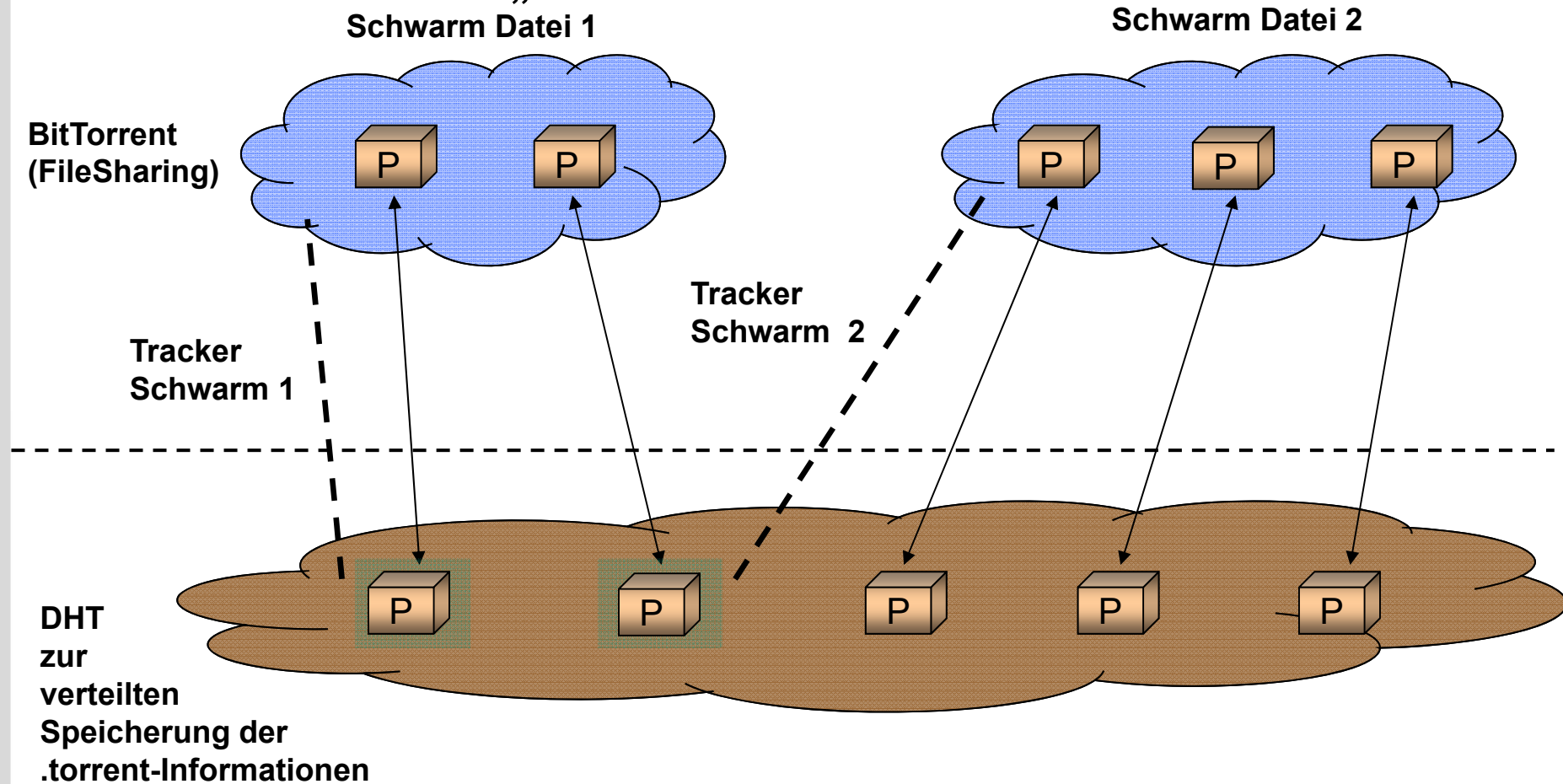
Trackerless Mode (1)

- Erweiterung um **Trackerless Mode** (ab 2005)
 - Keine Abhängigkeit von Servern (Trackern)
 - Rechtlich nicht mehr leicht angreifbar

- Nutzung einer **verteilten Hash-Tabelle** auf Basis eines **Kademlia-Overlays**
 - Alle Peers bilden gemeinsames Overlay zur Speicherung und Verwaltung von .torrent-Dateien

Trackerless Mode (2)

■ Jeder Peer in „zwei“ Netzen



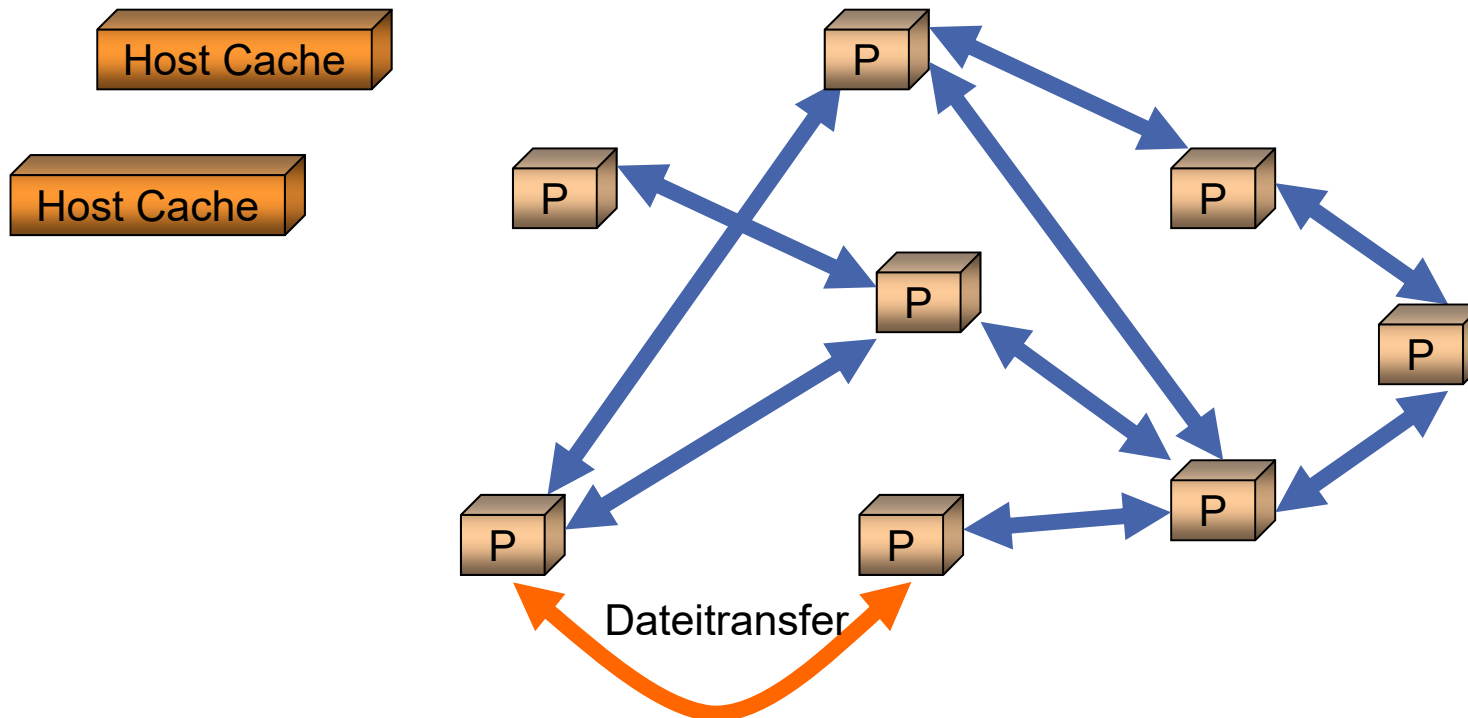
Zusammenfassung BitTorrent

- Sehr hohe akkumulierte verfügbare Bandbreite
 - sehr hohe Skalierbarkeit
 - Hohe Robustheit
- Trotzdem hohe Effizienz
 - Jede Datei wird in Stücken von verschiedenen Peers parallel runtergeladen
 - Damit **gute Auslastung trotz geringerem Upload**
- Weitere Entwicklungen
 - **Verschleierung** (keine Drosselung durch ISPs)
 - **Anonymität** (TOR-Unterstützung)

10.2.1 Unstrukturiertes Overlay Gnutella

■ Eigenschaften

- Dezentrales Filesharing (ab 2000)
 - Host Caches stellen Einstiegspunkte in das P2P-Netz zur Verfügung
- Unterstützt beliebige Dateiformate
- Peers heißen hier **Servents** (Kombination aus Server und Client)



Funktionsweise

- Peers durch TCP-Verbindungen verbunden
- Anfragen werden über das Gnutella-Netz geflutet
- Erkennung von Routingschleifen durch pseudo-eindeutige Message-ID (UUID)
- Begrenzung des „Horizonts“ durch TTL
- Antworten werden auf Basis der UUID geroutet

Bootstrapping

- Initiales Verbinden mit dem Gnutella-Netz (Bootstrapping)
 - Geschieht durch Verbinden mit einem beliebigen Peer innerhalb des Netzes
 - Das Auffinden eines solchen Peers ist nicht Teil des Gnutella-Protokolls
 - Host Caches stellen Listen mit Peers bereit
 - Standard-Host Caches sind im Gnutella-Servent meist vorkonfiguriert
 - Weitere Host Caches werden z.B. auf Webseiten veröffentlicht

Skalierbarkeit

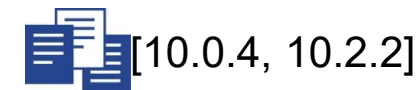
■ 1. Gnutella-Generation

- Zusammenbruch des Gnutella-Netzes im August 2000

■ 2. Gnutella-Generation

- Anzahl der Verbindungen eines Peers werden der verfügbaren Bandbreite angepasst
- Abbruch von Verbindungen zu überlasteten Peers
- Skalierbarkeit immer noch nicht zufriedenstellend

■ 3. Gnutella-Generation

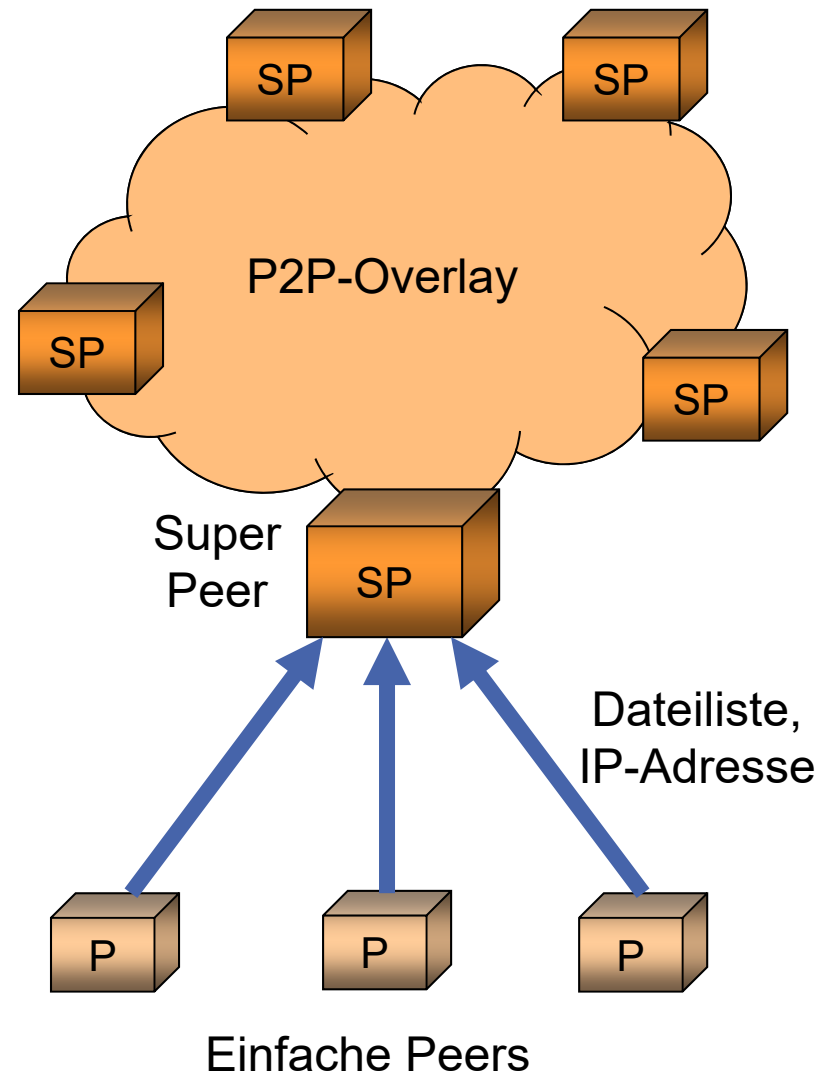


- Einführung von Hierarchien
- „Super-Peers“ übernehmen Netzlast für schmalbandig angebundene Peers
- Kein reines P2P mehr
- Architektur ähnlich wie bei FastTrack

FastTrack

- Wird benutzt von Morpheus, KaZaA, ...
- Erweiterung von Gnutella
- Super Peers
 - Einfache Peers haben nur Verbindungen zu einigen wenigen Super Peers
 - Einfache Peers teilen Super Peer ihre IP-Adresse und Dateiliste mit
 - Super Peers agieren als Proxy für einfache Peers
 - Anfragen an einfache Peers können von Super Peer beantwortet werden
 - Entlastung der einfachen Peers
 - Peers werden je nach verfügbarer Bandbreite und CPU-Leistung zu Super Peers bestimmt
 - Konfigurierbar

FastTrack



Gnutella Analysen

■ Analysen

- fast alle Broadcast-Nachrichten erreichen fast alle Knoten  [10.0.7]
 - Mittlere Anzahl von Verbindungen pro Knoten ist gleichgeblieben, auch wenn das Netz um zwei Größenordnungen gewachsen ist
 - Topologie
 - „Power Law Netz“
 - Knoten mit vielen Verbindungen werden stark belastet
 - Viel Overhead, wenig Netzverkehr
 - Benötigte Bandbreite pro Request wächst linear mit der Netzgröße
 - Nur 36% des Verkehrs für Query-Nachrichten
 - 55% für Ping und Pong
 - 9% für Push
- Anpassungen des Protokolls lösen dieses Problem teilweise

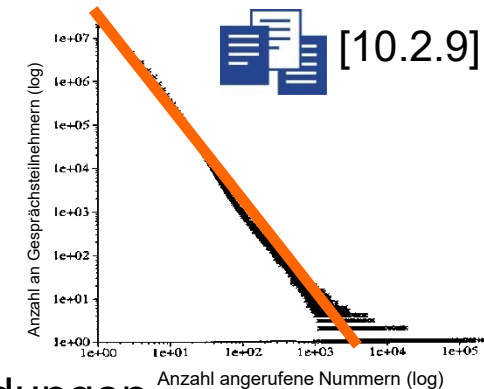
■ Pfadlänge und Bandbreite

- Mittlere Pfadlänge wächst logarithmisch mit der Netzgröße  [10.0.2]
- Benötigte Bandbreite pro Request wächst linear mit der Netzgröße
 - Nachteil von TCP-Broadcast

Small World Netze

■ Power-Law-Eigenschaft

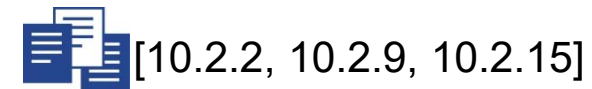
- Unabhängig von Netzgröße und Zeitpunkt
- Anzahl der Knoten mit L Kanten ist proportional zu L^{-k}
 - $k > 0$ ist eine Netz-spezifische Konstante
- Bedeutet: Die **Mehrheit** der Knoten besitzen relativ **wenige** Verbindungen
Einige **wenige** Knoten besitzen aber sehr **vielen** Verbindungen



■ Fehlertoleranz

- Robust gegen zufällige Ausfälle
 - Zumeist sind Knoten mit nur wenigen Verbindungen betroffen
- Anfällig gegen gezielte Angriffe auf Knoten mit vielen Verbindungen

■ Problem



- Small World Netze garantieren nur die **Existenz** von kurzen Wegen zwischen beliebigen Ecken
- Der Routingalgorithmus muss diese Wege auch finden!
- Vor allem die wenigen Knoten mit vielen Verbindungen spielen dabei eine große Rolle

10.3 Strukturierte Overlay-Netze

■ Definitionen **Overlay**:

- “Zwei Anwendungen kommunizieren über einen anderen Pfad, als durch die Netzwerkschicht vorgegeben wird”
- **Logisches Netz** über unterliegender Topologie (Underlay)
- **Eigene Wegewahl**, oft **eigene Adressierung**

