

Einführung in Peer-to-Peer Netzwerke

- Die Geschichte von Peer-to-Peer
- Die Natur von Peer-to-Peer: Paradigmen
- Unstrukturierte Peer-to-Peer Systeme
- Strukturierte Peer-to-Peer Systeme



A Peer-to-Peer system is a self-organizing system of equal, autonomous entities (peers) which aims for the shared usage of distributed resources in a networked environment avoiding central services.

Andy Oram



Die alte Zeit

- ▶ NetNews (nntp)
 - ▶ Usenet seit 1979, anfänglich realisiert durch UUCP
 - ▶ Austausch (Replikation) von abonnierten News Artikeln
 - ▶ Gruppenerzeugung und Löschung dezentral
- ▶ DNS
 - ▶ Verteilte Delegation von Namenshoheiten:
file sharing der host Tabellen
 - ▶ Name "Servers" sind Peers
 - ▶ Hierarchischer Informationsraum erlaubt exponentielles Wachstum
- ▶ Systeme sind manuell konfigurierte verteilte Peers



SETI@home: Verteiltes Rechnen

➤ Search for Extraterrestrial Intelligence (SETI)

➤ Analysiert Funksignale aus dem All

➤ Global geteilte Rechenressource

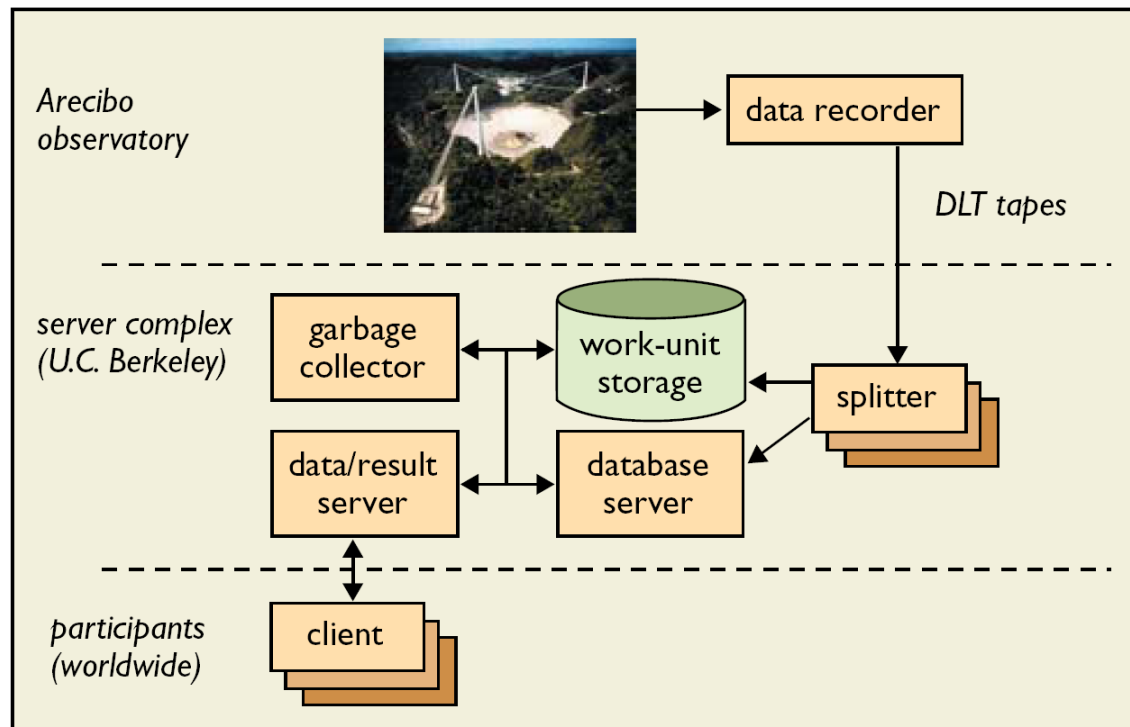
➤ Idee 1995

➤ Erste Version 1998

➤ 2002 \approx 4 Mio clnts

➤ Z.B. Screensaver

➤ <http://setiathome.berkeley.edu/> - läuft weiter



From Anderson et. al.: *SETI@home*, Comm. ACM, 45 (11), Nov. 2002



Napster



- ▶ Mai 1999: Aufrütteln der Internet Gemeinde, die erste Generation von File Sharing: Napster
 - ▶ Anwender laden nicht nur Daten, sondern offerieren selber
 - ▶ Teilnehmer etablieren ein virtuelles Netzwerk, vollständig unabhängig von physikalischem Netzwerk und administrativen Autoritäten oder Beschränkungen
 - ▶ Basis: UDP und TCP Kanäle zwischen Peers
- ▶ Napster steuert zentralisiertes Indexing bei
 - ▶ Clients laden ihre Dateiliste auf Napster Server
 - ▶ Clients befragen Index Server erhalten vollständige Anbieterliste
- ▶ Datenaustausch unmittelbar zwischen Peers



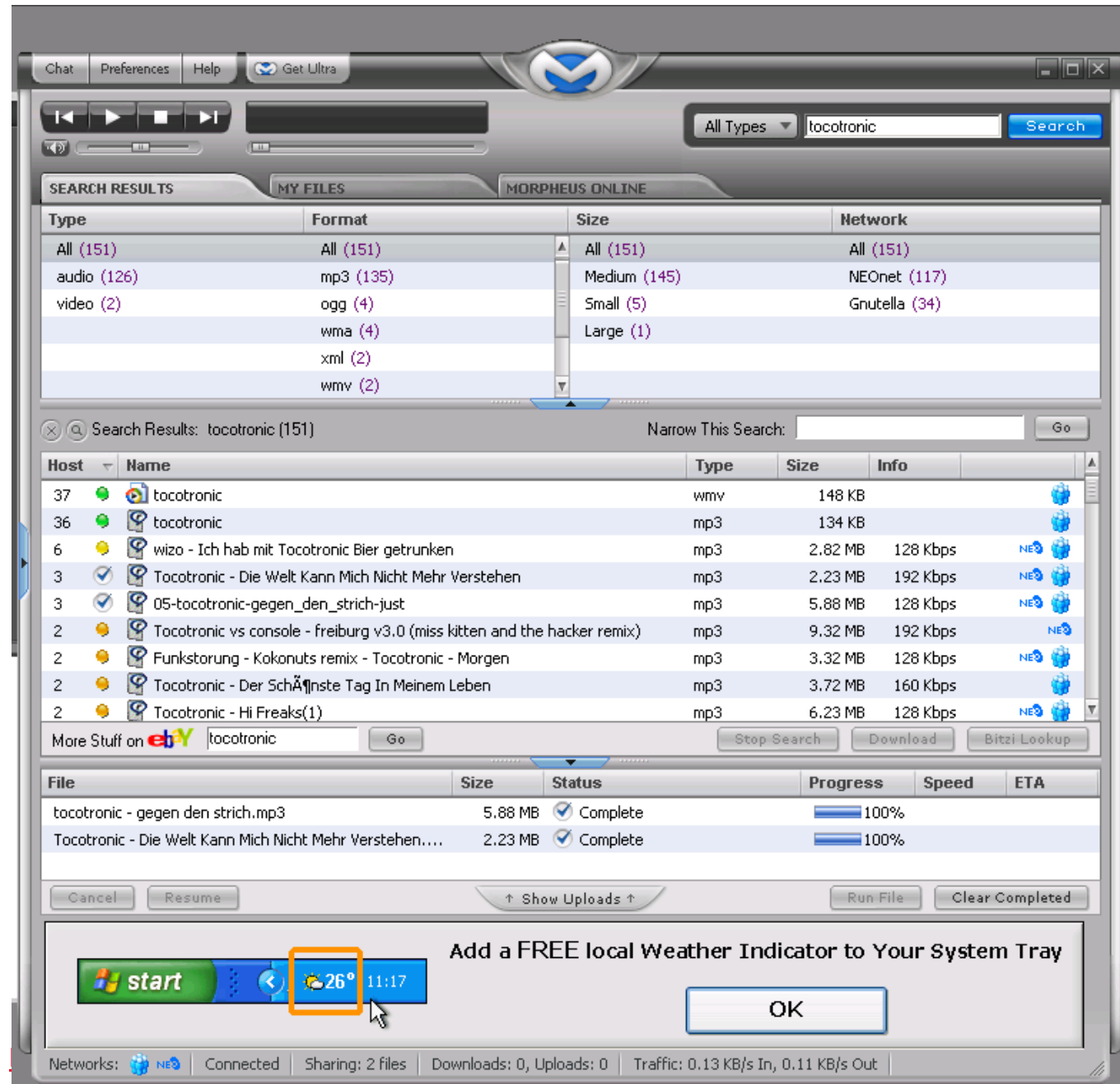
Napster

- ▶ Dezember 1999: RIAA klagt gegen Napster Inc.
- ▶ März 2000: University of Wisconsin identifiziert 25 % ihres IP Verkehrs als Napster Daten
- ▶ Februar 2001: 2.79 Milliarden Files werden pro Monat via Napster getauscht
- ▶ July 2001: Napster Inc. wird verurteilt
 - ▶ Angriffspunkt der RIAA: der zentrale Indexserver von Napster
 - ▶ Napster muss dessen Betrieb einstellen
 - ▶ Napster Netzwerk bricht zusammen
- ▶ Napster hat (technisch & rechtlich) an seinem Single Point of Failure versagt.