■ **Strukturierte Overlay-Netze**

- Aufbau der Topologie folgt einer festen Regel → dadurch deterministisches Routing möglich
- Spezifische Graphenstruktur erlaubt Objekte effizient zu lokalisieren
- Effizient (N = Anzahl der Overlay-Knoten):
 - Austausch von höchstens $O(\log N)$ Nachrichten
 - Speicheraufwand f. Routinginformation
 - Kommunikationsaufwand f. Suche (#Nachrichten u. Latenz)
 - Wartungsaufwand f. Routingtabelle

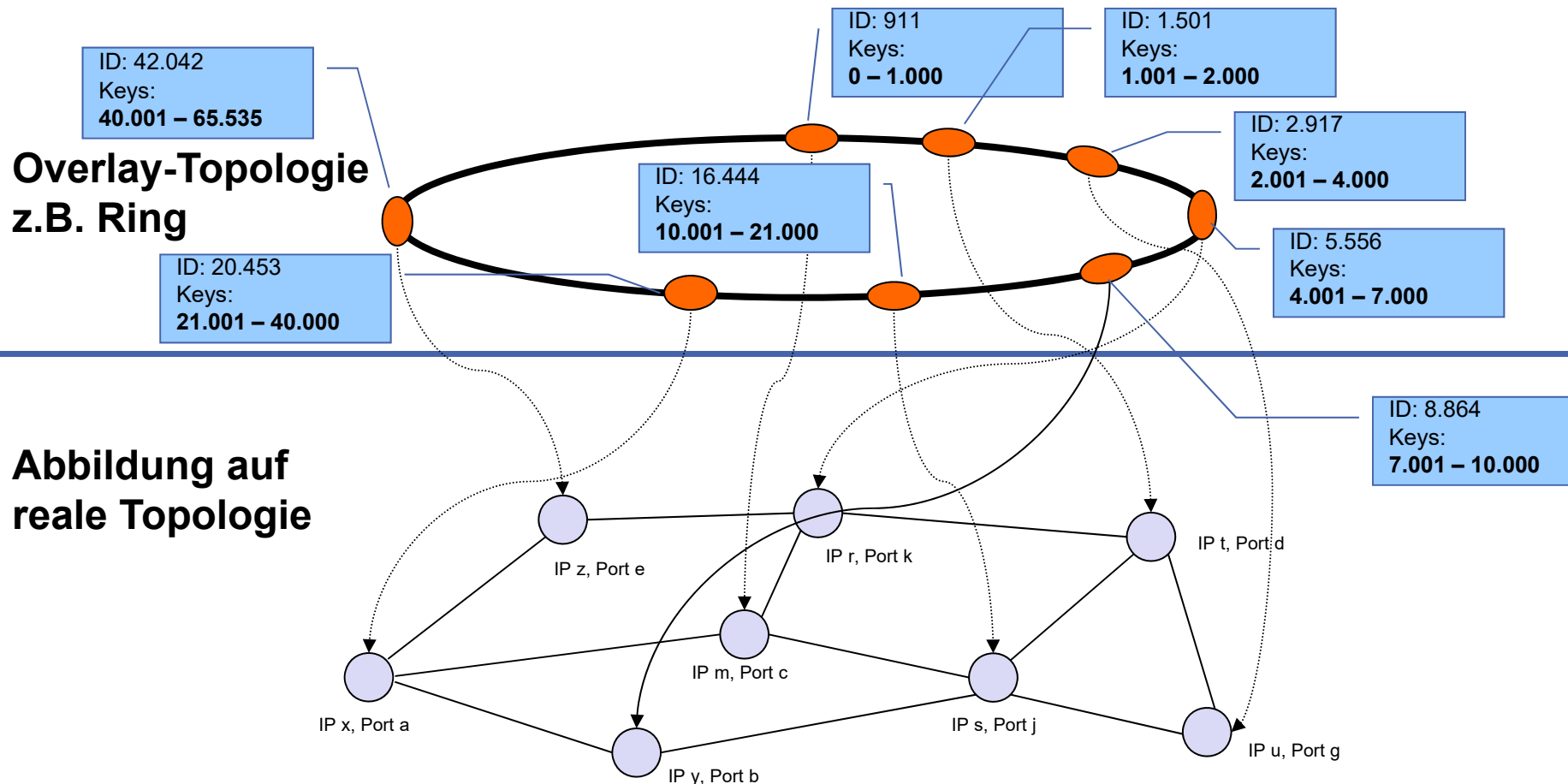
Key-based Routing

■ Overlay-Netz

- **Overlay-Knoten**: Instanz eines Overlay-Teilnehmers
 - ist einem physikalischen Knoten im Underlay zugeordnet
 - es können mehrere Overlay-Knoten in einem physikalischen Knoten existieren
 - Overlay-Knoten wird durch **NodeID** $\in$ ID-Space identifiziert (Knotenadresse), ID-Space z.B. $[0 \dots 2^{160} - 1]$
- Schlüsselwert (Key) **K** $\in$ ID-Space wird einem Knoten zugeordnet (Wurzelknoten für K)
- Overlay-Knoten besitzt **Routingtabelle**
 - Eintrag: Nachbar im Overlay $\rightarrow$ physikalische Knoten-Adresse im Underlay (z.B. IP-Adresse)
- **Key-Based Routing**: übermittle Nachricht an Wurzelknoten von K
 - Kann zur Lokalisierung eines Objekts dienen

Beispiel für Overlay

- Jeder Knoten ist für mindestens einen Teil des Wertebereichs zuständig
 - Oftmals Redundanz
 - Ständige Anpassungen: Knoten fallen weg, kommen neu hinzu



Einordnung

 [10.3.11]

■ Primitive von Key-Based Routing

- `route (Key K, Msg M, NodeHandle Hint)`
 - Nachricht M an für K zuständigen Knoten ggf. via Next hop Hint
- `forward (Key K, Msg M, NodeHandle Hint)`
- `deliver (Key K, Msg M)`

■ Key-Based Routing ist Grundlage für Anwendungen wie

- verteilte Hash-Tabelle (DHT)
 - Realisiert Abbildung **Key** → **Objekt**
- Dezentrale Objektlokalisierung und Routing
- Multicast/Anycast-Dienste

→ Merke: Lokalisierung (Lookup) ≠ Suche!

10.3.1 Chord

■ Key-Based-Routing-Verfahren

■ Probleme:

- Zuordnung Schlüssel $\rightarrow$ Knoten
- Für schnelles Lookup: Finde Knoten, der (Key, Value)-Paar verwaltet möglichst effizient

■ Liefert Abbildung **Schlüssel $K \rightarrow$ Knoten**

■ Lookup-Aufwand = Routing-Aufwand: $O(\log N)$

■ Knoten-Komplexität = Speicher-Aufwand: $O(\log N)$



Chord-Prinzip

■ Ringförmiger Wertebereich

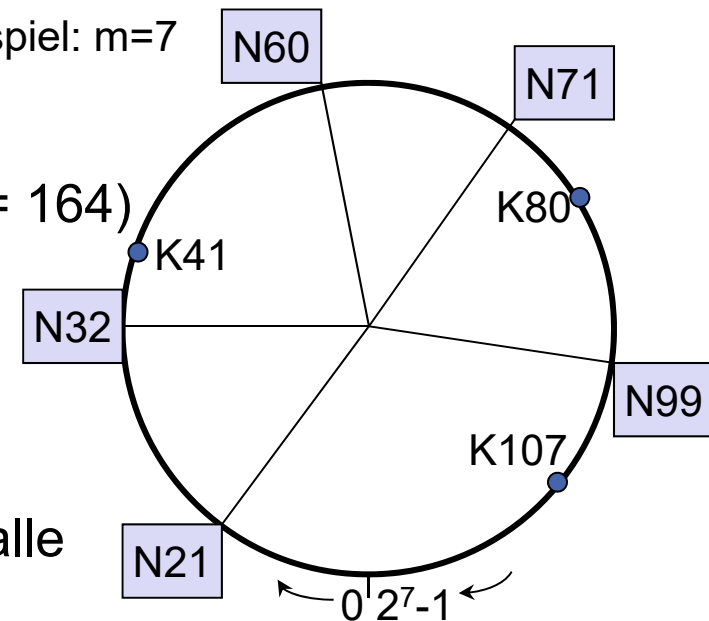
- $K = ID \in [0..2^m-1]$
- Empfohlene Hash-Funktion: SHA-1 ($m = 164$)

■ Zuordnung der Schlüssel zu Knoten?

■ Basiert auf Consistent-Hashing

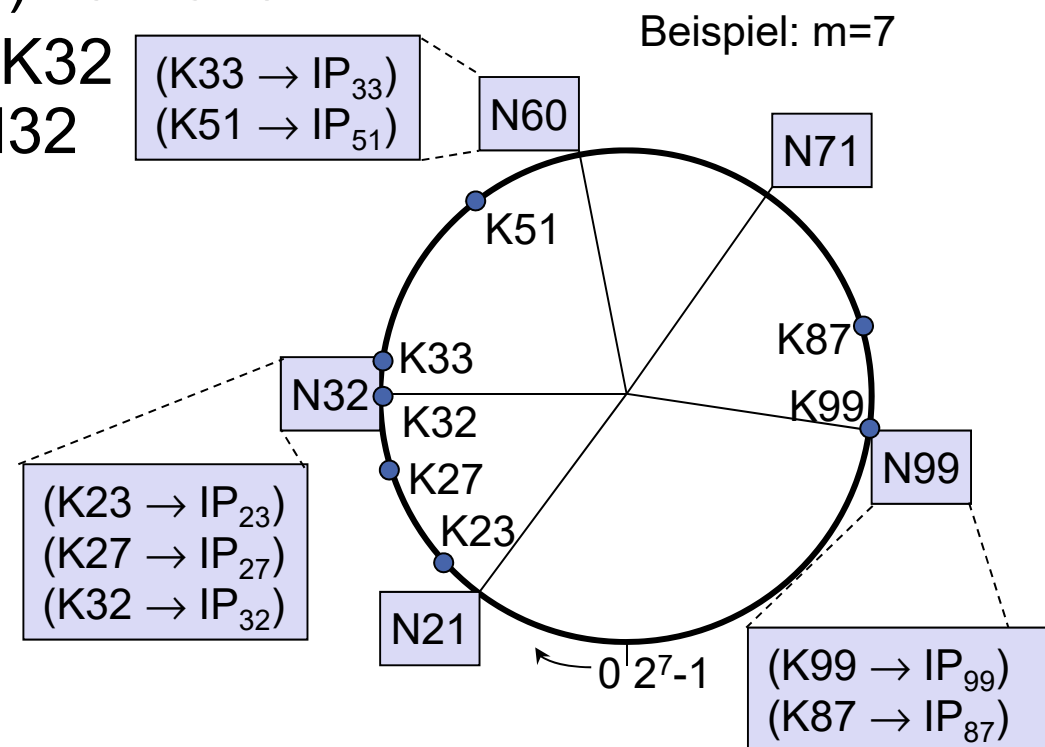
- verteilt Schlüssel gleichmäßig über alle Knoten
- Bei Ein- bzw. Austreten eines Knotens N muss lediglich ein $O(1/N)$ -Bruchteil der Schlüssel bewegt werden
- Knoten-Adressen werden in Wertebereich abgebildet, d.h. Knoten-IDs und Keys liegen im gleichen Wertebereich
 - **Knoten-ID**: $N_x = \text{Hash}(\text{IP-Adresse}) = K_x$
 - **Key-ID**: $K_y = \text{Hash}(\text{String})$, z.B. $\text{Hash}(\text{„Matrix.avi“}) = 107$
 - **Value**: Wert zu K_{107} zeigt auf IP-Adresse, Port des Speicherorts ($K_{107}, (51.12.3.2, 4711)$)

Beispiel: $m=7$



Chord – Consistent Hashing

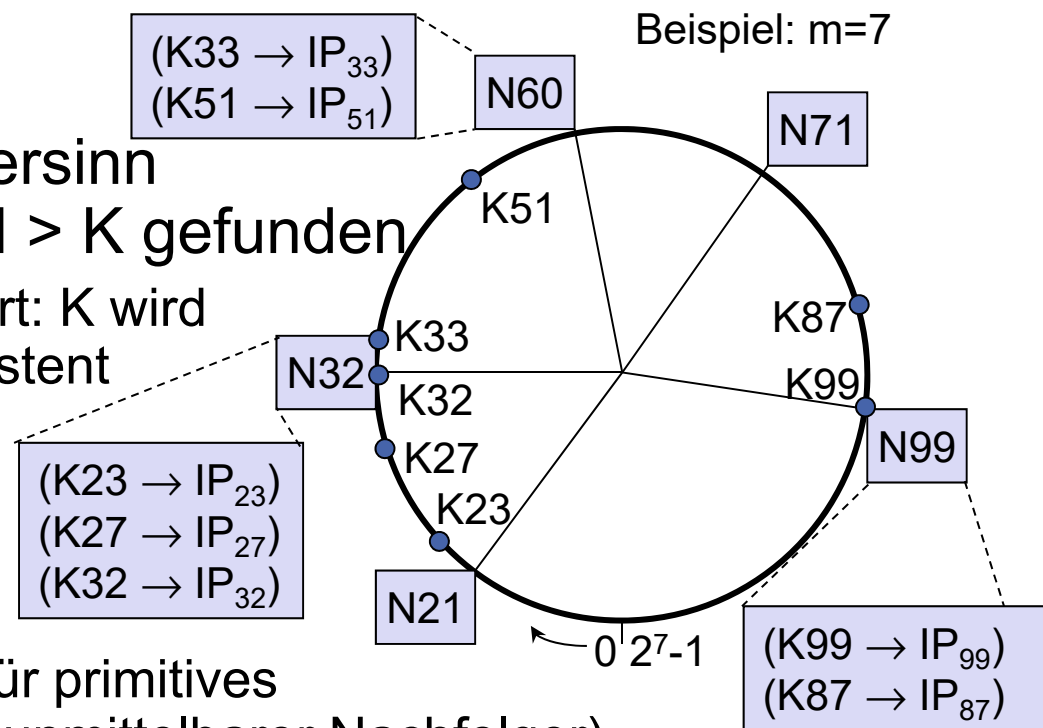
- Schlüsselzuordnung: Knoten verwalten Segmente des Wertebereichs $0 \dots 2^m - 1$
- Key-Value-Paare eines Segments werden im „Folge-Knoten“ (**Successor**) verwaltet
- Zum Beispiel K22...K32 werden in Knoten N32 verwaltet



Chord-Routing (1)

■ Vorgehen

1. Beginne Suche nach K bei beliebigem Knoten N
2. Verwaltet Knoten N gesuchten Schlüssel K , gib Tupel (K, V) zurück
3. Wenn nicht, leite Anfrage im Uhrzeigersinn weiter, bis Knoten $N > K$ gefunden
 - ▶ Zuverlässige Antwort: K wird gefunden, wenn existent
 - ▶ Effizientes Weiterleiten?
 - ▶ Es muss bekannt sein, welche Knoten existieren (für primitives Routing, zumindest unmittelbarer Nachfolger)



Routing in Chord (2)

■ Primitives Routing:

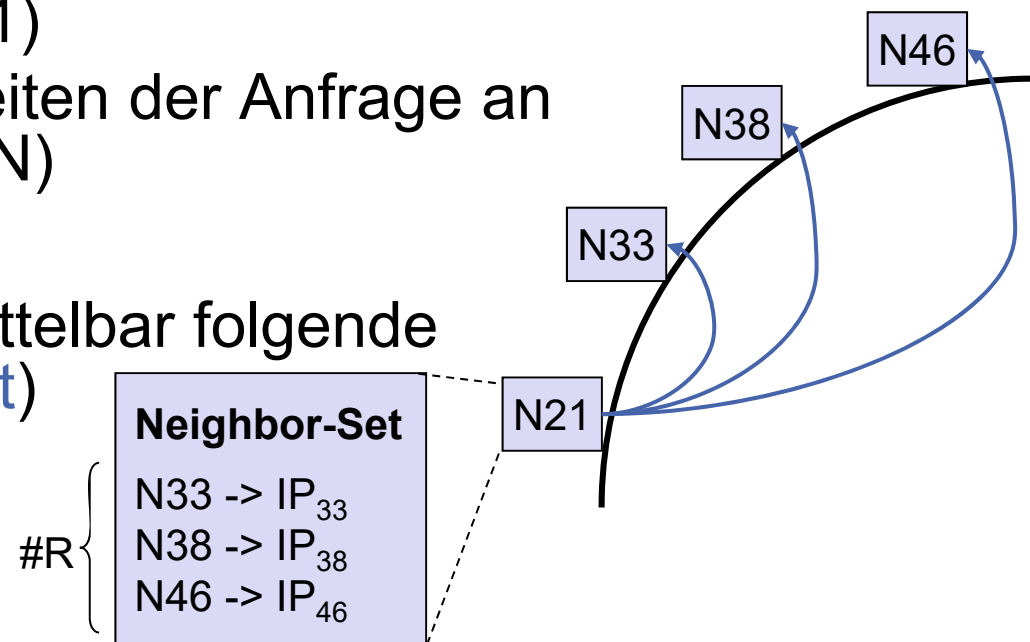
- Knoten muss Nachfolger (im Uhrzeigersinn) kennen
- Speicheraufwand $O(1)$
- Sukzessives Weiterleiten der Anfrage an nächsten Knoten: $O(N)$

■ Verbesserung:

- Knoten kennt R unmittelbar folgende Knoten (**Neighbor Set**)
- Aufwand $O(N/R)$

■ $R := N?$

- Suchaufwand: $O(1)$
- Aber: Speicheraufwand $O(N)$ und Probleme mit Aktualität der Tabelle

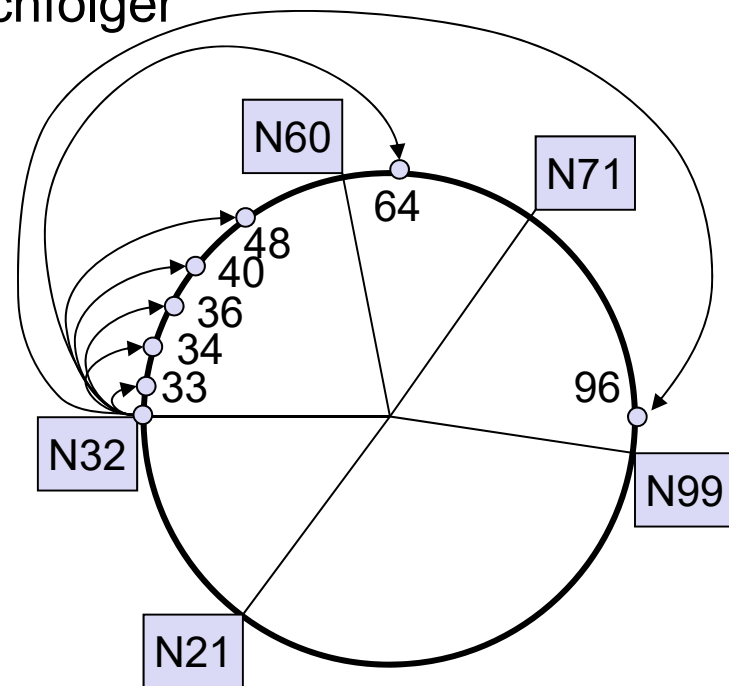


Routing in Chord (3)

■ Exponentielle Verzeigerung in Hash-Bereich

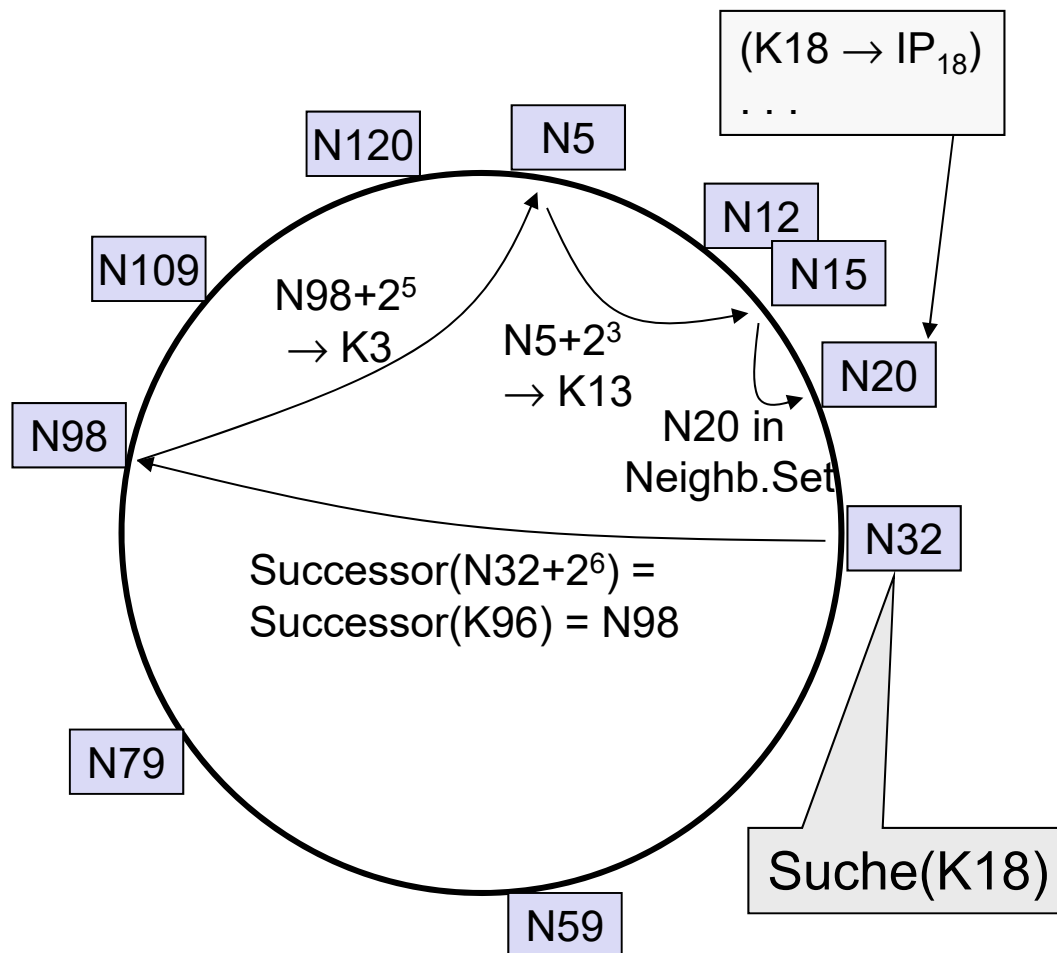
- Knoten verweist mit $m = \log N$ Zeigern in den Wertebereich
- Zeiger $\text{finger}[i]$ in Knoten n verweist auf Knoten, der Schlüssel $n + 2^{i-1}$ speichert, d.h. $\text{Successor}(n + 2^{i-1})$
 - $\text{finger}[1]$ ist immer der direkte Nachfolger
- Routing: Falls gesuchter Key k direktem Nachfolger zugeordnet werden kann, gib $\text{finger}[1]$ zurück, frage den Knoten von $\text{finger}[i] \in (n, k]$
 - Aufwand:
 - $O(\log N)$ Hops
 - $O(\log N)$ Verweise

Finger-Table		
i	Ziel	Zeiger
1	K33	N60
2	K34	N60
3	K36	N60
4	K40	N60
5	K48	N60
6	K64	N71
7	K96	N99



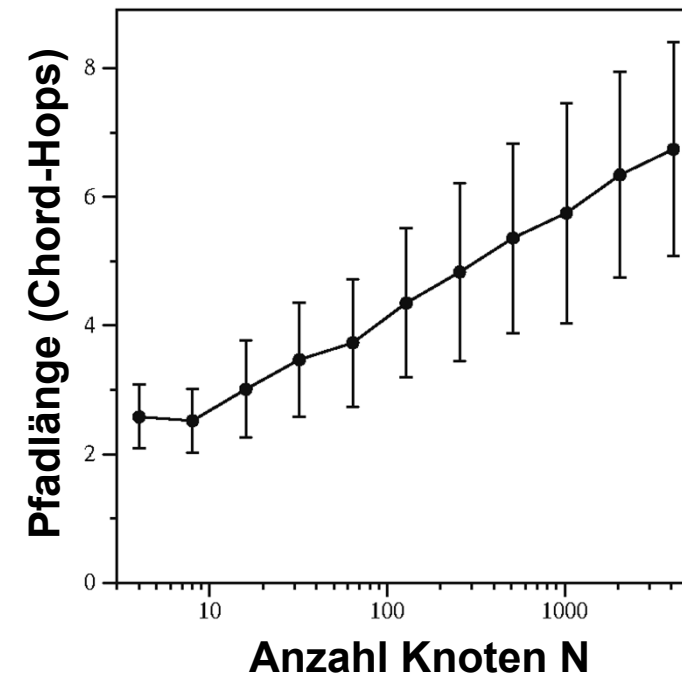
Beispiel: Routing in Chord – Beispiel / Pfadlänge

■ Suche nach K18 (m=7)



■ Verlauf der Pfadlänge je Netzgröße (Simulation)

■ $O(\log N)$



[10.5.1]

Knotenein-/austritte in Chord

- Chord muss Finger-Table aktuell halten
 - Dazu wird periodisch ein „Stabilisierungsprotokoll“ (**stabilize**) ausgeführt
 - Knoten n fragt seinen Nachfolger nach dessen Vorgänger p und entscheidet dann, ob nicht besser p sein Nachfolger wäre
 - das trifft zu, falls p neu hinzugekommen ist
 - Außerdem wird der Nachfolger benachrichtigt, damit dieser seinen Verweis auf seinen Vorgänger aktualisieren kann
- Außerdem wird noch **fix_fingers** ausgeführt
 - führt find_successor() für jeden Eintrag durch

Chord - Resümee

■ Komplexität

- Suchaufwand: $O(\log N)$
- Speicherbedarf pro Knoten: $O(\log N)$
- $\Omega(\log^2 N)$ Stabilisationsrunden zwischen N Node Joins

■ Vorteile

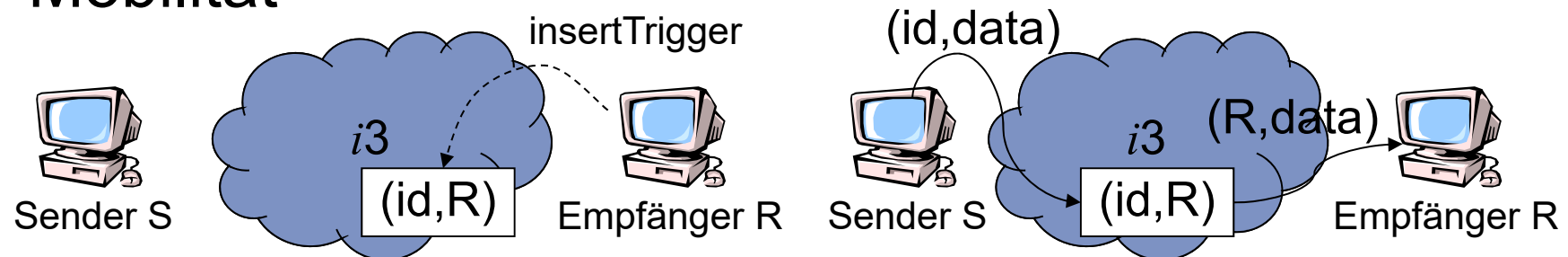
- Theoretische Validierung der Komplexität
- Auch im Fehlerfall logarithmischer Aufwand

■ Nachteile

- Keine explizite Suche nach nahen Knoten (No Proximity)
- Sicherheit noch unzureichend berücksichtigt
- Partitionierungsproblem nicht gelöst

Chord-Anwendung *i3*

- Internet Indirection Infrastructure *i3*  [10.5.7]
- Rendezvous-basiertes System
 - Pakete werden an logischen Identifier geschickt
 - Empfänger gibt Interesse am Paketempfang über *Trigger* bekannt
 - Sender schickt Paket an ID
 - *i3* verteilt Paket an Empfänger (ggf. mehrere)
 - Chord als Basis-Overlay
- Ermöglicht Dienste wie Multicast, Anycast, Mobilität



10.3.2 Kademlia

■ Rigide Chord Struktur

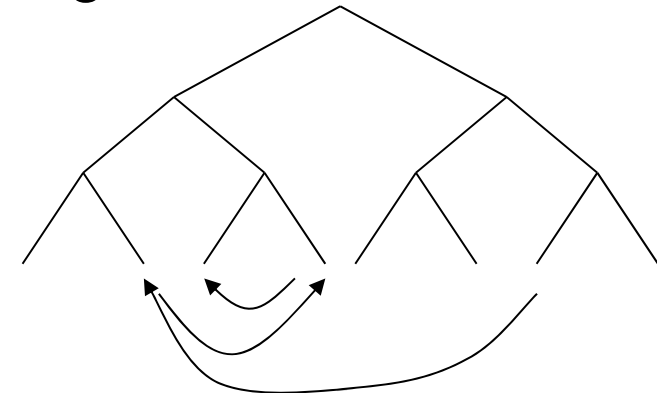
- erschwert Wiederherstellung der Konsistenz bei Knotenausfall
- Unidirektionalität der Struktur erlaubt keine aktive Aktualisierung der Routingtabellen durch Lookups
- Erlaubt kein Proximity-based Routing

■ Kademlia

- verwendet flexible Routingtabelle und unterstützt proximity-based Routing
- Reduzierter Verwaltungsaufwand, robuster gegen Änderungen
- Einträge der Routingtabelle werden durch Lookups aktualisiert
- jeder Knoten benötigt auch nur $O(\log N)$ „Kontakte“

Kademlia-Topologie

- Schlüssel: 160-bit gleichverteilt (Hashes)
- Fasse Wertebereich als **Binärbaum** auf
 - auch hier sollen weite Sprünge in die Nähe des Ziels führen, danach wird sich genauer angenähert
 - Anfrage kann an jeden Knoten im Unterbaum gestellt werden

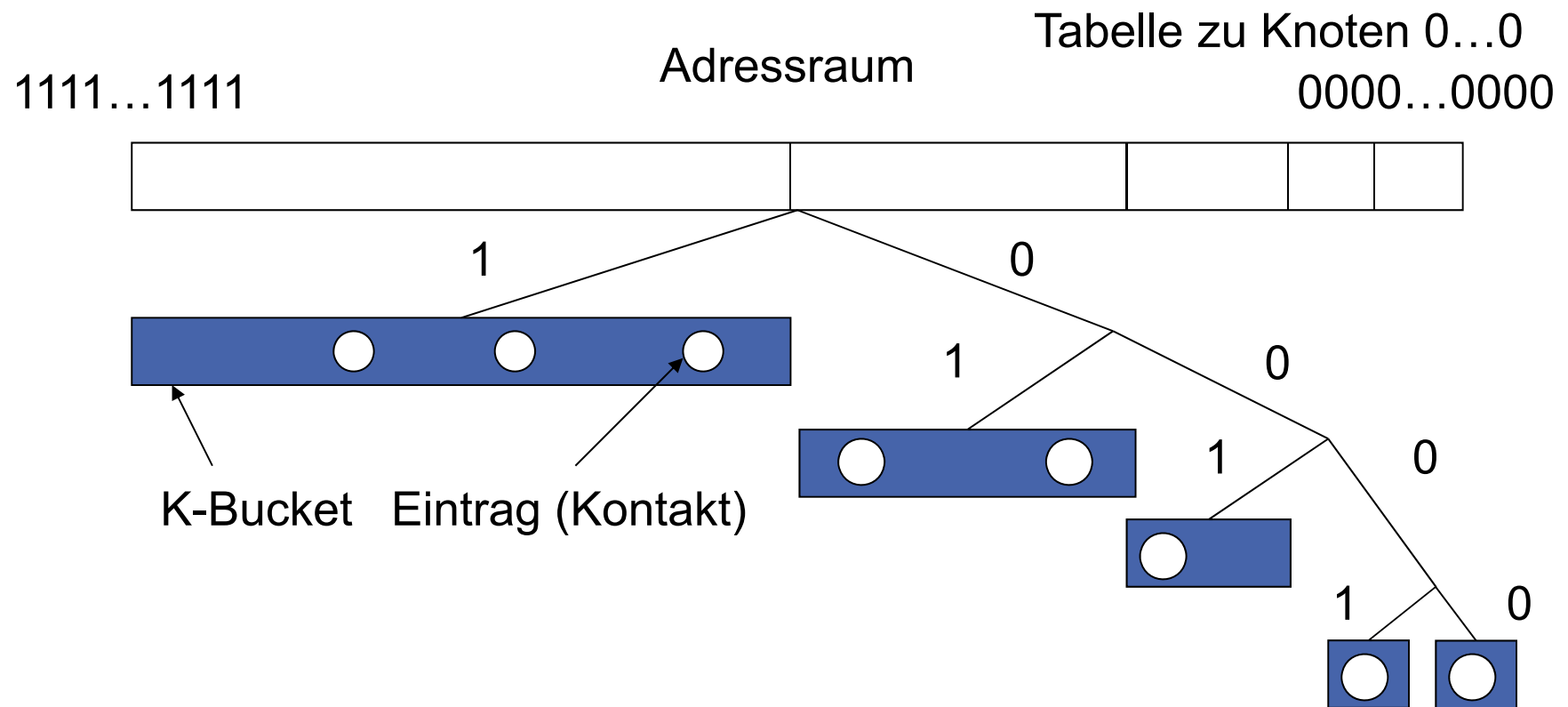


- **XOR-Topologie**: Metrik $d(X,Y) = X \oplus Y = d(Y,X)$
 - Unterschiede in den höherwertigen Bits fallen stärker ins Gewicht als Unterschiede in den niederwertigen Bits
 - Baum: Knoten im gleichen Unterbaum sind viel näher zusammen als Knoten in anderen Teilbäumen