Gnutella

- ▶ Völlig dezentralisiertes File Sharing
- ▶ Open source software
- ▶ März 2000:
Release 0.4 – basiert auf Netzwerkfluten
- ▶ Baldiger Zusammenbruch
- ▶ Frühjahr 2001:
Release 0.6 – erhöhte Skalierbarkeit



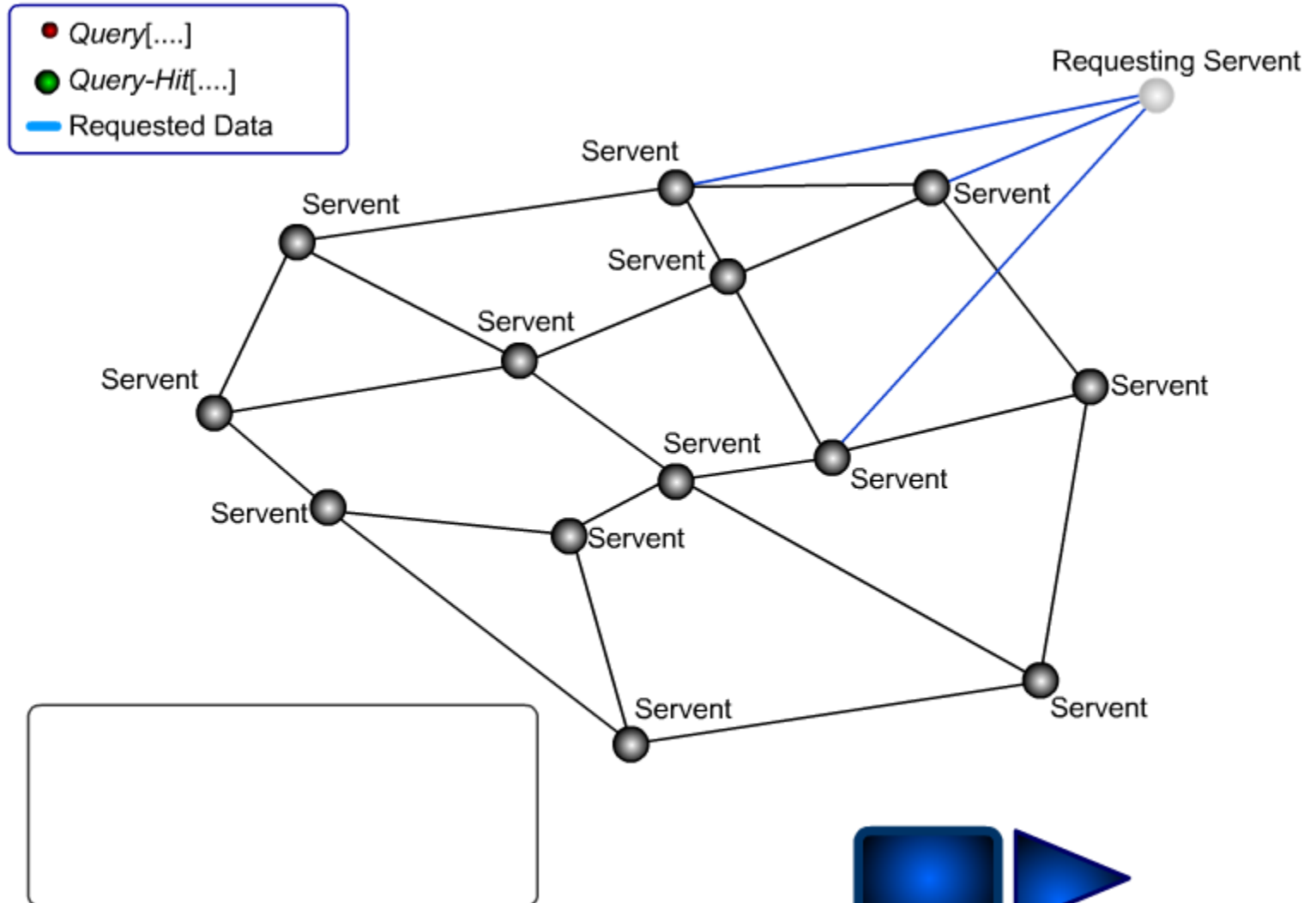
Gnutella 0.4



- Reines P2P System – kein zentraler Indexing Server
- Operationen:
 1. Verbinde mit einem aktiven Peer (Addressfindung beim Bootstrap)
 2. Erkunde Nachbarschaft (PING/PONG)
 3. Setze Query einer Liste von Schlüsselworten an die Nachbarn ab (diese leiten weiter)
 4. Suche "beste" der richtigen Antworten (die nach einer Weile eintrudeln)
 5. Verbinde mit anbietendem Peer
- Skalierungsprobleme durch Netzwerkfluten
- Suchen können irrtümlich erfolglos sein



Gnutella 0.4: Arbeitsweise



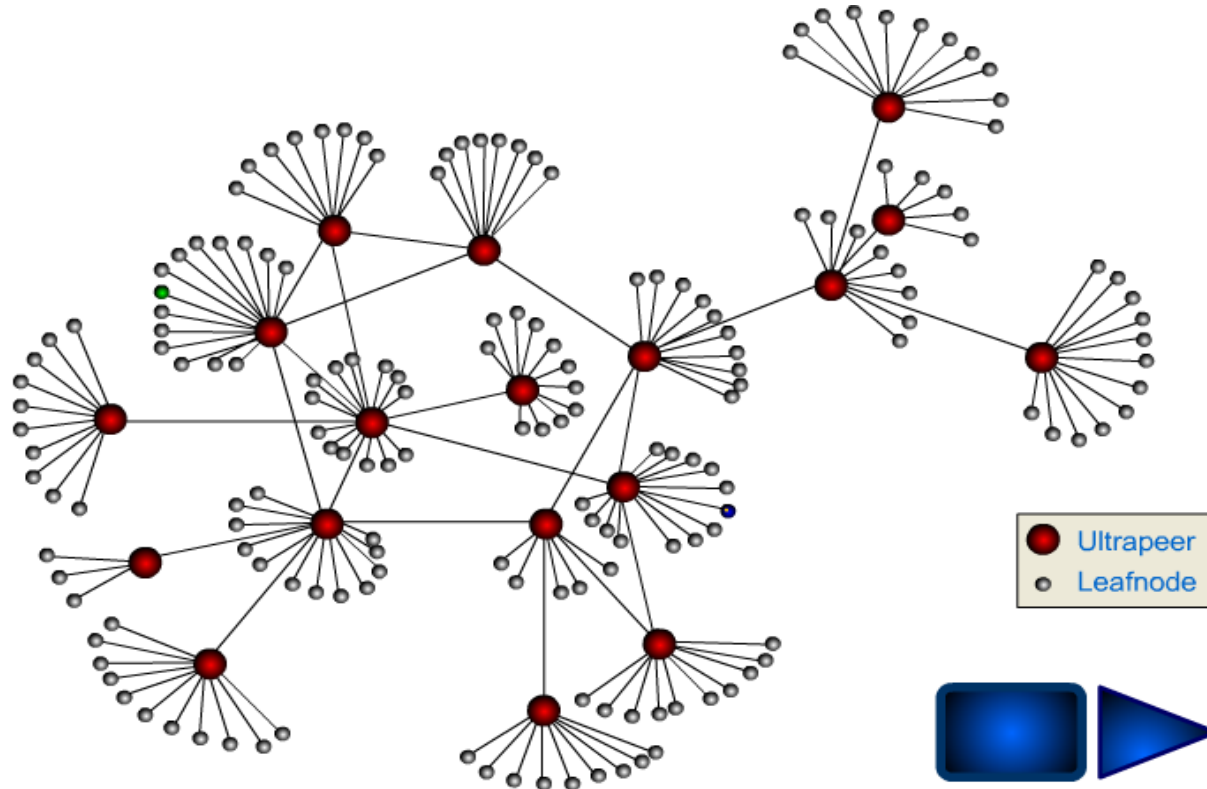
Gnutella 0.6



- ▶ Hybrides P2P System – Einführung von Superpeers
- ▶ Erhöhte Skalierbarkeit: Signalisierung wird auf Superpeers beschränkt
- ▶ Wahlmechanismus entscheidet über die Auszeichnung als Superpeer oder Leafnode (abhängig von Knotenkapazitäten, Netzwerkkonnektivität, Lebenszeit, ...)
- ▶ Leafnodes annoncieren ihr Inhaltsangebot zu ihrem Superpeer
- ▶ Superpeers halten lokale Routingtabellen