Kademlia Datenstrukturen

- **Kontakt:** (Knoten-ID, IP+UDP Port)
- **k-Bucket**
 - Container für höchstens k Kontakte (z.B. k=20)
 - Kontakte werden nach LRU-Strategie eingetragen, u.a. Absender von empfangenen Anfragen
- **Routing-Tabelle**
 - Baum aus k-Buckets
 - Jeder **Bucket** ist **für** einen **Wertebereich** zuständig
- **Node Join**
 - Kopiert Kontakte eines bereits aktiven Knotens
 - Suche nach eigener ID, $O(\log N)$
- **Node Leave:** nichts zu tun

Routingtabelle K-Bucket Baum

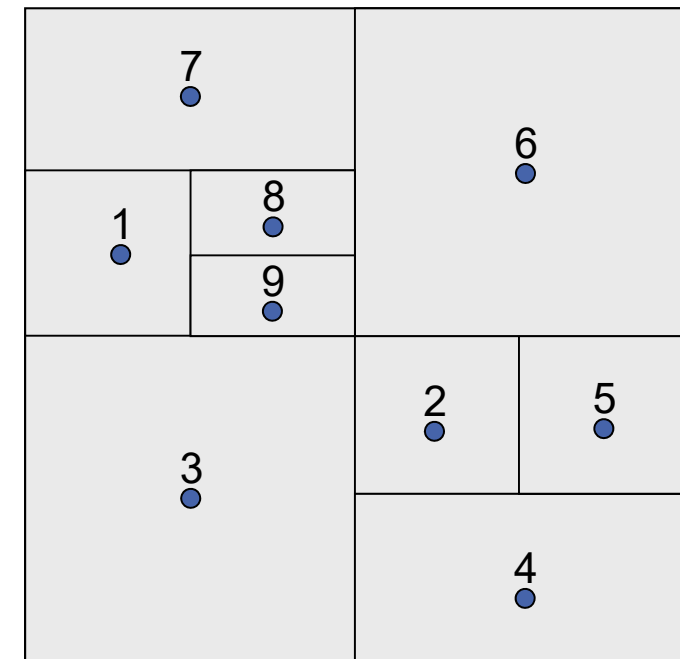


Kademlia Lookup

- Suche nach k Knoten, die dem Ziel T am nächsten sind
- $n.find_node(T)$ gibt alle Kontakte N zurück, die sich im ersten nicht leeren k -Bucket befinden, der sich am nächsten zum Ziel befindet
- n_0 (Anfragender Knoten),
 $N_1 = n_0.find_node(T)$,
 $N_2 = n_1.find_node(T)$, ... usw. so lange bis keine neuen Kontakte mehr zurückgeliefert werden
- n_i ist irgendein Kontakt aus N_i

10.3.3 CAN – Content Addressable Network

- Verteilte Hash-Tabelle mit Aufwand $O(\frac{D}{4} N^{\frac{1}{D}})$
- Hash-Wert entspricht Punkt in D-dimensionalem Raum
 - bspw. $H(\text{„Matrix.avi“}) \rightarrow (0.7, 0.2)$
 - DHT verwaltet (key, value)
- Ein Overlay-Knoten verwaltet eine Partition des Raums
 - Knoten 4 verwaltet alle Hash-Werte im Bereich $x \in [0.5, 1], y \in [0, 0.25]$
 - Angrenzende Bereiche heißen „Nachbarn“
 - 6, 2, 4, 3 Nachbarn von 5 (1 nicht!)
 - „Wrap around“ an DHT-Grenzen
 - Erwartete Anzahl Nachbarn: $O(2 \cdot D)$

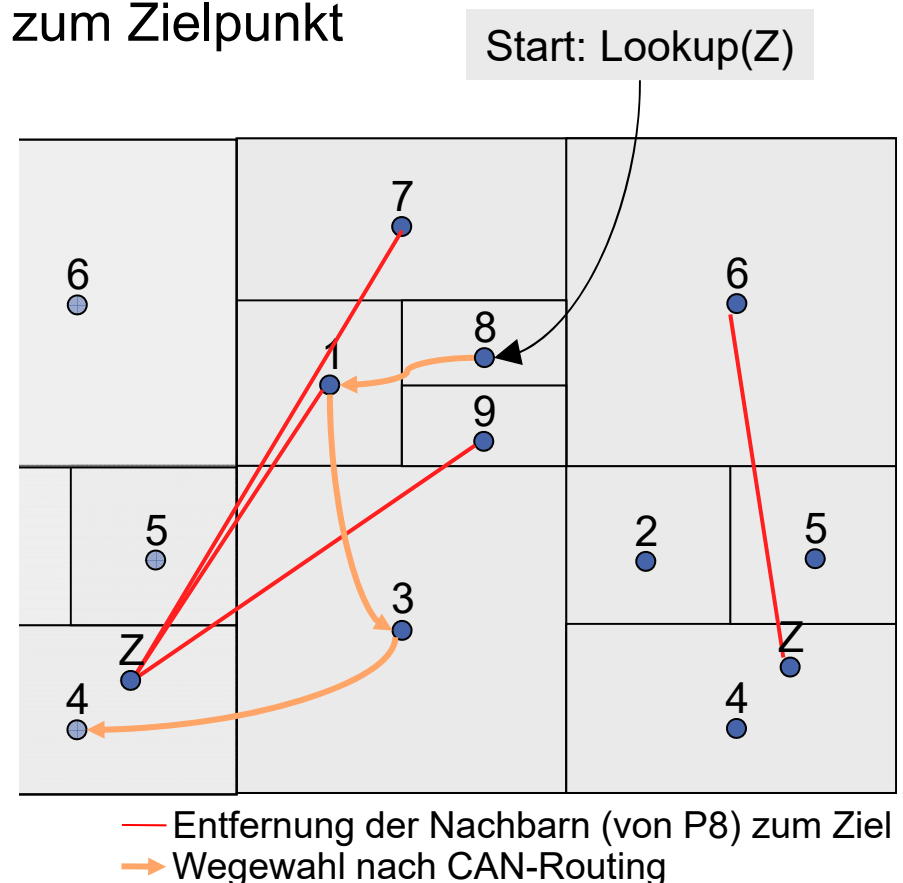


Routing in CAN

- Wie komme ich von P8 nach Z?
 - Route entlang des kürzesten Pfades im D-dimensionalen Raum
 - Genauer:
 - Nachbar mit kürzestem Abstand zum Zielpunkt ist nächster Hop!
 - Abstand zum Ziel sinkt kontinuierlich
 - Aufwand: $O(\frac{D}{4} N^{\frac{1}{D}})$ Hops in der DHT

■ Beispiel:

- P8: Lookup(Z)
- Welcher Nachbar von P8 ist am nächsten zu Z?
 - Route zu P1
- Im nächsten Schritt zu P3
- Und dann zu Z

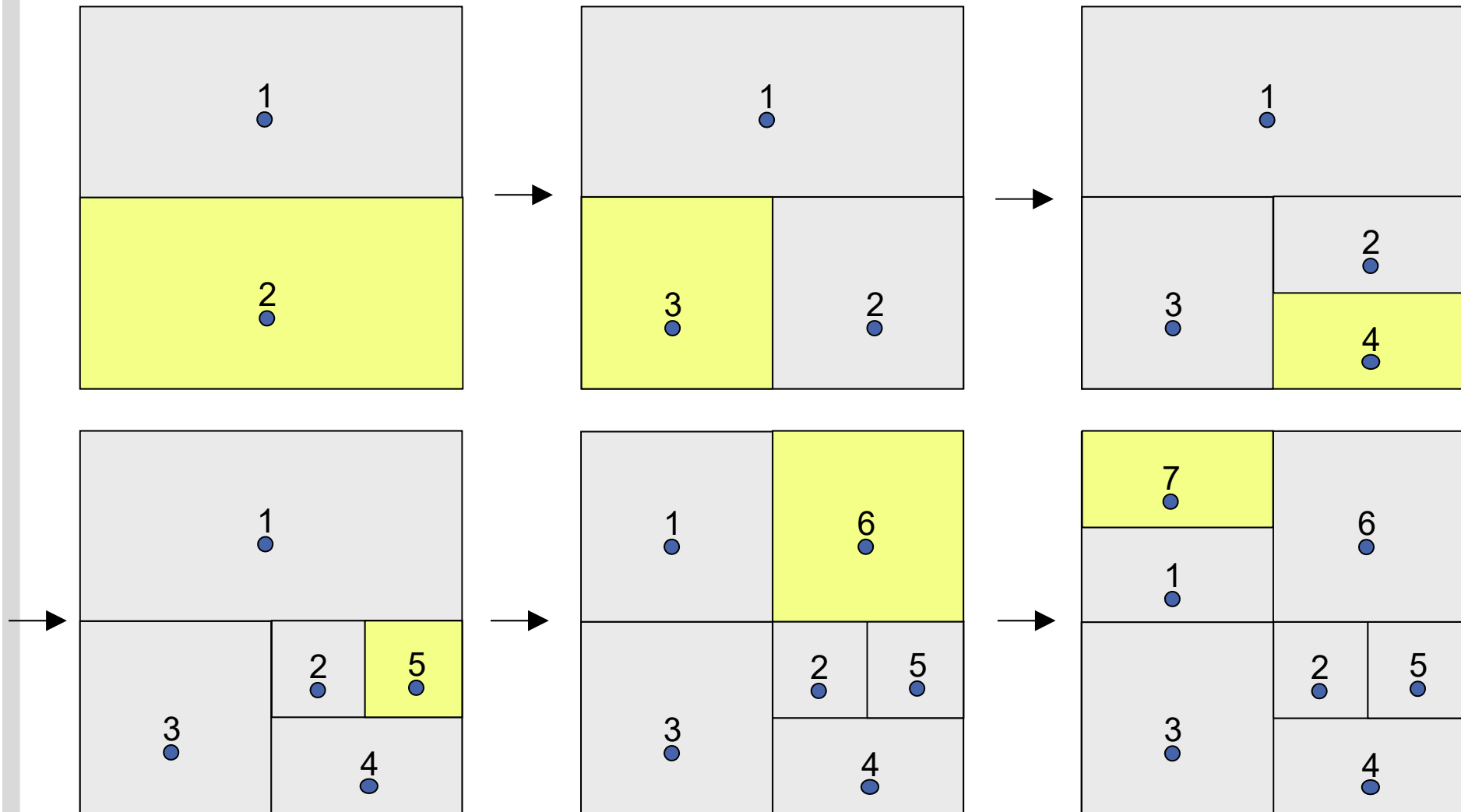


Aufbau eines CAN (1)

- Einfügen eines Knotens X
 - Suche einen CAN-Knoten als Einstieg in DHT
 - Wähle einen zufälligen Punkt im D-dimensionalen Raum
 - Route JOIN-Nachricht zu diesem Punkt
bzw. zu Knoten K, der diesen Punkt verwaltet
 - Teile die Region dieses Knotens K in zwei Hälften
 - Dimension zur Teilung wird gemäß Ordnung ausgewählt
 - Bspw. x, y, z, x, y, z, ... bei D=3
 - Verwaltete (Key, Value)-Paare der neuen Region von X werden von Knoten K an X übergeben
 - Neuer Knoten „erbt“ seine Nachbarn vom alten Knoten
 - Nachbarn frischen ihre Nachbar-Informationen auf
- Gesamter Aufwand: $O(D)$ – unabhängig von N (!)

Aufbau eines CAN (2)

■ Beispiel: Einfügen von P2, ..., P7



Entfernen eines Knotens aus CAN

■ Verlassen eines Knotens K aus dem CAN

- Region und verwaltete (Key,Value)-Paare gehen an Nachbar über
 - Idealfall: Regionen können gemäß Teilung verschmelzen
 - Falls nicht: verwaltet kleinster Nachbar (nach Anzahl der Schlüssel) beide Regionen (kein Verschmelzen!)
- Bei Abmelden eines Knotens: ordentliche Übergabe
- Bei Ausfall eines Knotens: TAKEOVER-Prozedur
 - Ausbleiben der periodischen Update-Infos bei Nachbarn
 - Relativ zur Größe der Region starten Nachbar Timer (vgl. ARP)
 - (Kleinsten) Nachbar signalisiert TAKEOVER den anderen Nachbarn und übernimmt die Region
- Restrukturierung im Hintergrund

Verbesserungen zur Leistungssteigerung bei CAN

■ Komplexität von CAN:

- Zustandsinformationen pro Knoten :
 $O(D)$ (unabhängig von N)!
- Routing: $O(DN^{\frac{1}{D}})$ Hops (im Overlay!) (Tendenz linear!)
 - Problem: Overlay-Hop $\neq$ Hop in IP-Topologie
 - Ziel: Ähnliche Verzögerung zw. Hops im Overlay wie im IP-Netz

■ Verbesserungen:

- Mehr Dimensionen:
 - Erhöhte Anzahl der Nachbarn D
- Mehrere Koordinatensysteme (Realities)
 - Mehrere DHTs gleichzeitig
 - Punkt (x,y,z) wird auf mehreren Knoten gespeichert → Redundanz

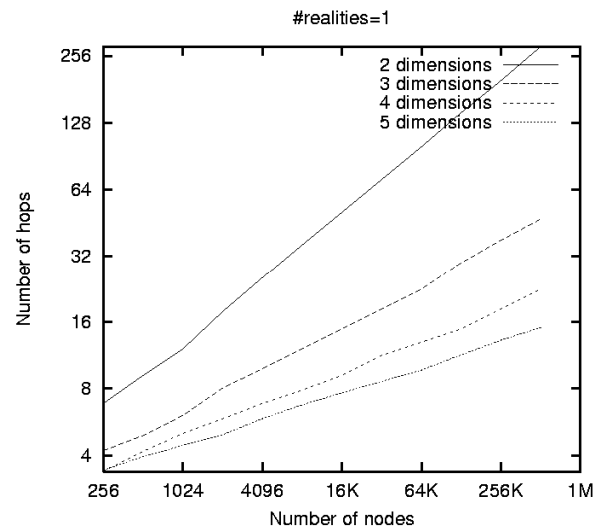
Mehr Dimensionen ↔ Parallele Koordinatensysteme

■ Mehr Dimensionen

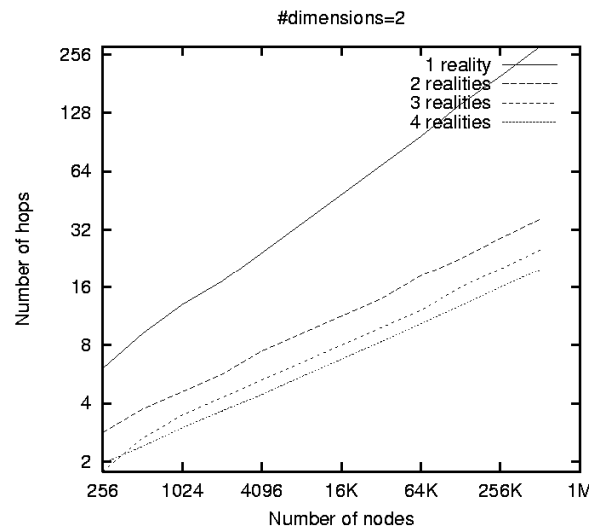
- Mehr Nachbarn
- Mehr Routingmöglichkeiten
- Mehr Statusinformationen $O(2D)$

■ Mehrere Koordinatensysteme (r)

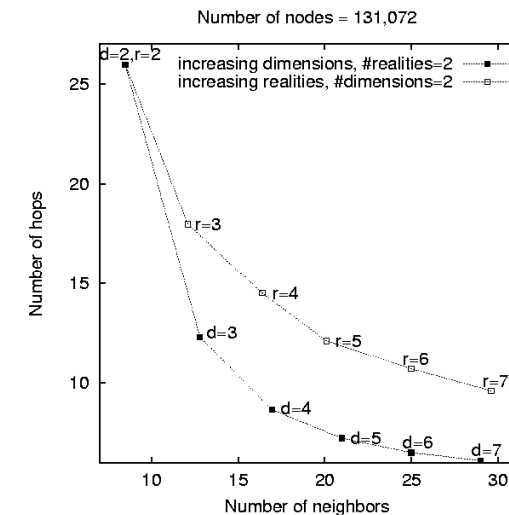
- r Möglichkeiten für Routing
- Statusinformationen $O(r \cdot D)$
- r-fache Redundanz!



Steigerung der Dimensionen



Steigerung der Realities



Vergleich

- Fazit: Mehr Dimensionen führt zu kürzeren Pfaden im Overlay
(...aber mehr Koordinatensysteme steigern die Redundanz)



[10.6.1]

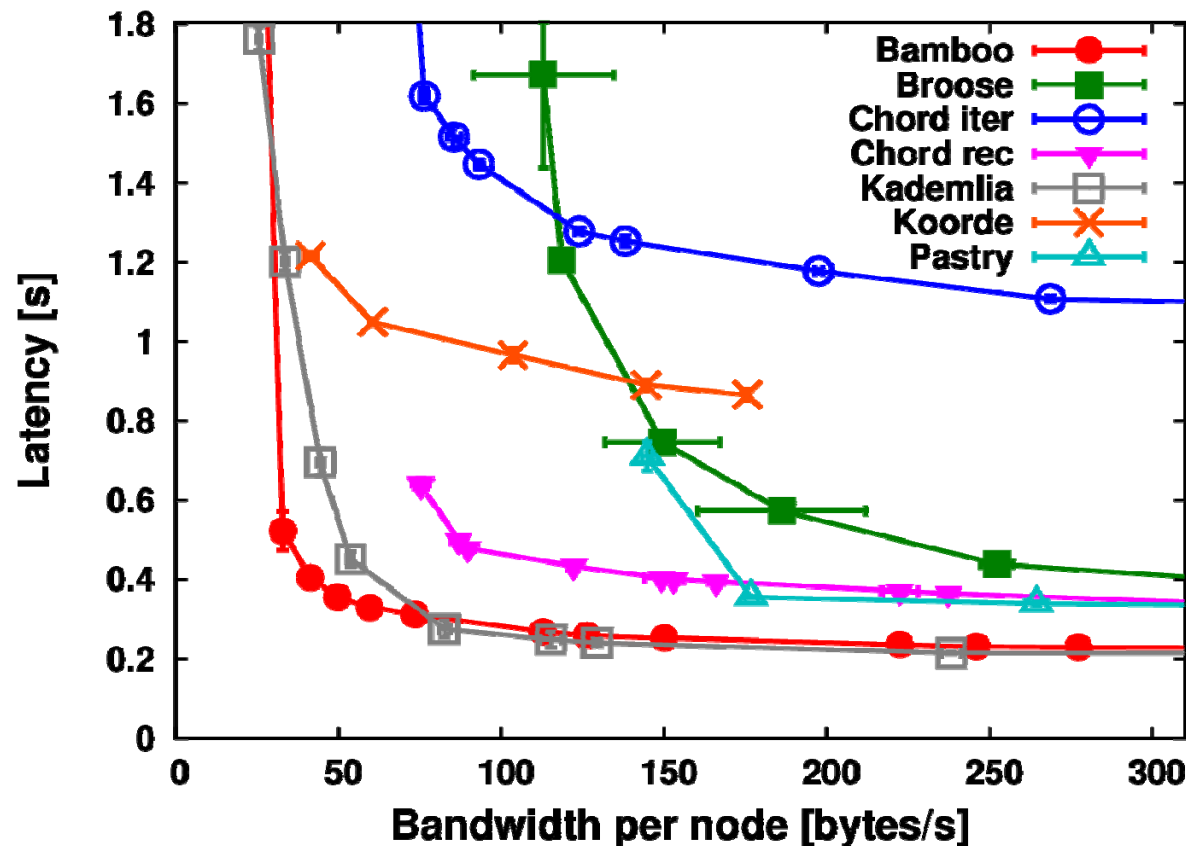
Weitere Verbesserungen von CAN

- Bessere Routing Metriken
 - Messen der Laufzeit zu Nachbarn
 - Auswahl des Nachbarn mit dem größten Fortschritt/Laufzeit
- Überlappende Regionen
 - Mehr Redundanz
 - Schnellere Routen, weil Anzahl der Zonen sinkt
- Mehrere Hash-Funktionen
 - Redundanz
- Gleichmäßigere Teilung von Regionen
 - Zielzone prüft beim Einfügen, ob nicht ein Nachbar größer ist und sich besser zur Teilung eignet
- Berücksichtigung der Topologie bei CAN-Wachstum

Komplexität strukturierter Overlays

	CAN	Chord	Pastry	Tapestry
Zustände pro Knoten	$O(D)$	$O(\log N)$	$O(\log N)$	$O(\log N)$
$O(\frac{D}{4} N^{\frac{1}{D}})$ Pfadlänge (Routing)		$O(\log N)$	$O(\log N)$	$O(\log N)$
Knoten hinzufügen	$O(DN^{\frac{1}{D}})$	$O(\log^2 N)$	$O(\log N)$	$O(\log N)$
Knoten entfernen	?	$O(\log^2 N)$	?	?

Vergleich strukturierter Overlays im Simulator OverSim



- Bamboo und Kademlia mit geringer Latenz und Kommunikationsaufwand
- Topologie oft nicht so wichtig (z.B. Timeout-Behandlung oft kritischer!)

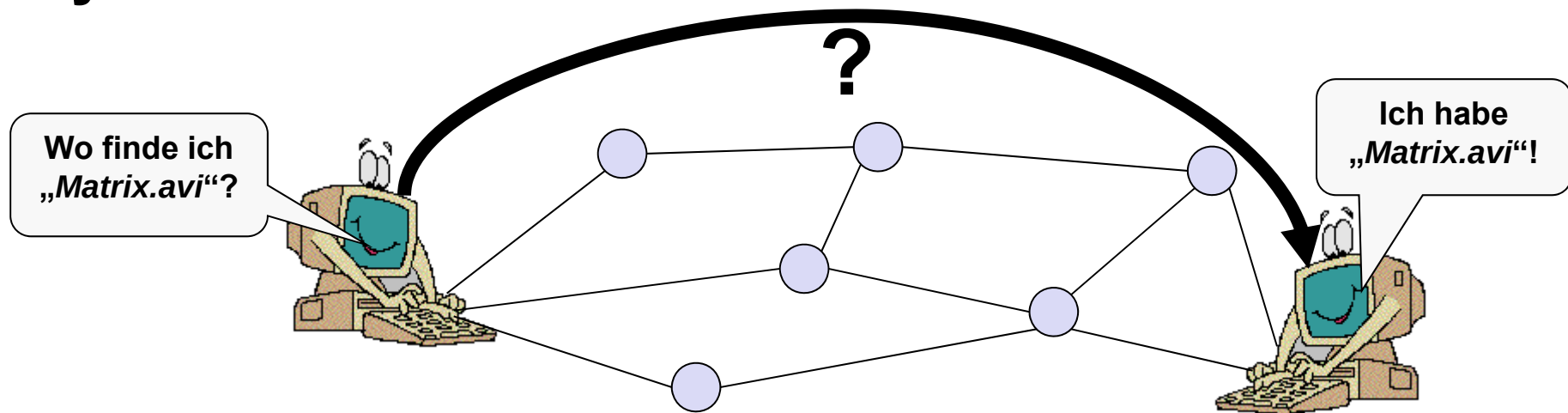
[1] I. Baumgart, B. Heep, Fast but economical: A simulative comparison of structured peer-to-peer systems, Proceedings of the 8th Euro-NF Conference on Next Generation Internet (NGI 2012), Karlskrona, Sweden, Juni 2012

10.4 Verteilte Hash-Tabellen

- Was sind verteilte Hash-Tabellen (Distributed Hash Tables – DHTs)?
 - Einsatzgebiete? – Wofür DHTs?
 - Komplexität? – Wie effizient/robust sind DHTs?

- Geeignete Overlay-Netze
 - Pastry – Rice, Microsoft  [10.4.1]
 - Chord – UC Berkeley, MIT  [10.5.1]
 - Kademlia – U of NY  [MaMa02]
 - CAN – UC Berkeley, ICSI/ICIR  [10.6.1]
 - (Tapestry – UC Berkeley)  [10.7.1]

Informationssuche in Peer-to-Peer-Systemen



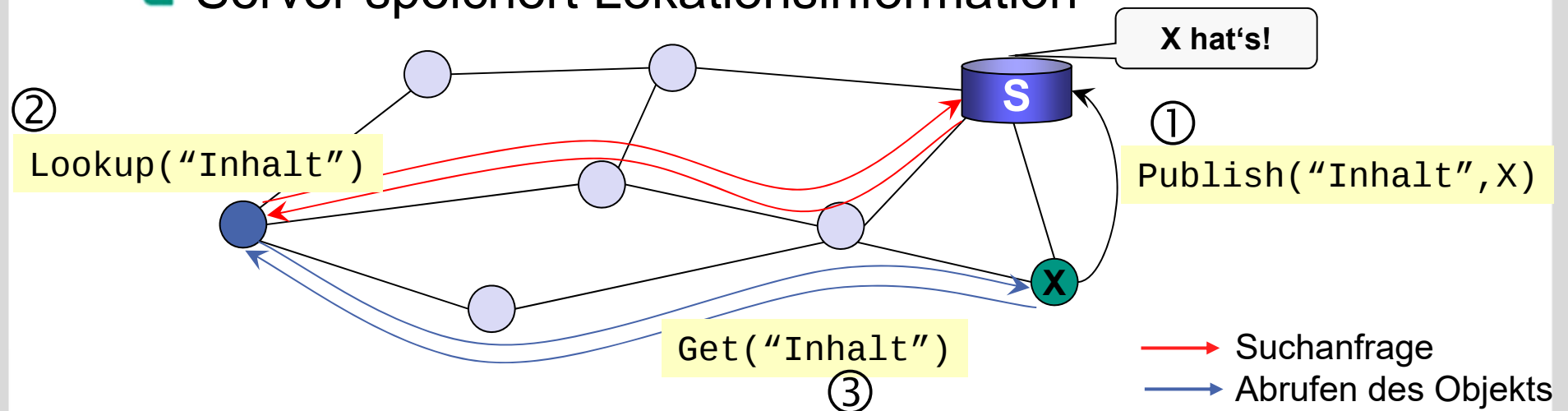
Wesentliches Problem in allen Peer-to-Peer-Systemen?

- Auffinden eines Datums im verteilten System
 - Wo soll Datum gespeichert werden?
 - Publish("Inhalt", ...)
 - Wie findet Anfrage den Speicherort?
 - Lookup("Inhalt")
- Geringer Aufwand: Kommunikation, Speicher
- Robust gegen Ausfälle und häufige Änderungen

Auffinden von Inhalten: zentraler Server

■ Einfachste Strategie: Client-Server

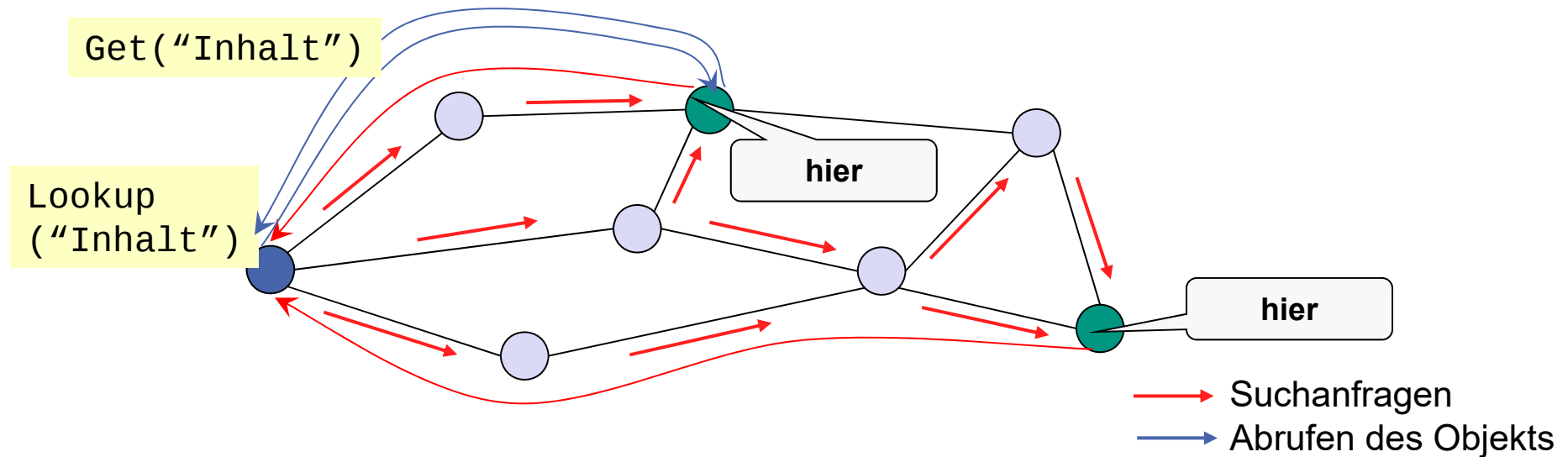
- Server speichert Lokationsinformation



■ Alle bereits genannten Probleme:

- Skalierbarkeit ($O(N)$ Speicheraufwand), Aktualität der Daten, Single-Point-of Failure, Kosten der Realisierung, etc.
- Trotzdem:
 - Bestes Prinzip für einfache Anwendungen

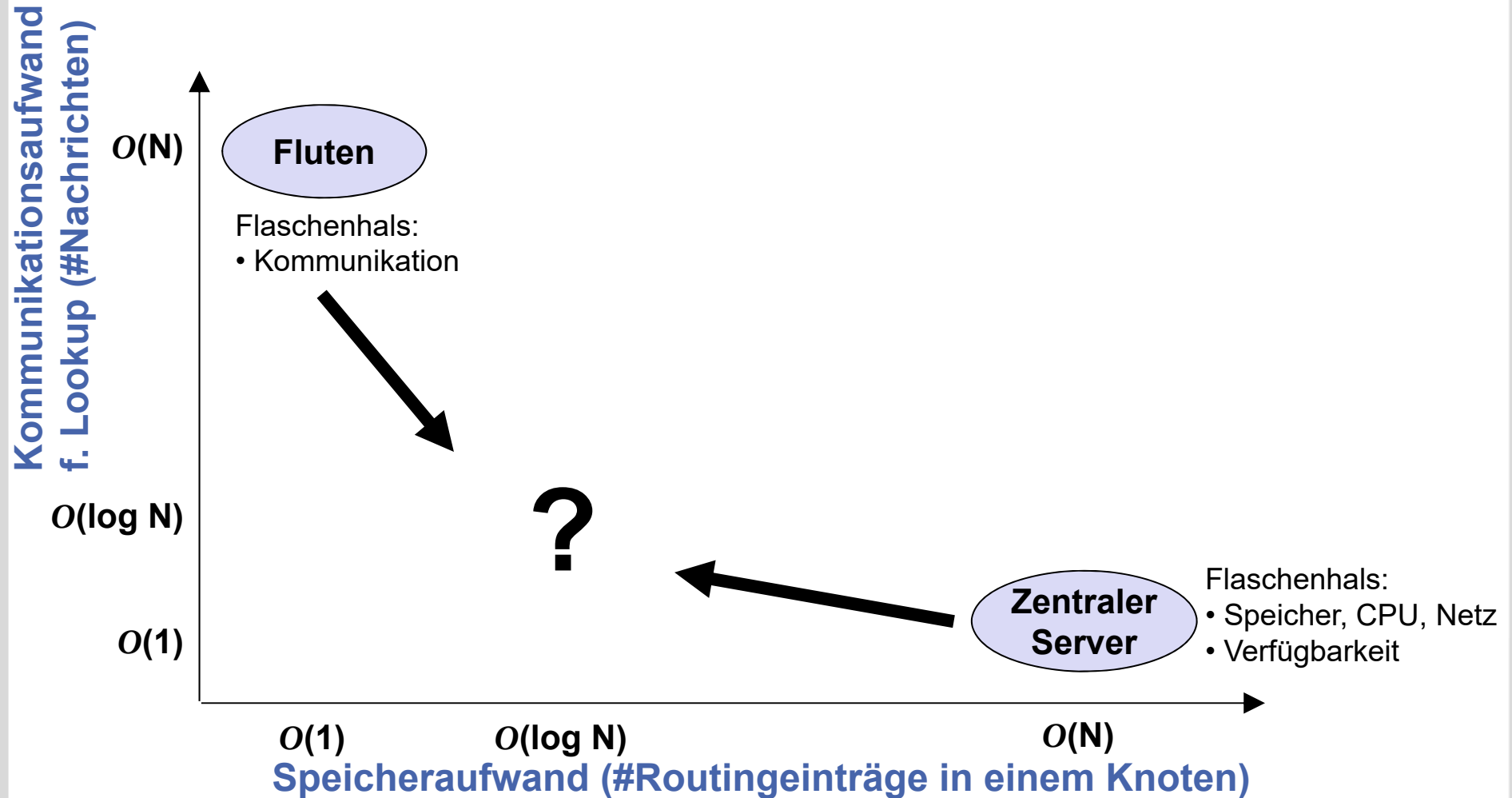
Auffinden von Inhalten: Breitensuche/Fluten



■ Breitensuche (Fluten, wie bei Gnutella)