Gnutella 0.6: How Does It Work

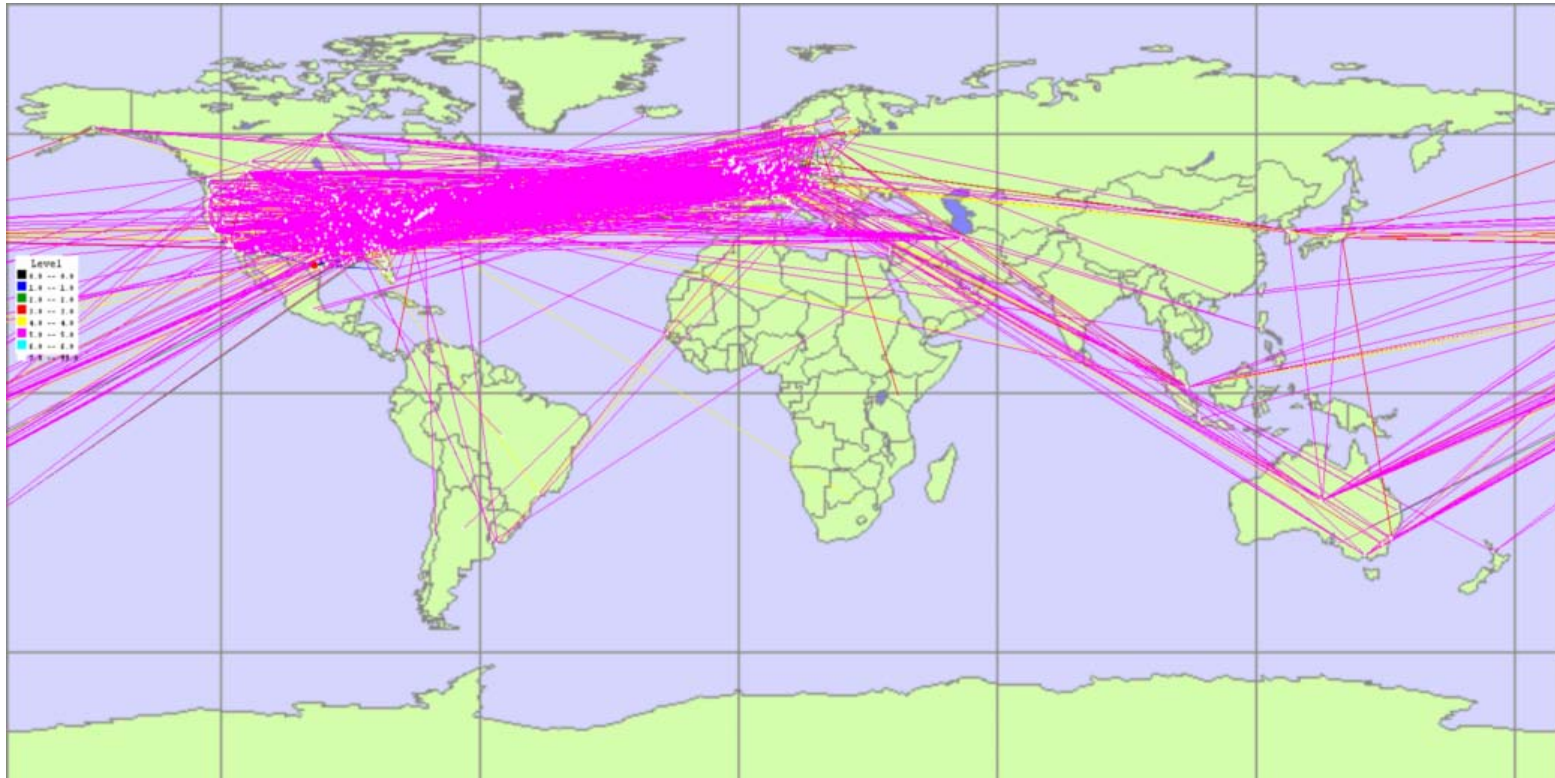


From:

J. Eberspächer, R. Schollmeier: *First and Second Generation Peer-to-Peer Systems*, in LNCS 3485



Das Gnutella Netzwerk



Measurements from May 2002

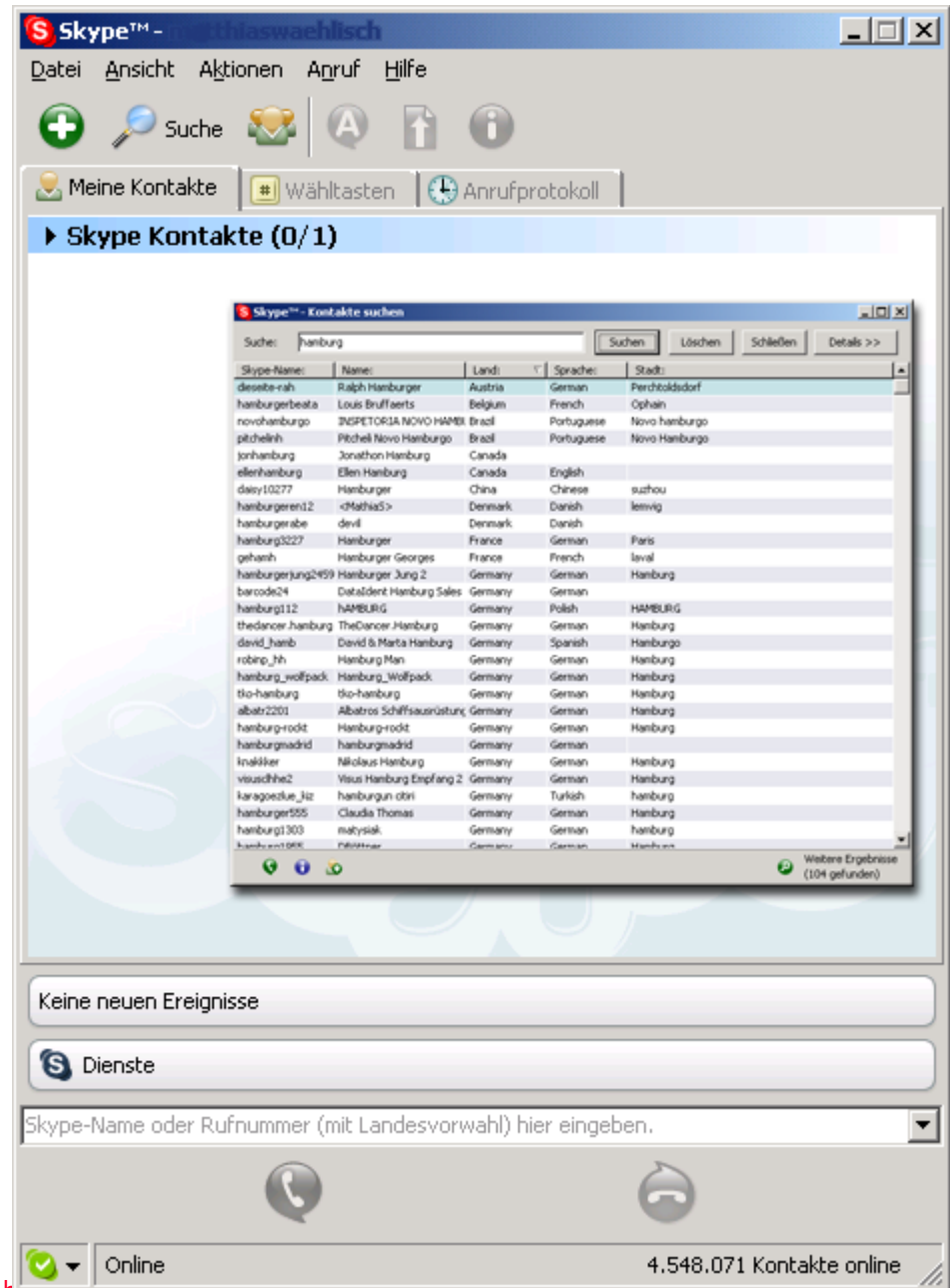
From:

J. Eberspächer, R. Schollmeier: *First and Second Generation Peer-to-Peer Systems*, in LNCS 3485



Skype

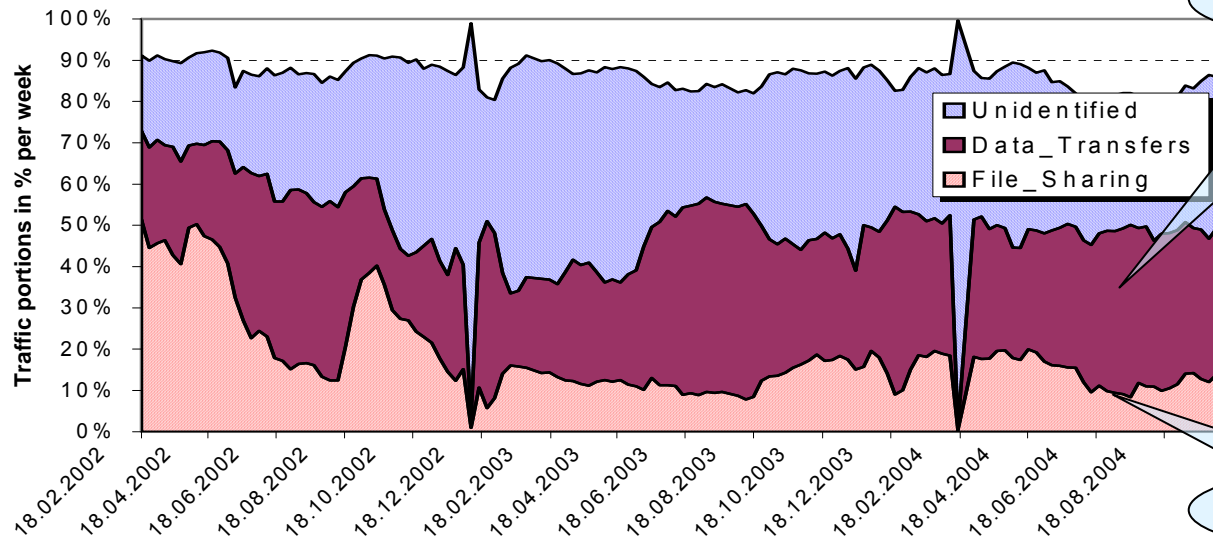
- VoIP Konferenzsystem
- Release 2003
- Zentraler Login-Server
- Sonst hybrides P2P System
- Kernfunktionen:
 - Teilnehmerlokalisierung
 - Traversierung von NAT & Firewalls (STUN)
- Wählt Superpeers nach Netzwerkkonnektivität
- Nutzt Superpeers als Relays