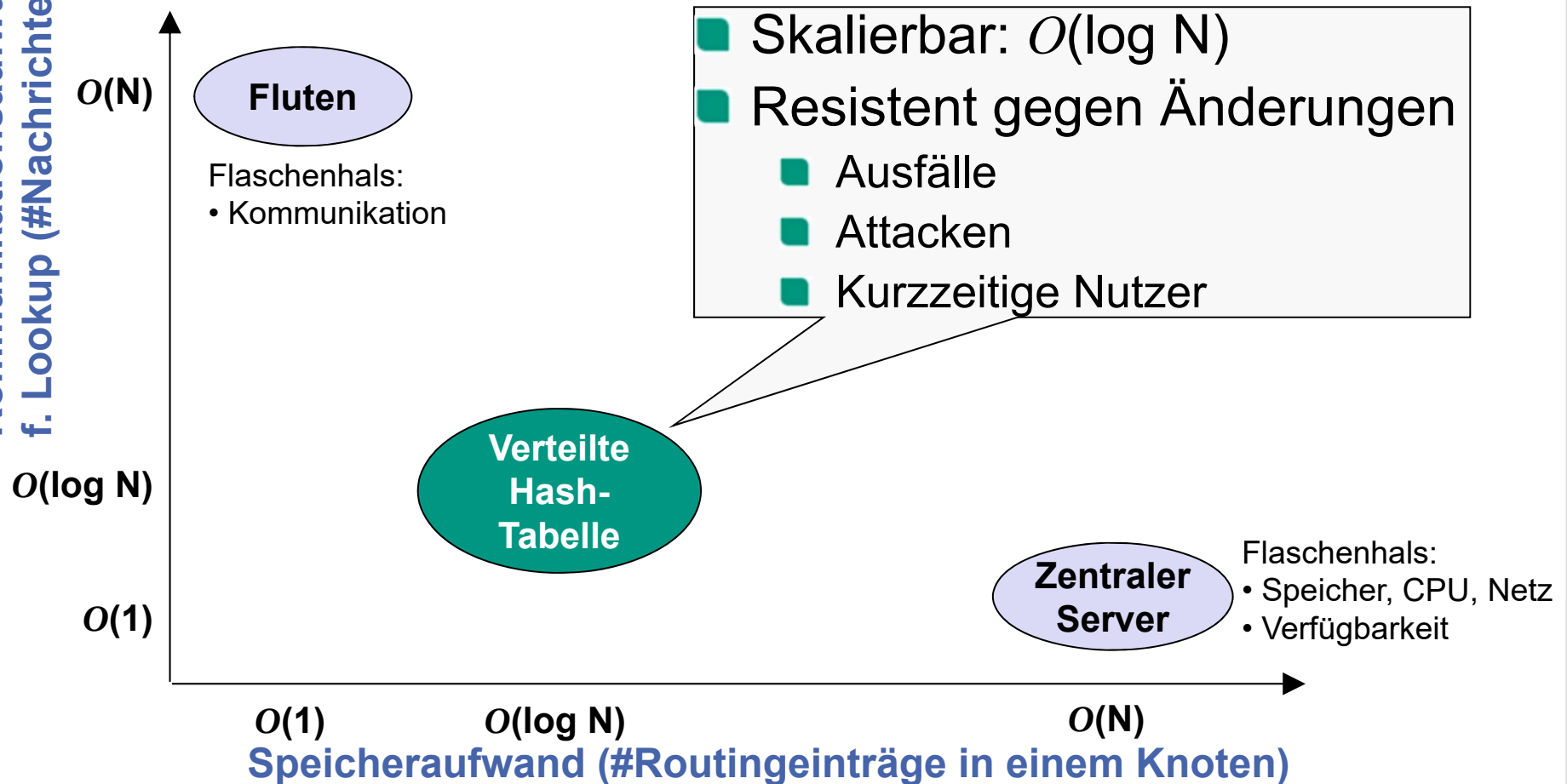
- Hohe Anzahl an Nachrichten
- Skaliert nicht
 - Netzbelastung
 - Kommunikationsaufwand

Auffinden v. Inhalten – Such- vs. Speicheraufwand



Auffinden v. Inhalten - Such- vs. Speicheraufwand

Kommunikationsaufwand
f. Lookup (#Nachrichten)



Prinzip verteilter Hash-Tabellen

■ Idee verteilter Hash-Tabellen

- Verteile Daten (Inhalte) über alle Knoten
 - bzw. Information über tatsächl. Lokation eines Inhalts (Indirektion)
- Bei Anforderung eines Datums frage Knoten, der Inhalt (Info) speichert

■ Herausforderungen:

- Gleichmäßige Verteilung der Inhalte auf Knotenmenge
 - Effizientes Auffinden von Inhalten
- Ständige Anpassung durch Ausfall, Beitritt und Austritt von Knoten → Selbstorganisation
 - Vergabe von Zuständigkeiten an neue Knoten
 - Übernahme und Neuverteilung von Zuständigkeiten bei Ausfall, Austritt

Prinzip verteilter Hash-Tabellen – Abbildung

■ Schritt 1:

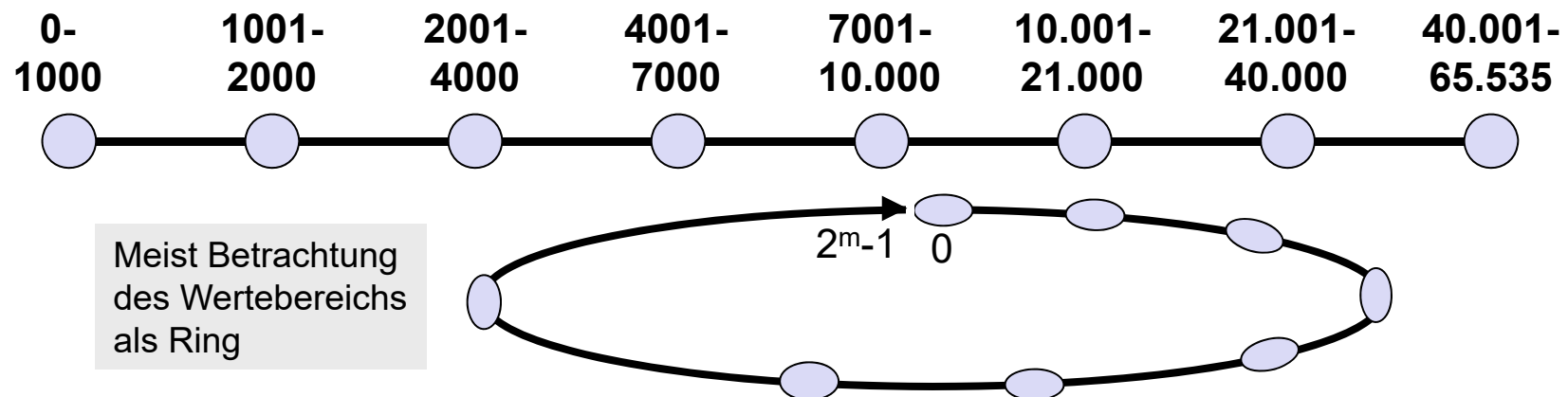
■ Abbildung des Inhalts auf linearen Wertebereich

- Inhalt wird auf **Identifikator** (ID bzw. Schlüssel) aus diesem Wertebereich abgebildet, meist:

$ID \in [0, \dots, 2^m - 1] \gg N_{\max}$
($N_{\max}$: Anzahl der maximal zu speichernden Objekte)

- Meistens Abbildung des Inhalts in Wertebereich durch Hashfunktion H , z.B. $H(\text{Zeichenkette}) \bmod 2^m$:
 $H(\text{„/movie/Matrix/divx/en“}) \rightarrow 2313$

■ Verteilung des Wertebereichs über DHT-Knoten



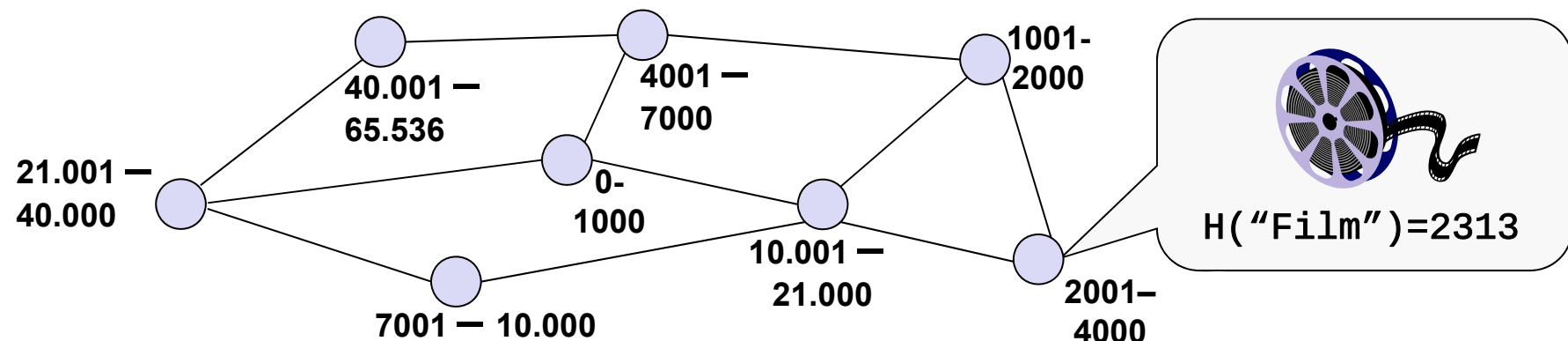
Speicherung der Inhalte in DHTs (1)

Wie werden „Inhalte“ in Knoten gespeichert?

- Kann von DHT-Basisfunktionalität getrennt werden
- Annahme: $H(\text{„Inhalt“})$ ist Abbildung in Wertebereich
- Abbildung Wertebereich $\rightarrow$ Knoten notwendig (meist $H(\text{Underlay-ID}) \rightarrow \text{ID}$), z.B. Underlay-ID=(IP-Adresse, Port)

■ Direkt:

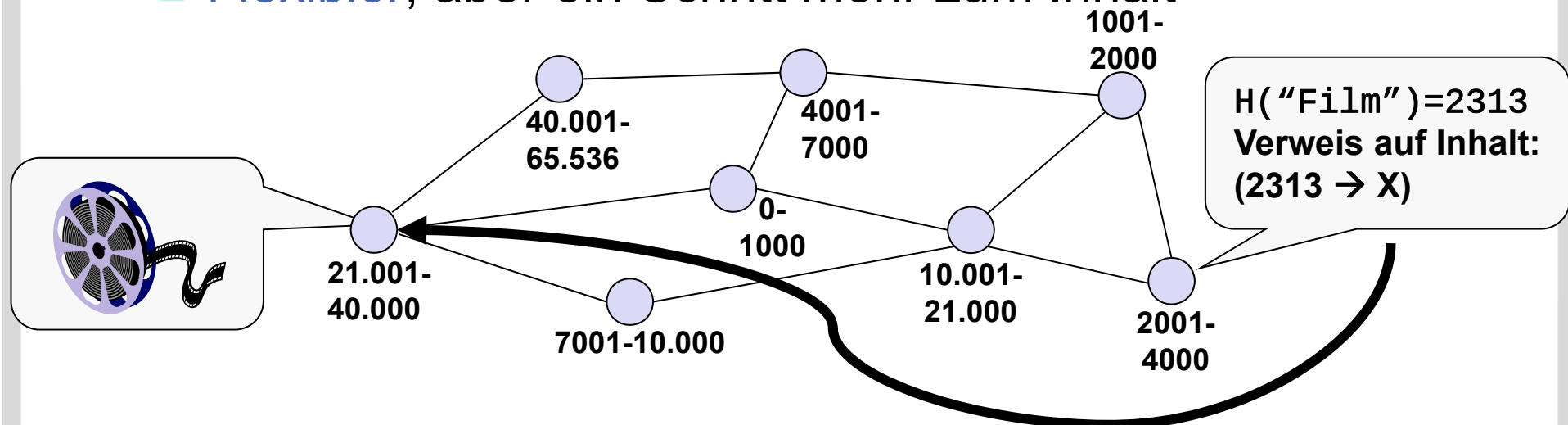
- Inhalt wird in Knoten $H(\text{„Inhalt“})$ gespeichert
 $\rightarrow$ **unflexibel** für große Inhalte



Speicherung der Inhalte in DHTs (2)

■ Indirekt:

- Knoten in DHT speichern Tupel (Key, Value)
 - Key = Hash(„/movie/Matrix/divx/en“) → 2313
 - Value: meist reale Adresse, an der Inhalt gespeichert wird:
(IP, Port) = (129.13.15.3, 4711)
- Flexibler, aber ein Schritt mehr zum Inhalt



Verteilte Hash-Tabellen – Auffinden des Inhalts

■ Schritt 2:

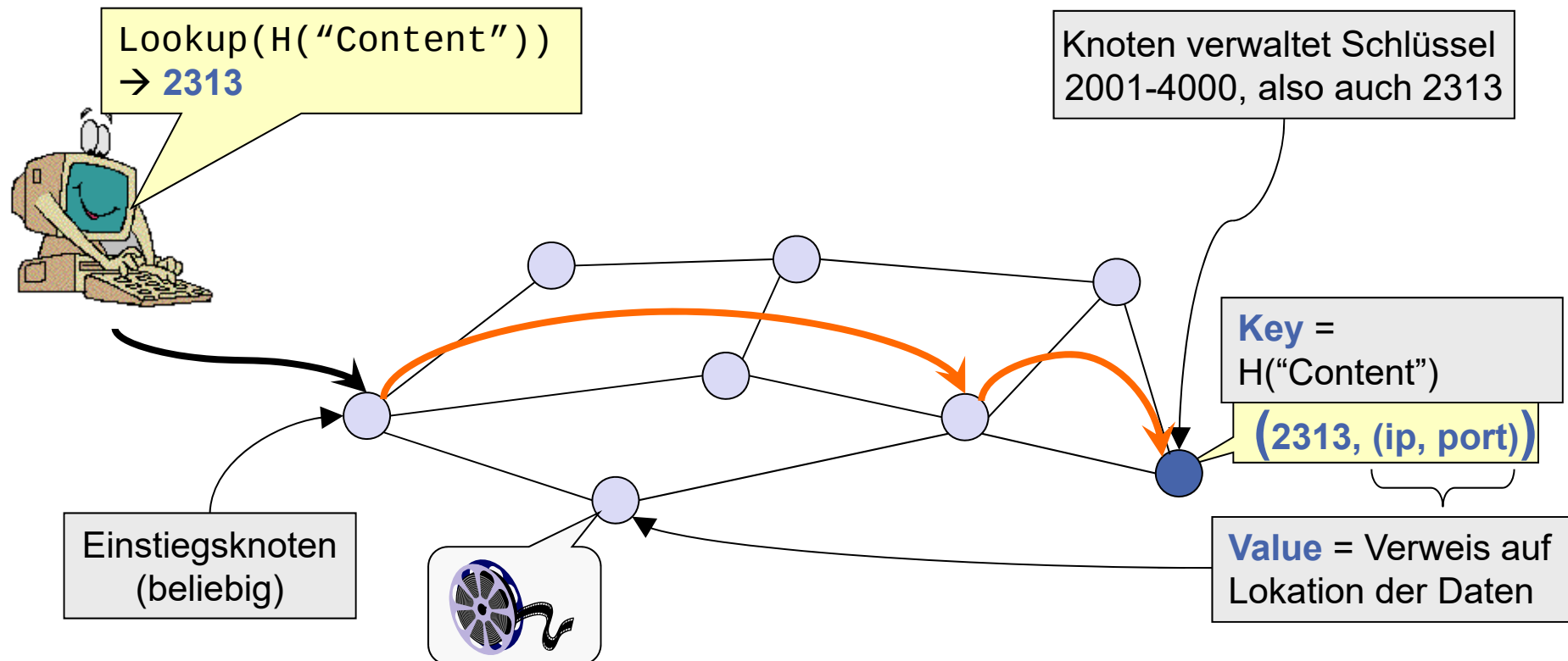
Auffinden des Inhalts (Inhaltsbasierte Suche – Content based Routing)

- Achtung: keine Suche wie z.B. Finde alle Daten, die bestimmtes Stichwort enthalten
- Ziel: geringer, skalierbarer Aufwand
 - $O(1)$ bei zentraler Hash-Tabelle
 - Aber: Verwaltung zentraler Hash-Tabelle zu umfangreich (→Server!)
 - Angestrebter Aufwand bei verteilter Hash-Tabelle
 - $O(\log N)$: Hops zum Auffinden eines Objekts
 - $O(\log N)$: Anzahl der Schlüssel und Routing-Infos pro Knoten

Auffinden des Inhalts (2)

■ Suche nach Schlüssel

- Einstieg bei beliebigem Knoten
- Routing zu gesuchtem Inhalt (Key)



DHT-Aktualisierung: Bei- und Austritt von Knoten

■ Beitritt eines neuen Knotens

- Zuteilung eines bestimmten zu verwaltenden Hash-Bereichs
- Kopieren der (Key,Value)-Tupel des Hash-Bereichs (meist mit Redundanz)
- Einbindung in Routing-Strukturen

■ Austritt eines Knotens

- Aufteilung des verwalteten Hash-Bereichs auf “Nachbar”-Knoten
- Kopieren der (Key,Value)-Tupel auf zuständige Knoten
- Ausgliederung aus Routing-Strukturen

■ Ausfall eines Knotens

- Nutzung redundanter (Key,Value)-Tupel (bei Ausfall des Tupel-Knotens)
- Nutzung redundanter/alternativer Routing-Wege
- Key-Value meist erreichbar, wenn mindestens eine Kopie erreichbar

Single Hop DHT?