Einfluss von P2P auf das Abilene Backbone

- Unidentified + data_transfers + file_sharing verursacht 90% des datenverkehrs
- Unidentified traffic + data_transfers wächst signifikant
 - Teile von P2P-Daten bleiben verborgen (port hopping,...)
 - Einige P2P Applikationen nutzen Port 80 → data_transfers

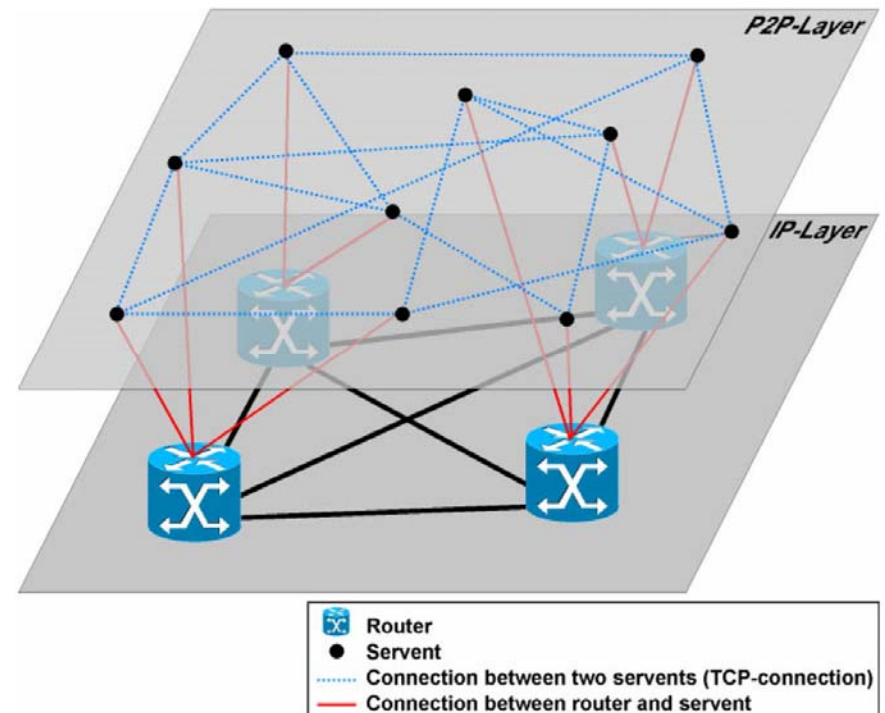
Core of Internet2 infrastructure, connecting 190 US universities and research centers



Data source: <http://netflow.internet2.edu/weekly/>

Die Natur von P2P

- ▶ P2P Netzwerke bilden ein Overlay der Netzwerk Infrastruktur
- ▶ Sind auf der Anwendungsschicht implementiert
- ▶ Die Overlay Topologie bildet ein virtuelles Signalisierungsnetzwerk via TCP Verbindungen
- ▶ Peers sind Inhaltsanbieter + Inhaltskonsumenten + Router im Overlay Netzwerk
- ▶ Adressen: Eindeutige IDs



P2P im Paradigma verteilter Systeme

- Koordination zwischen gleichen Komponenten
- Dezentralisiert & selbstorganisierend
- Unabhängig von individuellen Peers
- Skalierbar in extremem Maße
- Hohe Dynamik durch volatile Teilnehmer
- Geschützt gegen Fehler der Infrastruktur und Teilnehmer
- Anreizgesteuert statt kontrolliert



P2P im Paradigma des Internetworking

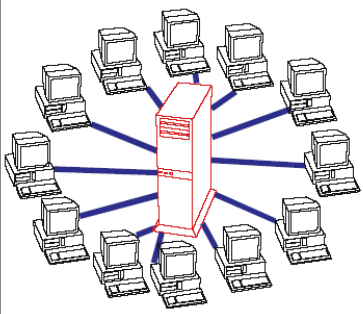
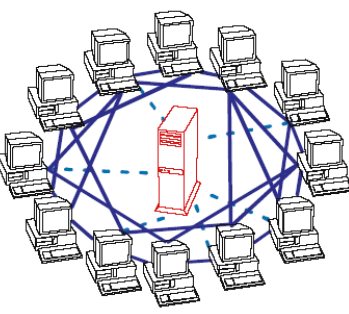
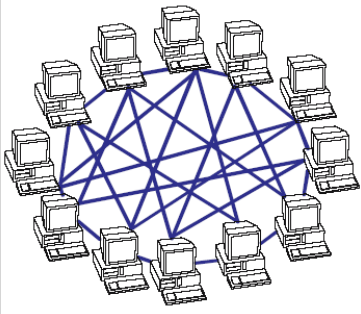
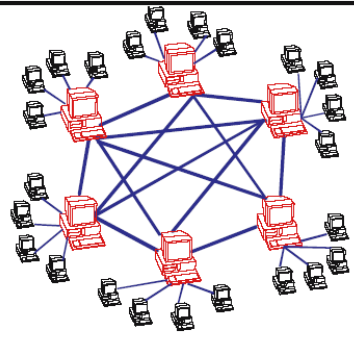
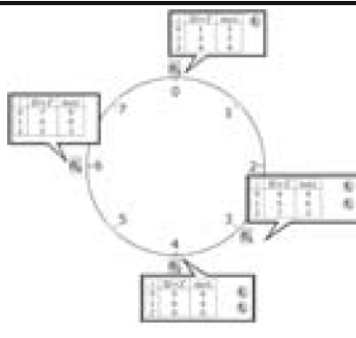
- Lose, zustandsfreie Koppelung zwischen Peers
 - Serverlos & ohne infrastrukturelle Entitäten
 - Dynamische Adaptation an die Netzwerk Infrastruktur
 - Überwindung von NATs und Portbarrieren
 - Client-Server reduziert auf die Kommunikationsschnittstelle, kein Anwendungsparadigma mehr
 - "Back to the Internet roots":
 - Freiheit von Information
 - Freiheit der Skala
- Aber auch: Freiheit von der Internet Infrastruktur & Regulierung**



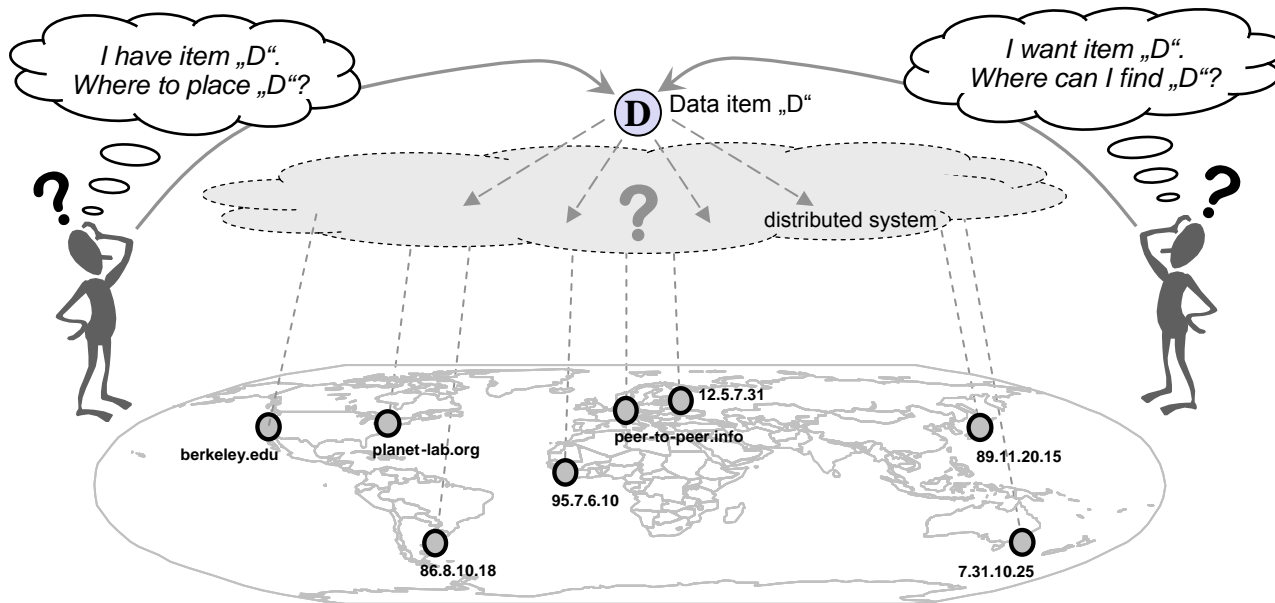
P2P Anwendungsgebiete

- File sharing
- Medienkonferenzen
- Resource Sharing: Grids
- Kollaborative Groupware
- Verteilte Spiele
- Inhaltsbasierte Netze: z.B. Semantische Netze
- Geschäftsanwendungen: z.B. "Market Management"
- Ubiquitous Computing: z.B. Vehicular Communication
- ...