Ansatz:

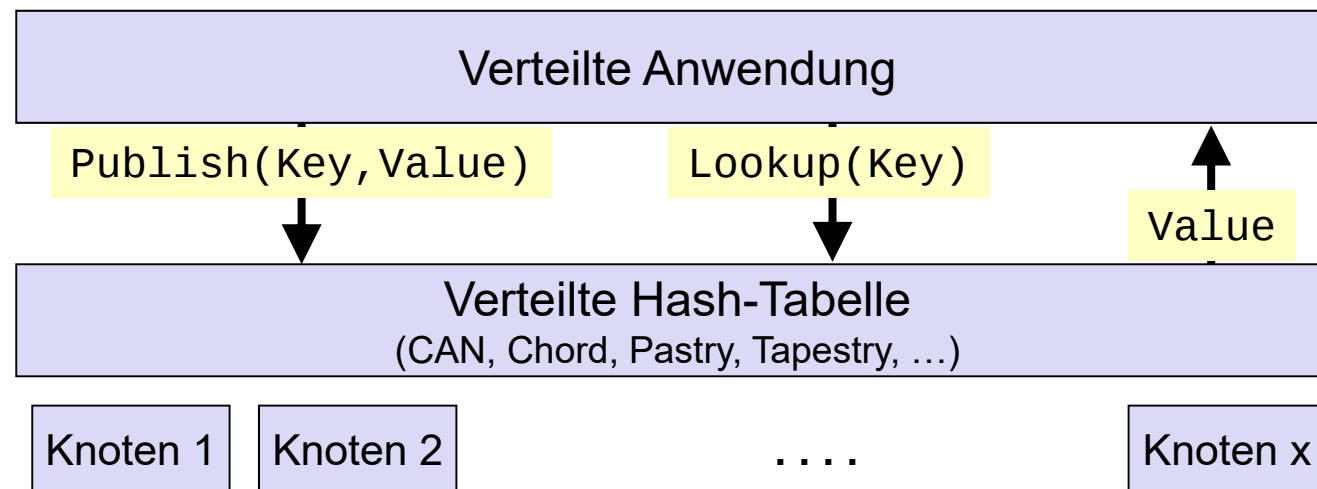
- Jeder Knoten besitzt vollständige Routingtabelle
 - Auffinden von Knoten in einem Schritt
→ nur eine Nachricht, geringe Latenz
 - hoher Speicheraufwand (praktisch kaum ein Problem) für jeden Knoten ein Eintrag
 - hoher Wartungsaufwand:
 - jeder Knoten muss über Änderungen informiert werden
 - hoher Bandbreitenbedarf
 - Stabilität?
- D1HT  [MoAm06]
 - 99% der Lookups in einem Hop
 - intelligenter Broadcast: jeder Peer schickt höchstens $\log_2(n)$ Nachrichten in einem Status-Update-Intervall
 - Kontrollverkehr braucht bis zu 20kbit/s in System mit $n=10^6$ Knoten (mittlere Verweilzeit eines Peers 60min)

Schnittstelle DHT

■ Generische Schnittstelle von DHTs

- Einstellen der Information (z.B. Lokation eines Inhalts)
 - `Publish(Key, Value)`
- Abrufen der Information (Suche nach Inhalt)
 - `Lookup(Key)`
- Antwort:
 - Value

■ DHT-Ansätze austauschbar (bzgl. Schnittstelle)



Resümee: Verteilte Hash-Tabellen (1)

■ Resümee: Eigenschaften DHTs

- Schlüssel werden **gleichmäßig** über Knoten der DHT **verteilt** (oft mit Redundanz)
 - Keine Flaschenhälse
 - Erlauben stetiges Wachstum der Anzahl gespeicherter Schlüssel
 - Tolerieren Ausfälle von Knoten
 - Überleben gezielte Attacken
- **Selbstorganisierend**
- Einfache und günstige Umsetzung
- Unterstützen breites Feld von Anwendungen
 - Schlüssel hat keine semantische Bedeutung
 - Wert ist anwendungsabhängig

Resümee: Verteilte Hash-Tabellen (2)

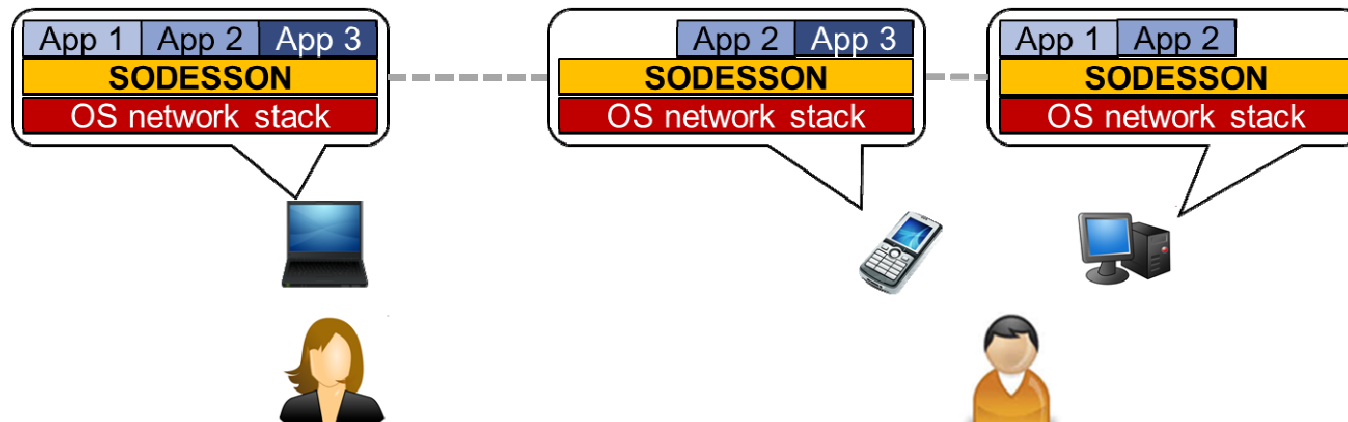
- Geringer Such- und Speicheraufwand
 - $O(\log N)$ bei Chord (und Pastry)
 - CAN tendiert zu linearem Aufwand bei Lookup
 - Aber nur $O(D)$ Speicheraufwand
 - Chord und Pastry sehr ähnlich
 - Pastry evtl. besser bei Lokalität
- Probleme:
 - Sicherheit
 - DoS Attacken gegen einzelne Knoten werden verkräftet
 - Sybil Attack: Angreifer versucht viele Knoten einzugliedern
 - Problem: Übernahme eines Knotens, gezielte Falschinformationen
 - Partitionierung: Problem bei manchen DHTs
 - echtes, dezentrales Bootstrapping?

10.5 Aktuelle P2P-Forschung am Institut

- P2P-Bootstrapping ohne Mogeln
- Simulation von P2P-Netzen (OverSim) 
- Dezentraler GeoCast für Smart Traffic (OverDrive) 
- Schutz der Privatsphäre im Smart Grid
- Implementierung des IETF RELOAD-Protokolls 
- Dezentrale soziale Netzwerke (SODESSON)

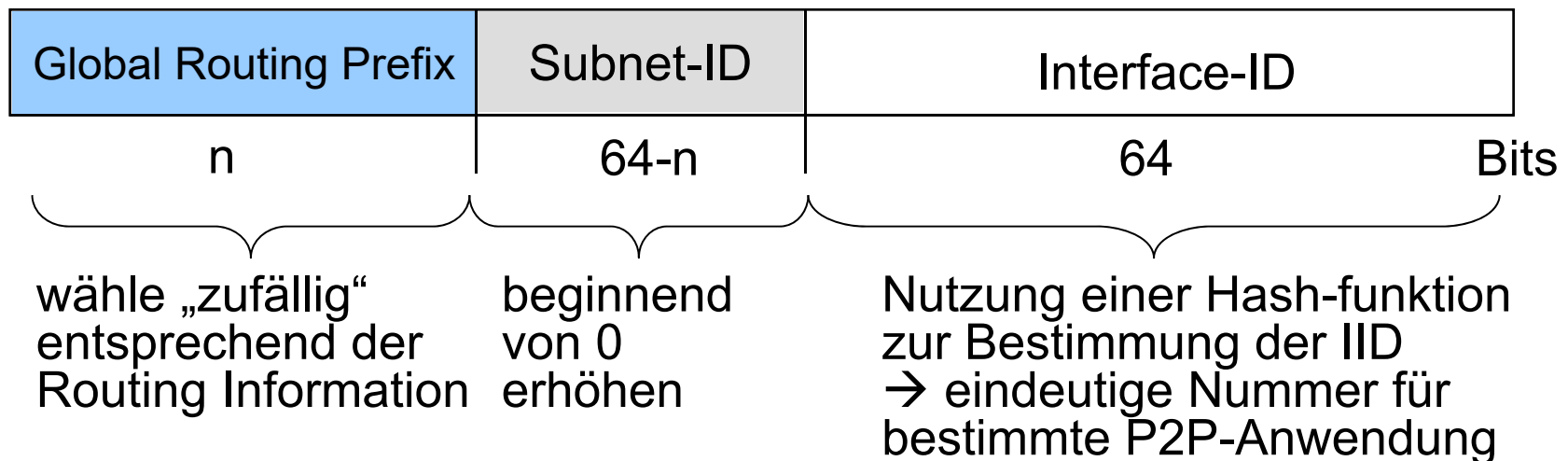
User-centric networking powered by SODESSON

- Today your mobile devices mainly access cloud services
 - Do you **always have cheap Internet access available**?
 - Do you **trust the service provider**?
- SODESSON: A user-centric middleware between OS and applications
 - No third party providers – **just you and your friends!**
 - Prefer **local communication**, only use Internet if needed
 - **Unified contact list** and **security** across all applications and devices



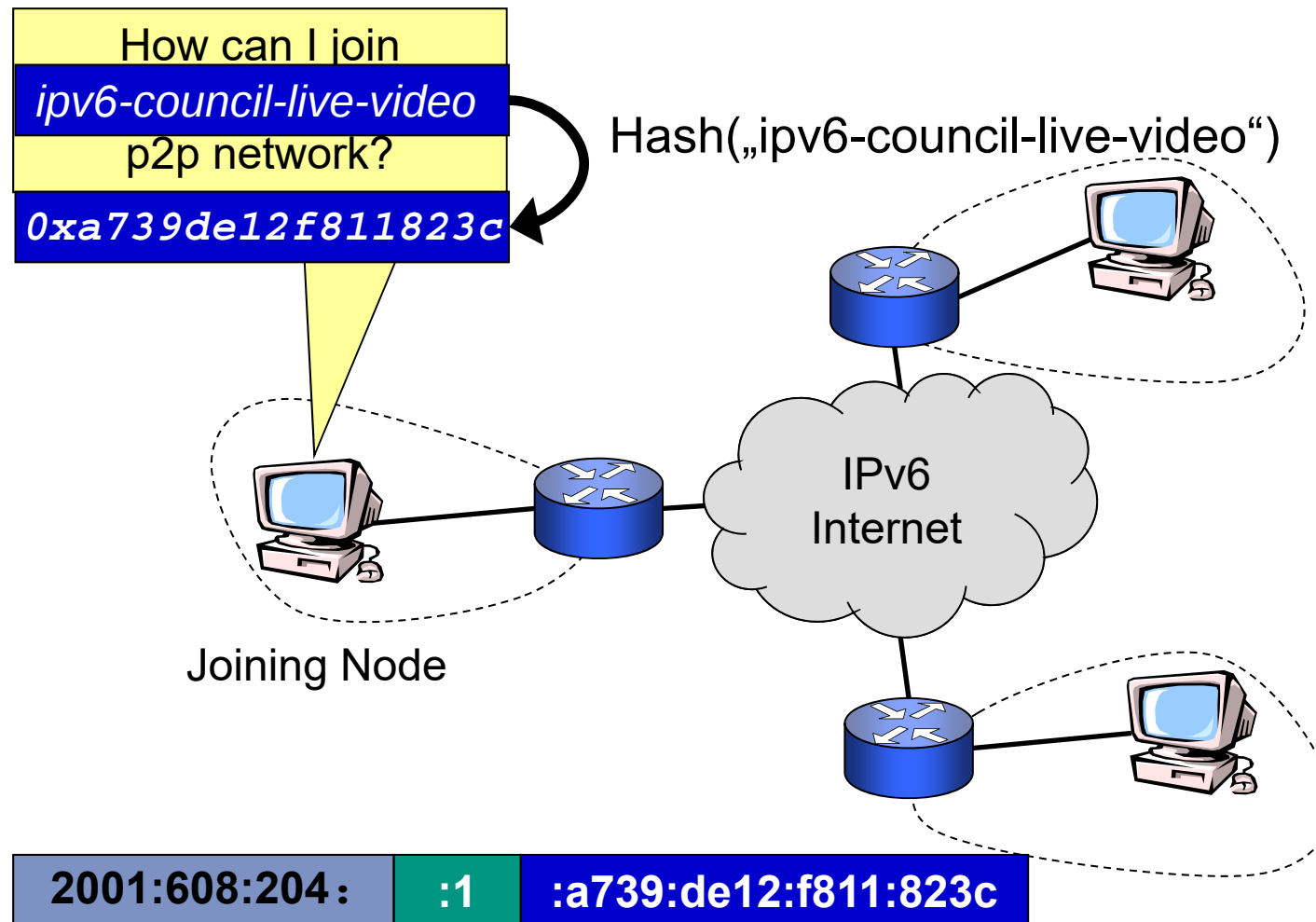
10.5 Dezentrales Bootstrapping mit IPv6

- Gewinner des intern. IPv6 Ideenwettbewerbs 2009
- Zielgerichtete, zufällige Adresswahl für IPv6
 - Ausnutzung des großen IPv6-Adressraums
 - Netzwerkteil der Adresse wird zufällig probiert

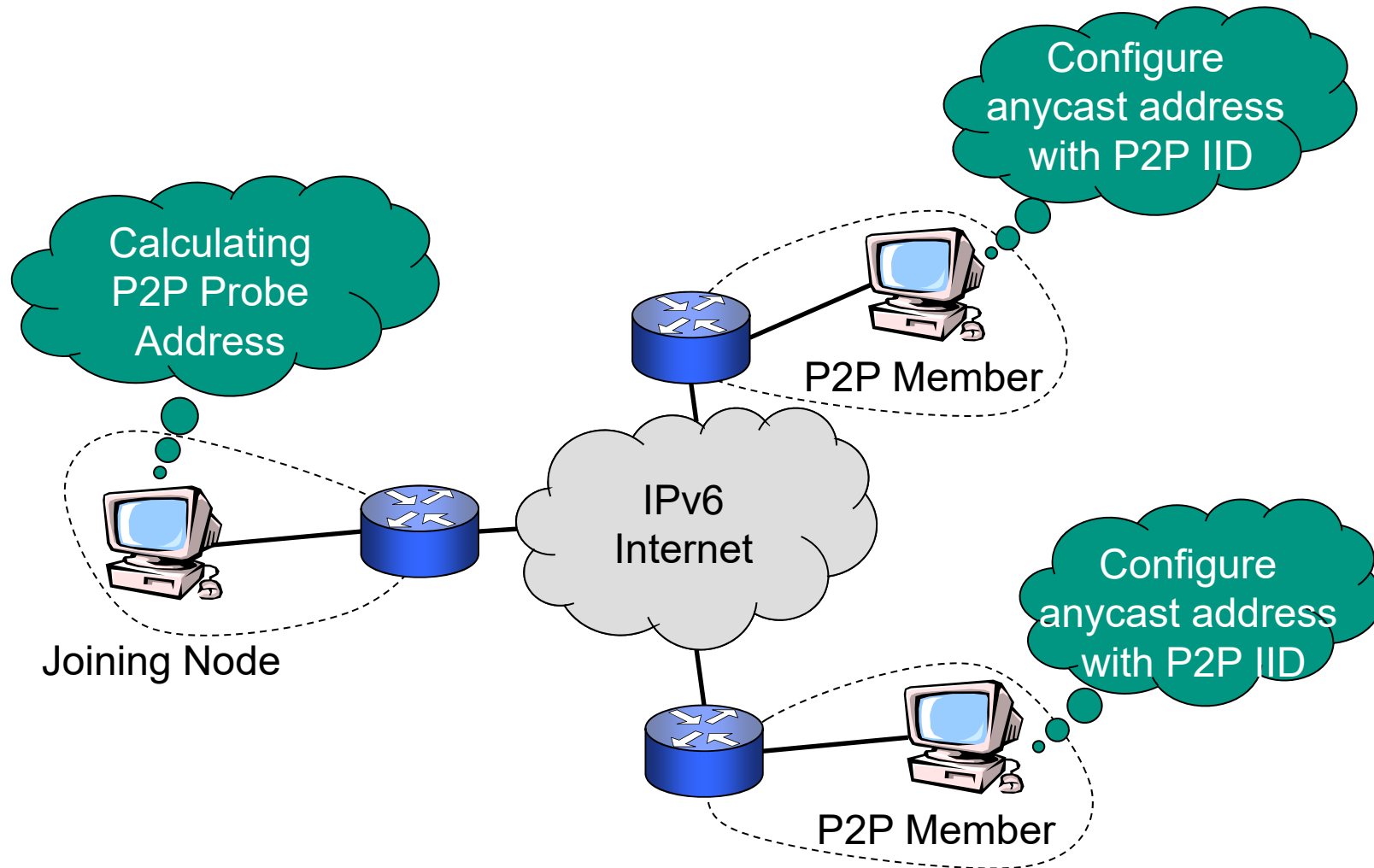


- Integrierte Peers registrieren die Adresse als
 - Unicast Adresse (einfacher Ansatz), oder
 - Anycast Adresse (fortgeschrittener Ansatz)