Client-Server	Peer-to-Peer			
	<ol style="list-style-type: none"> 1. Resources are shared between the peers 2. Resources can be accessed directly from other peers 3. Peer is provider and requestor (Servent concept) 			
	Unstructured P2P			Structured P2P
	1st Generation		2nd Generation	
<ol style="list-style-type: none"> 1. Server is the central entity and only provider of service and content. → Network managed by the Server 2. Server as the higher performance system. 3. Clients as the lower performance system <p>Example: WWW</p>	<p><i>Centralized P2P</i></p> <ol style="list-style-type: none"> 1. All features of Peer-to-Peer included 2. Central entity is necessary to provide the service 3. Central entity is some kind of index/group database <p>Example: Napster</p>	<p><i>Pure P2P</i></p> <ol style="list-style-type: none"> 1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities <p>Examples: Gnutella 0.4, Freenet</p>	<p><i>Hybrid P2P</i></p> <ol style="list-style-type: none"> 1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → dynamic central entities <p>Example: Gnutella 0.6, JXTA</p>	<p><i>DHT-Based</i></p> <ol style="list-style-type: none"> 1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities 4. Connections in the overlay are "fixed" <p>Examples: Chord, CAN</p>
				

Herausforderungen in Peer-to-Peer Systemen



- **Ressourcenlokalisierung** der zwischen Systemen verteilten Daten
 - Wo sollen meine Daten gespeichert/registriert werden?
 - Wie findet ein Suchender diese auf?
- **Skalierbarkeit**: Beschränke die Komplexität von Kommunikation und Speicher
- **Robustheit und Korrektur** bei Fehlern und häufigen Wechseln

Unstrukturierte P2Ps - Resümee

Zwei Strukturansätze:

- ▶ Zentralisiert

- ▶ Einfache, flexible Suche auf Server ($O(1)$)
- ▶ Single point of failure, $O(N)$ Serverzustände

- ▶ Dezentralisiertes Fluten

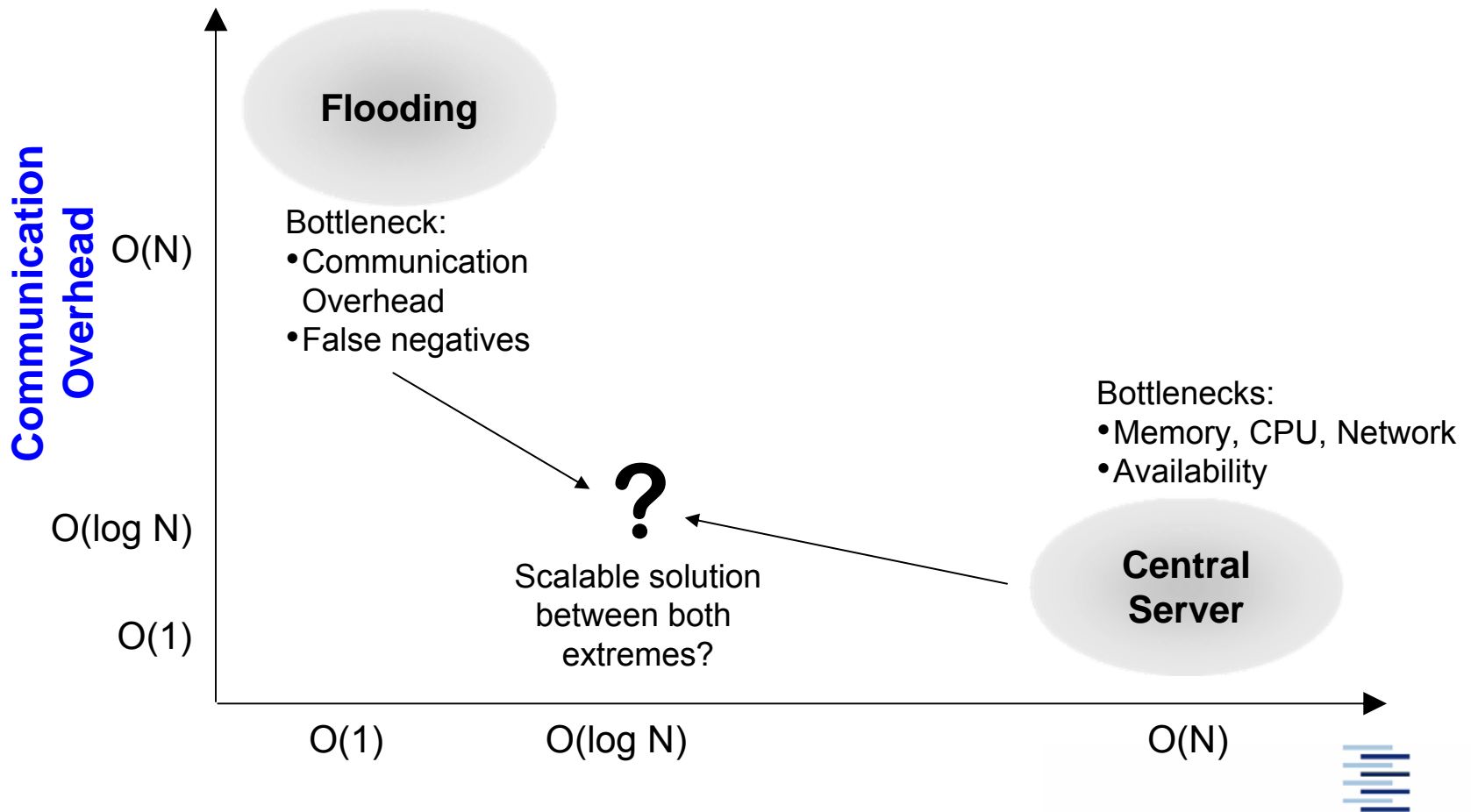
- ▶ Fehlertolerant, $O(1)$ Knotenzustände
- ▶ Kommunikationsaufwand $\geq O(N^2)$, Suchen teilweise erfolglos

Aber:

- ▶ Es gibt keine Referenzstruktur zwischen den Knoten

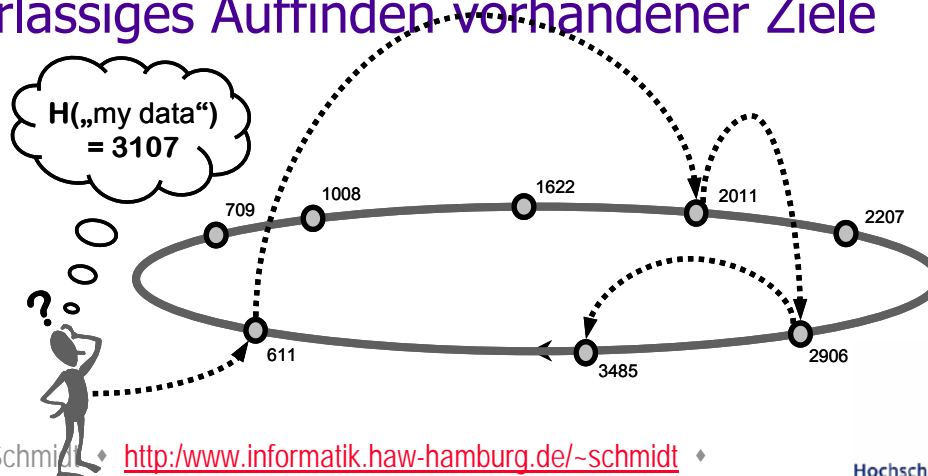


Unstrukturierte P2P: Komplexitäten



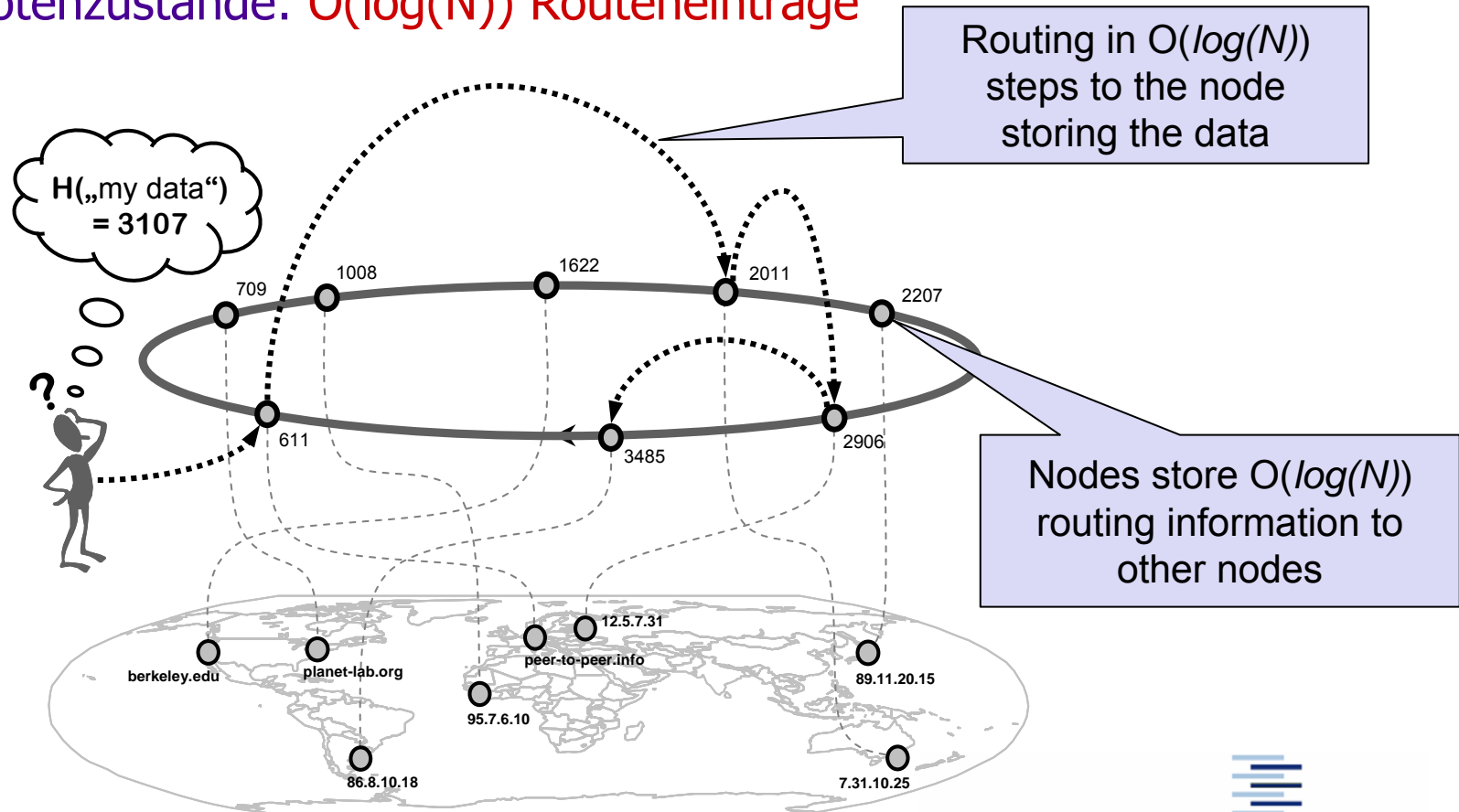
Idee: Verteilte Indexierung

- ▶ Ursprüngliche Ideen für verteilten gemeinsamen Speicher (1987 ff.)
- ▶ Knoten werden in einem Adressraum strukturiert
- ▶ Daten werden in den **selben** Adressraum abgebildet
- ▶ Zwischenknoten erhalten Routing-Informationen über Zielknoten
 - ▶ Effizientes Auffinden der Ziele
 - ▶ Zuverlässiges Auffinden vorhandener Ziele

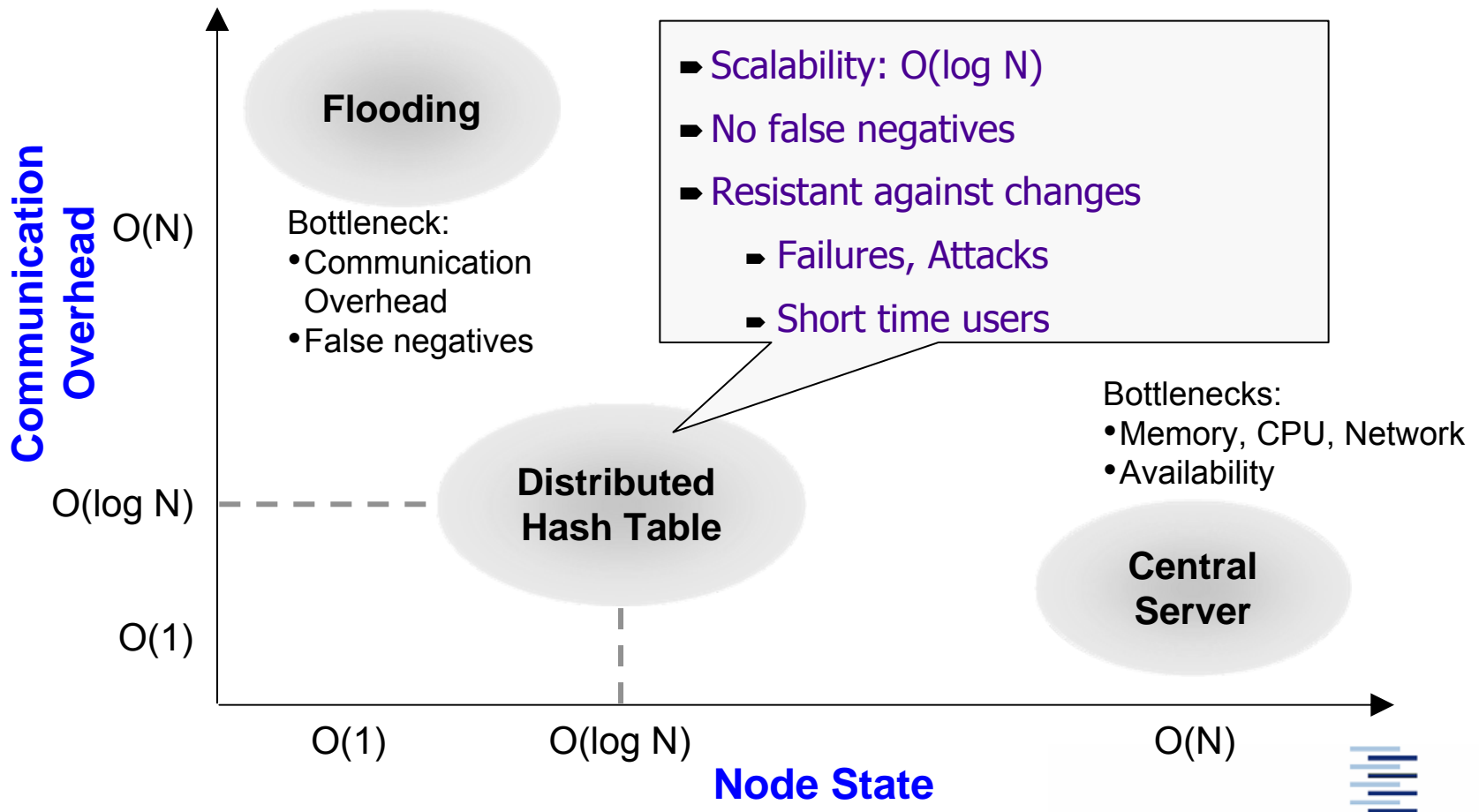


Skalierbarkeit verteilter Indexierung

- Kommunikationsaufwand: $O(\log(N))$ Hops
- Knotenzustände: $O(\log(N))$ Routeneinträge



Verteilte Indexierung: Komplexitäten



Grundlagen verteilter Hash Tabellen – Distributed Hash Tables (DHTs)

- Gewünschte Eigenschaften:
Flexibilität, Zuverlässigkeit, Skalierbarkeit
- Designherausforderungen für DHTs:
 - Gleichverteilung der Inhalte auf die Knoten
 - Kritisch für effiziente Inhaltsauffindung
 - Kontinuierliche Anpassung an Fehler, Joins, Exits
 - Zuweisung von Verantwortlichkeiten auf neue Knoten
 - Neuverteilung von Verantwortlichkeiten bei Knotenwegfall
 - Pflege der Routing Informationen



Verteiltes Datenmanagement

1. **Abbildung der Knoten und Daten in den selben Adressraum**
 - Peers und Inhalte nutzen flache Identifizierer (IDs)
 - Knoten verantwortlich für definierte Teile des Adressraums
 - Assoziation von Daten zu Knoten ändert sich, wenn Knoten verschwinden
2. **Speichern / Auffinden von Daten in der DHT**
 - Suche von Daten = Routing zu verantwortlichem Knoten
 - Verantwortlicher Knoten muss nicht vorab bekannt sein
 - Zuverlässige Aussage über die Verfügbarkeit von Daten



Addressierung in verteilten Hash Tabellen

Step 1: Abbildung von Inhalten/Knoten in linearen Adressraum

- Gewöhnlich: $0, \dots, 2^m - 1 \gg$ Anzahl zu speichernder Objekte
- Adressabbildung per Hash Funktion
 - z.B. $\text{Hash}(\text{String}) \bmod 2^m: H(\text{„my data“}) \rightarrow 2313$
- Zuordnung von Adressbereichen zu DHT Knoten