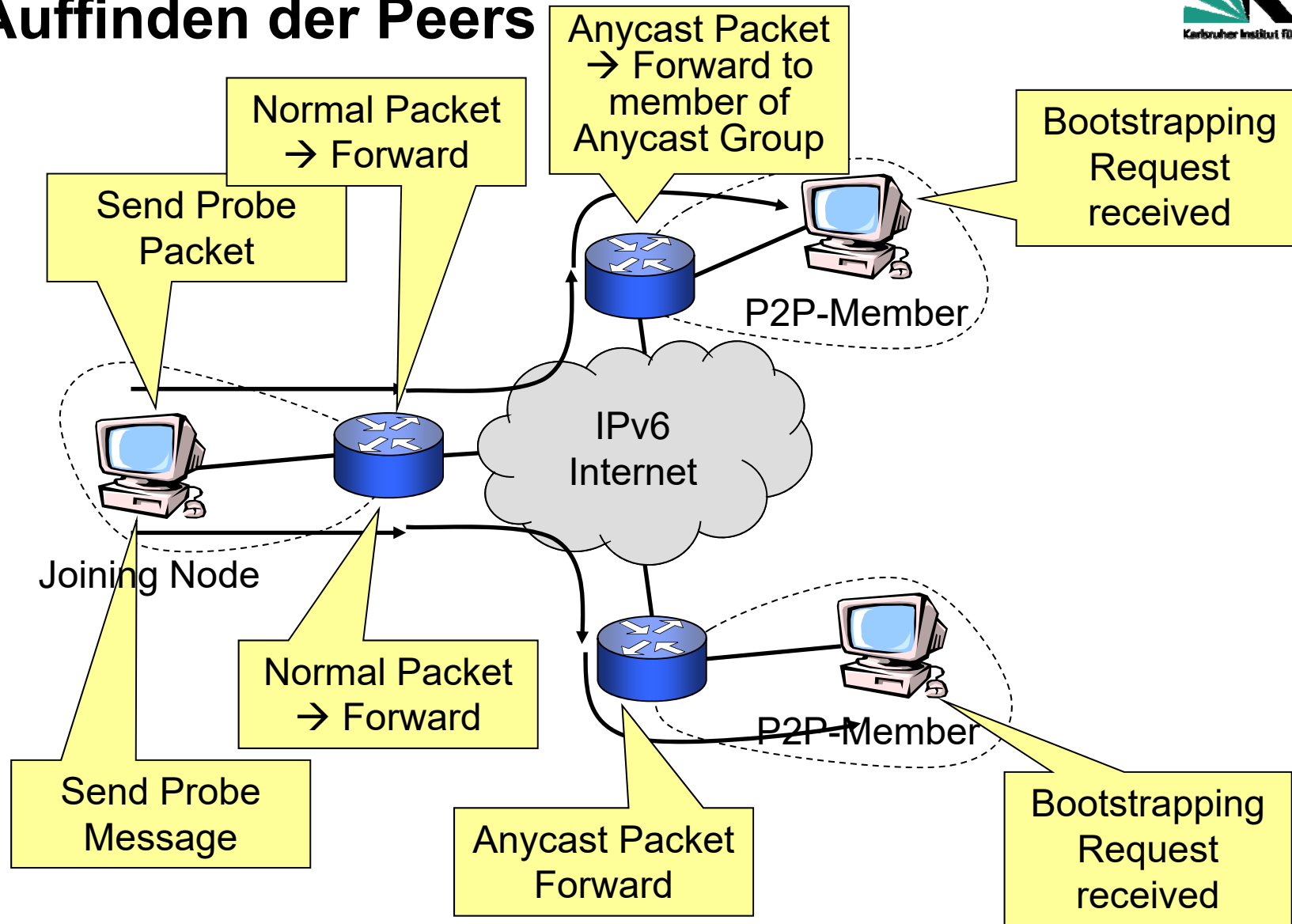
Erzeugung der Ziel-Adresse



Registrierung von Adressen



Auffinden der Peers



10.6 Übungen

- 10.1 Was kennzeichnet P2P-Anwendungen?
- 10.2 Warum war Napster keine reine P2P-Anwendung?
- 10.3 Warum skaliert Gnutella nicht?
- 10.4 Was sind die Hauptunterschiede zwischen BitTorrent und Gnutella?
- 10.5 Woher weiß ein Chord-Knoten, welche anderen q Knoten existieren?
- 10.6 Wie bekommt ein Chord-Knoten eine ID zugeordnet?
- 10.7 Wie wird ein Knotenausfall in Chord behandelt?

10.7 Referenzen und weiterführende Literatur

■ P2P Allgemein

- [10.0.0] Ralf Steinmetz, Klaus Wehrle (Eds): Peer-to-Peer Systems and Applications, LNCS 3854, Springer 2005
- [10.0.1] B. Moore, J. Hebeler; P2P – Building secure, scalable and managable networks; McGraw Hill, 2001
- [10.0.2] Andy Oram (Ed.); Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology, O'Reilly, 2001
- [10.0.3] Bo Leuf: Peer to Peer: Collaboration and Sharing over the Internet, Addison Wesley, 2002
- [10.0.4] Frode Kileng; Peer-to-Peer file sharing technologies – Napster, Gnutella and beyond, R&D Report 18/2001, 2001
- [10.0.5] Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble: A Measurement Study of Peer-to-Peer File Sharing Systems, Technical Report, UW-CSE-01-06-02, 2002
- [10.0.6] Dejan S. Milojevic et al.: Peer-to-Peer Comupting, Technical Report, HP Laboratories Palo Alto, HPL-2002-57, 2002
- [10.0.7] IEEE Internet Computing, Januar/Februar 2002

■ Gnutella

- [10.1.1] Gnutella.com Webseite: <http://www.gnutella.com>
- [10.1.2] Gnutelliums Webseite: <http://www.gnutelliums.com>
- [10.1.2] Clip2 Distributed Search Services: The Gnutella Protocol Specification v0.4
- [10.1.3] Matei Ripeanu, Adriana Iamnitchi, Ian Foster: Mapping the Gnutella Network, IEEE Internet Computing, Januar/Februar 2002
- [10.1.4] Igor Ivkovic: Improving Gnutella Protocol: Protocol Analysis And Reserach Proposal, September 2001
- [10.1.5] K. Aberer, M. Puceva, M. Hauswirth, R. Schmidt: Improving Data Access in P2P Systems, IEEE Internet Computing, Januar/Februar 2002
- [10.1.6] K. Truelove, A. Chasin: Morpheus Out of the Underworld, <http://www.openp2p.com>, 2001
- [10.1.7] Adar, Hubermann; Free Riding on Gnutella, http://www.firstmonday.uk/issues/issue5_10/adar/index.html

■ Small World

- [10.2.1] Reka Albert, Albert-Laszlo Barabasi: Statistical mechanics of complex networks, Reviews of Modern Physics, Vol. 74, 2002
- [10.2.2] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, Bernardo A. Huberman: Search in Power-Law Networks, Physical Review E, Volume 64
- [10.2.3] Lada A. Adamic, Rajan M. Lukose, Bernardo A. Huberman: Local Search in Unstructured Networks, Wiley-VCH Verlag Berlin
- [10.2.4] Qin Lv, Sylvia Ratnasamy, Scott Shenker: Can Heterogeneity Make Gnutella Scalable?, 2002
- [10.2.5] Beverly Yang, Hector Garcia-Molina: Improving Search in Peer-to-Peer Networks, 2001
- [10.2.6] Beom Jun Kim, Chang No Yoon, Seung Kee Han, Hawoong Jeong: Path finding strategies in scale-free networks, Physical Review E, Vol. 65, 2002
- [10.2.7] Qin Lv, Pei Cao, Edith Cohen, Kai Li, Scott Shenker: Search and Replication in Unstructured Peer-to-Peer Networks, 2001
- [10.2.8] Reka Albert, Hawoong Jeong, Albert-Laszlo Barabasi: The Internet's Achilles' Heel: Error and attack tolerance of complex networks, 2000

Literatur

- [10.2.9] William Aiello, Fan Chung, Linyuan Lu: A Random Graph Model for Massive Graphs, 32nd Annual Symposium on Theory of Computing, 2000
- [10.2.10] Edith Cohen, Scott Shenker: Replication Strategies in Unstructured Peer-to-Peer Networks, ACM SIGCOMM, Pittsburgh, 2002
- [10.2.11] Duncan J. Watts, Steven H. Strogartz: Collective dynamics of small-world networks, Nature Vol. 393, 1998
- [10.2.12] Albert-Laszlo Barabasi, Reka Albert: Emergence of Scaling in Random Networks, Science Vol. 286, 1999
- [10.2.13] Michalis Faloutsos, Petros Faloutsos, Christos Faloutsos: On Power-Law Relationship of the Internet Topology, ACM SIGCOMM, 1999
- [10.2.14] Cooperative Association for Internet Data Analysis:
<http://www.caida.org>
- [10.2.15] R. Albert, H. Jeong, A.-L. Barabasi: Diameter of the World-Wide Web, Nature, Vol. 401, September 1999

Literatur

■ DHTs allgemein

- [10.3.1] R. Albert, A. Barabasi: „Statistical mechanics of complex networks“, Reviews of Modern Physics, Vol. 74, 2002
- [10.3.2] L.A. Adamic, R.M. Lukose, A.R. Puniyani, B.A. Huberman: „Search in Power-Law Networks“, Physical Review E, Volume 64
- [10.3.3] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, I. Stoica: “Load Balancing in Structured P2P Systems”, 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), Berkeley, Feb. 2003.
- [10.3.4] H. Balakrishnan, S. Shenker, M. Walfish: “Semantic-Free Referencing in Linked Distributed Systems”, 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), Berkeley, CA, February 2003.
- [10.3.5] T. Ottmann, P. Widmayer: „Algorithmen und Datenstrukturen“, Spektrum Akademischer Verlag, 2. Auflage, 2002
- [10.3.6] H. Balakrishnan, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica: „Looking up Data in P2P Systems“, Communications of ACM, Vol. 43, No.2, Feb. 2003

Literatur

- [10.3.7] K. Hildrun, J. Kubiawicz, S. Rao, B. Zhao: „Distributed Object Location in a Dynamic Network“, Proc. of 14th ACM Symp. On Parallel Algorithms and Architectures (SPAA), August 2002
- [10.3.8] C. Plaxton, R. Rajaraman, A. Richa: „Accessing nearby copies of replicated objects in a distributed environment“, Proc. of ACM Symp. On Parallel Algorithms and Architectures, June 1997
- [10.3.9] S. Ratnasamy, S. Shenker, I. Stoica: „Routing Algorithms for DHTs: Some Open Questions“, 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), Cambridge, February 2002
- [10.3.10] D. Karger, E. Lehmann, F. Leighton, et.al.: „Consistent hashing and random trees“, Proc. Of 29th Annual ACM Symposium on Theory of Computing, El Paso, Mai 1997
- [10.3.11] F. Dabek and B. Zhao and P. Druschel and I. Stoica: „Towards a common API for structured peer-to-peer overlays “, Workshop on Peer-to-Peer Systems (IPTPS '03), Berkeley, CA, February 2003.

■ Chord (Mehr Literatur zu Chord <http://www.pdos.lcs.mit.edu/chord/>)

- [10.5.1] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan: "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", Proc. of ACM SIGCOMM'01, San Diego, September 2001.
- [10.5.2] F. Dabek, E. Brunskill, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica, H. Balakrishnan: „Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service“, Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), Mai 2001
- [10.5.3] D. Liben-Nowell, H. Balakrishnan, D. Karger: “Observations on the Dynamic Evolution of Peer-to-Peer Networks”, Proc. of the 1st Intern. Workshop on Peer-to-Peer Systems (IPTPS '02), Cambridge, März, 2002
- [10.5.4] E. Sit, R.T. Morris: “Security Considerations for Peer-to-Peer Distributed Hash Tables”, Proc. of the 1st Intern. Workshop on Peer-to-Peer Systems (IPTPS '02), Cambridge, März, 2002
- [10.5.5] D. Liben-Nowell, H. Balakrishnan, D. Karger: “Analysis of the Evolution of Peer-to-Peer Systems“, Proc. of ACM Conf. on Principles of Distributed Computing (PODC), Monterey, Juli 2002
- [10.5.6] D.R. Karger, M. Ruhl: „Finding Nearest Neighbors in Growth-restricted Metrics“, ACM Symposium on Theory of Computing (STOC '02), Montréal, Mai 2002
- [10.5.7] Stoica, I. Adkins, D. Shelley Zhuang Shenker, S. Surana, S., Internet indirection infrastructure, IEEE/ACM Transactions on Networking, Volume: 12 , Issue: 2, April 2004, pp.205 – 218; s.a. <http://i3.cs.berkeley.edu/>

■ Pastry

- [10.4.1] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg, November, 2001
- [10.4.2] R. Mahajan, M. Castro and A. Rowstron, "Controlling the Cost of Reliability in Peer-to-peer Overlays", IPTPS'03, Berkeley, CA, February 2003.
- [10.4.3] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks", SIGOPS European Workshop, France, September, 2002.
- [10.4.4] M. Castro, P. Druschel, Y. C. Hu and A. Rowstron, "Exploiting network proximity in peer-to-peer overlay networks", Proc. Of Intern. Workshop on Future Directions in Distributed Computing, Bertinoro, Italy, June, 2002.
- [10.4.5] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach: "Security for structured peer-to-peer overlay networks". Proc. of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02), Boston, Dezember 2002

Mehr Literatur zu Pastry unter <http://research.microsoft.com/~antr/Pastry/pubs.htm>

■ CAN

- [10.6.1] S. Ratnasami, P. Francis, M. Handley, R. Karp, S. Shenker: „A Scalable Content-Addressable Network“, Proc. ACM SIGCOMM 2002, San Diego
- [10.6.2] S. Ratnasami: „A Scalable Content-Addressable Network“, Ph.D. Thesis, UC Berkeley, Oktober 2002
- [10.6.3] S. Ratnasami, M. Handley, R. Karp, S. Shenker: “Application-level Multicast using Content-Addressable Networks”, Proc. of NGC 2001.

■ Weitere DHT-Modelle

- [10.7.1] B.Y. Zhao, J. Kubiawicz, A. Joseph: „Tapestry: An Infrastructure for Fault-tolerant Wide-Area Location and Routing“, Technical Report, UC Berkeley
- [10.7.2] B. Silaghi, B. Bhattacharjee, P. Keleher: „Query Routing in the TerraDir Distributed Directory“, Proc. of SPIE ITCOM'02
- [10.7.3] I. Clarke, O. Sandberg, B. Wiley, T. Hong: „Freenet: A distributed anonymous information storage and retrieval system“, Proc. of ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, Juni, 2000
- [10.7.4] Freenet Homepage; <http://freenetproject.org/>
- [MaMa02] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In Proceedings of IPTPS02, Cambridge, USA, March 2002. <http://www.cs.rice.edu/Conferences/IPTPS02/>
- [MoAm06] L. Monnerat, C. Amorim: D1HT: A Distributed One Hop Hash Table, Proceedings of IEEE IPDPS, April 2006