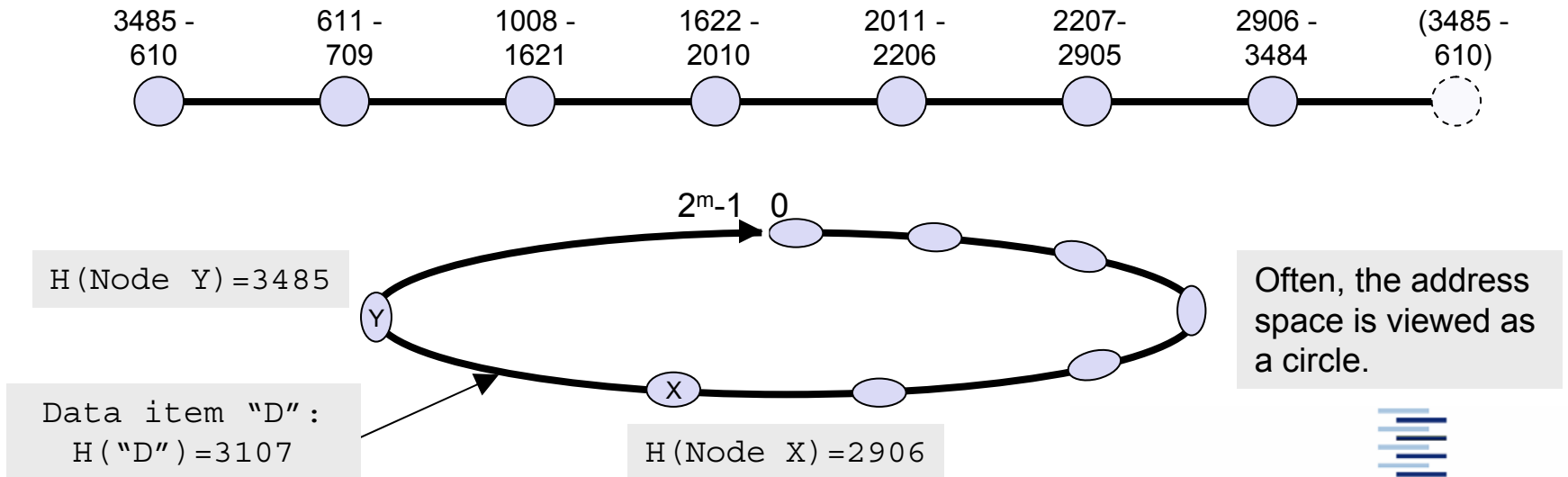


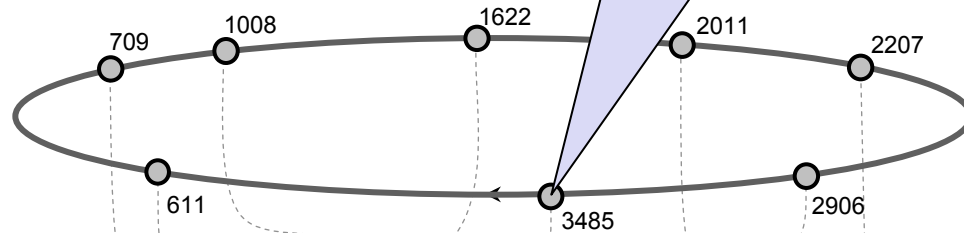
Abbildung des Adressraums zu Knoten

Jeder Knoten ist verantwortlich für Teil des Wertebereichs

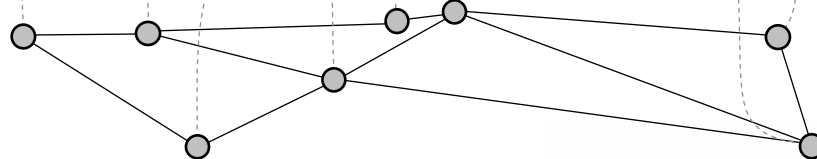
- Oft redundant (Überlappen der Bereiche)
- Kontinuierliche Anpassung
- Reale (Underlay) und logische (Overlay) Topologie unkorreliert

Node 3485 is responsible for data items in range 2907 to 3485 (in case of a Chord-DHT)

Logical view of the Distributed Hash Table



Mapping on the real topology

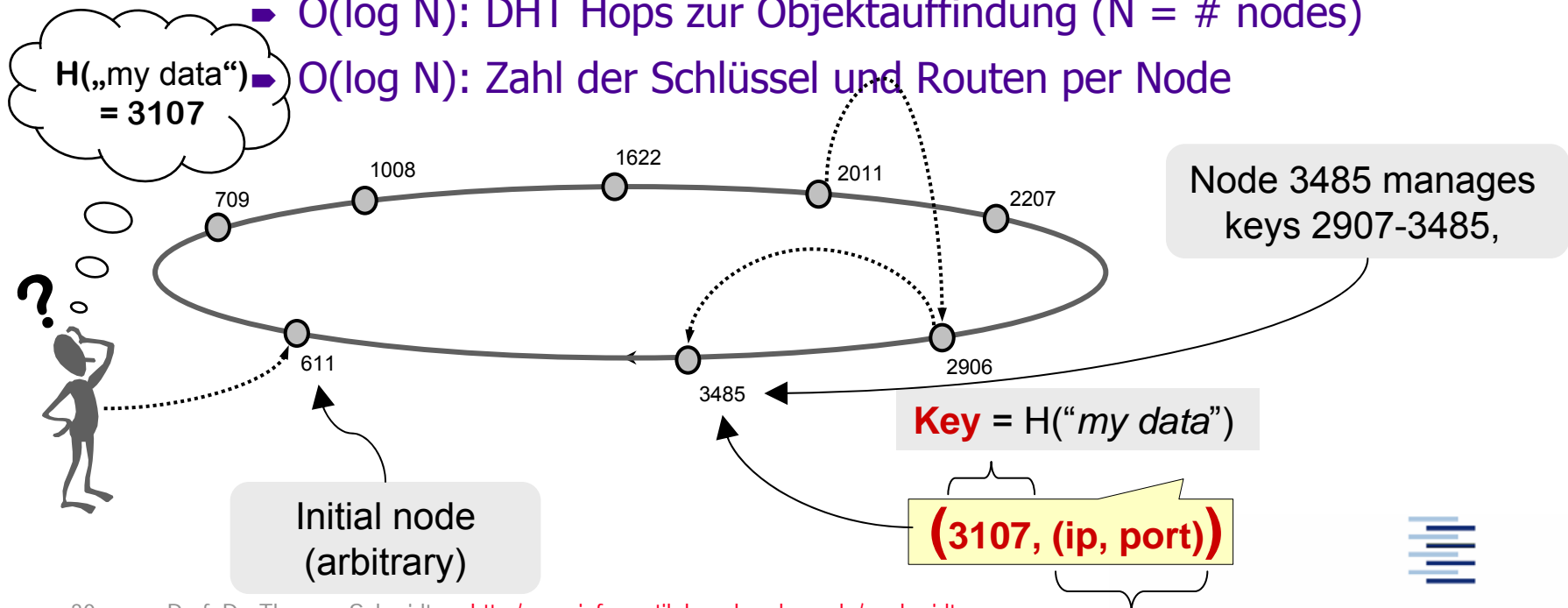


Auffinden/Routen zu Daten

Step 2: Lokalisierung der Daten (content-based routing)

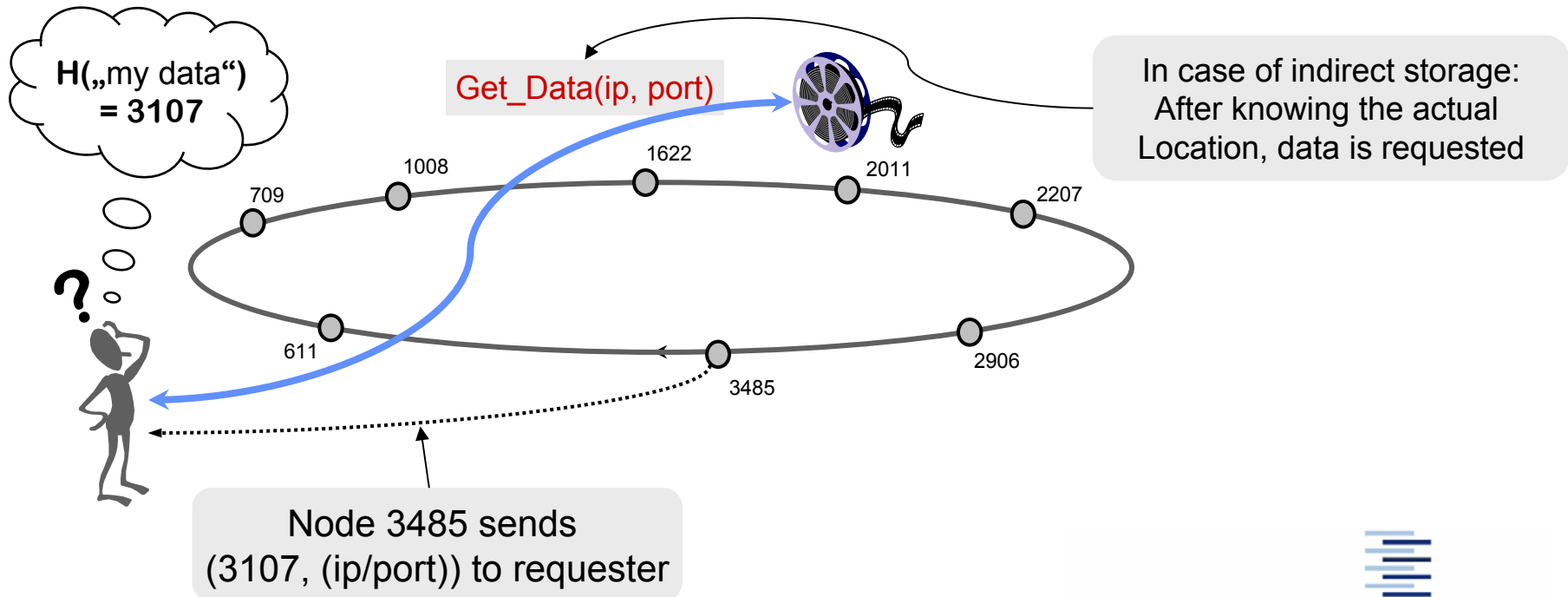
Ziel: Einfach, mit skalierbarem Aufwand

- ▶ $O(1)$ mit zentraler Hash Tabelle
- ▶ Minimaler Overhead mit Distributed Hash Tables
- ▶ $O(\log N)$: DHT Hops zur Objektauffindung ($N = \# \text{ nodes}$)
- ▶ $O(\log N)$: Zahl der Schlüssel und Routen per Node



Auffinden/Routen zu Daten (2)

- Zugriff auf Kontent
 - K/V-Paar wird an Suchenden zurückgegeben
 - Suchender analysiert K/V-Tupel
(und lädt Daten von Speicherort – im indirekten Fall)



DHT Algorithmen

- **Lookup algorithm** for nearby objects (Plaxton et al 1997)
 - Vor P2P ... später in Tapestry genutzt
- **Chord** (Stoica et al 2001)
 - Unmittelbare Umsetzung einer 1-dim. DHT
- **Pastry** (Rowstron & Druschel 2001)
 - Entfernungsoptimierte Nachbarschaftswahl
- **CAN** (Ratnasamy et al 2001)
 - Routenoptimierung in multidimensionalem ID-Raum
- **Kademlia** (Maymounkov & Mazières 2002) ...



Chord: Überblick

- ▶ Früher, erfolgreicher Algorithmus
- ▶ Simpel & elegant
 - ▶ einfach zu verstehen und zu implementieren
 - ▶ viele Erweiterungen und Optimierungen
- ▶ Kernaufgaben:
 - ▶ Routing
 - ▶ Logischer Adressraum: I-bit Identifiers statt IPs
 - ▶ Effizientes Routing: $\log(N)$ Hops
 - ▶ Selbstorganisation
 - ▶ Verarbeitung von Knoteneintritt, -austritt und -fehlern



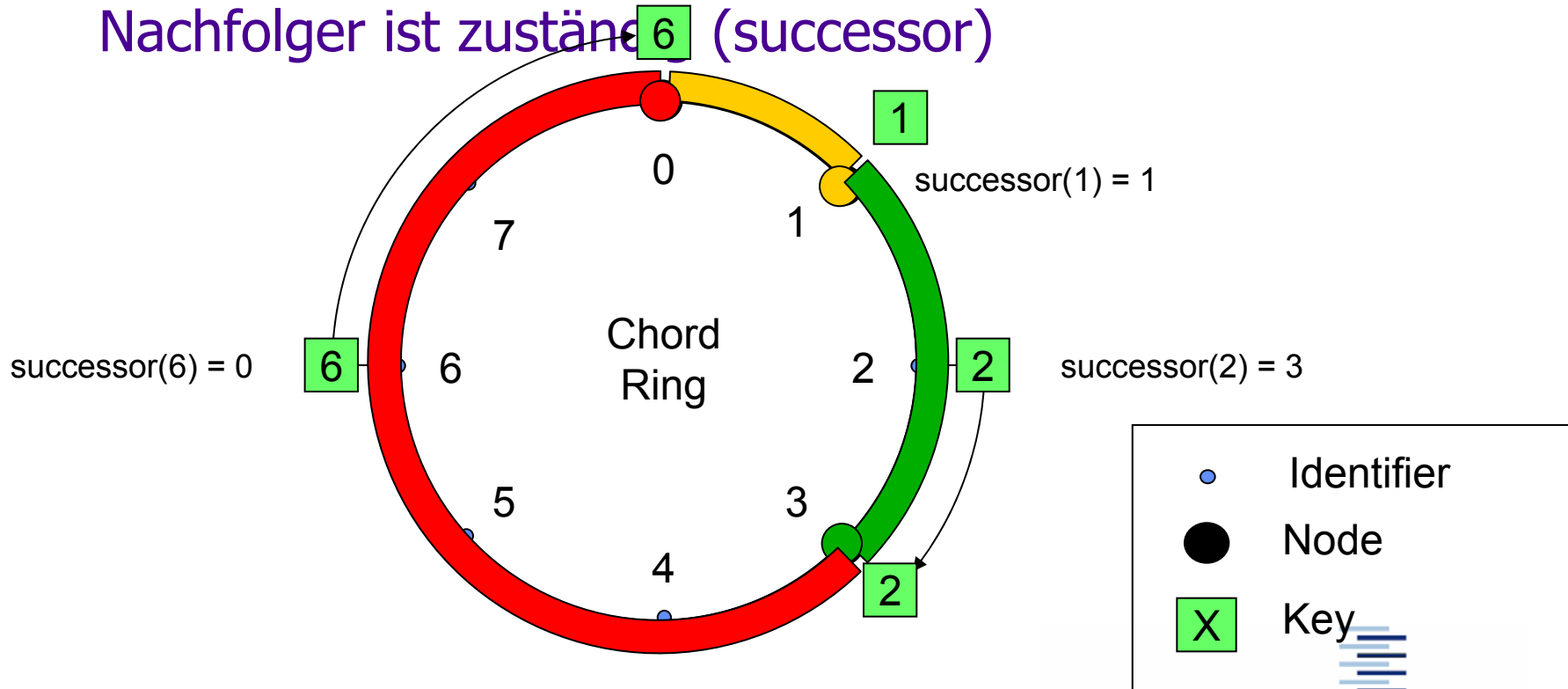
Chord: Topologie

- ▶ Hash-table Speicherung
 - ▶ put (key, value) fügt Daten in Chord ein
 - ▶ Value = get (key) ermittelt Daten aus Chord
- ▶ Identifizierer durch **consistent hashing**
 - ▶ Nutzt monotone, lastverteilende Hash Funktion
 - ▶ z.B. SHA-1, 160-bit output $\rightarrow 0 \leq \text{identifizierer} < 2^{160}$
 - ▶ *Key* assoziiert mit Datenwert
 - ▶ z.B. key = sha-1(value)
 - ▶ *ID* assoziiert mit Host
 - ▶ z.B. id = sha-1 (IP address, port)



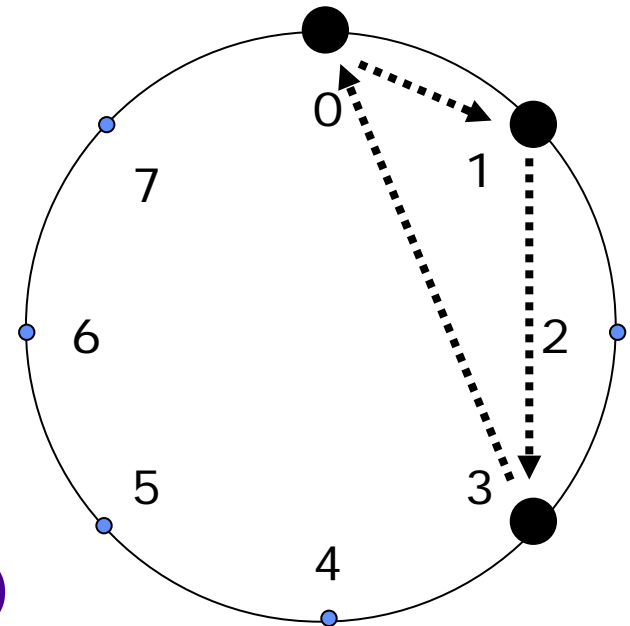
Chord: Topologie

- Keys und IDs auf Ring, d.h. Arithmetik modulo 2^{160}
- (key, value) Paare zugeordnet im Uhrzeigersinn:
Nachfolger ist zuständig (successor)



Chord: Topologie

- Topologie bestimmt durch Links zwischen Knoten
 - Link: Wissen um Nachbarknoten
 - Gespeichert in Routing Tabelle
- Einfachste Topologie: kreisförmig verlinkte Liste
 - Jeder Knoten hat Verweis auf nächsten Knoten (im Uhrzeigersinn)



Verbessertes Routing auf dem Ring ?

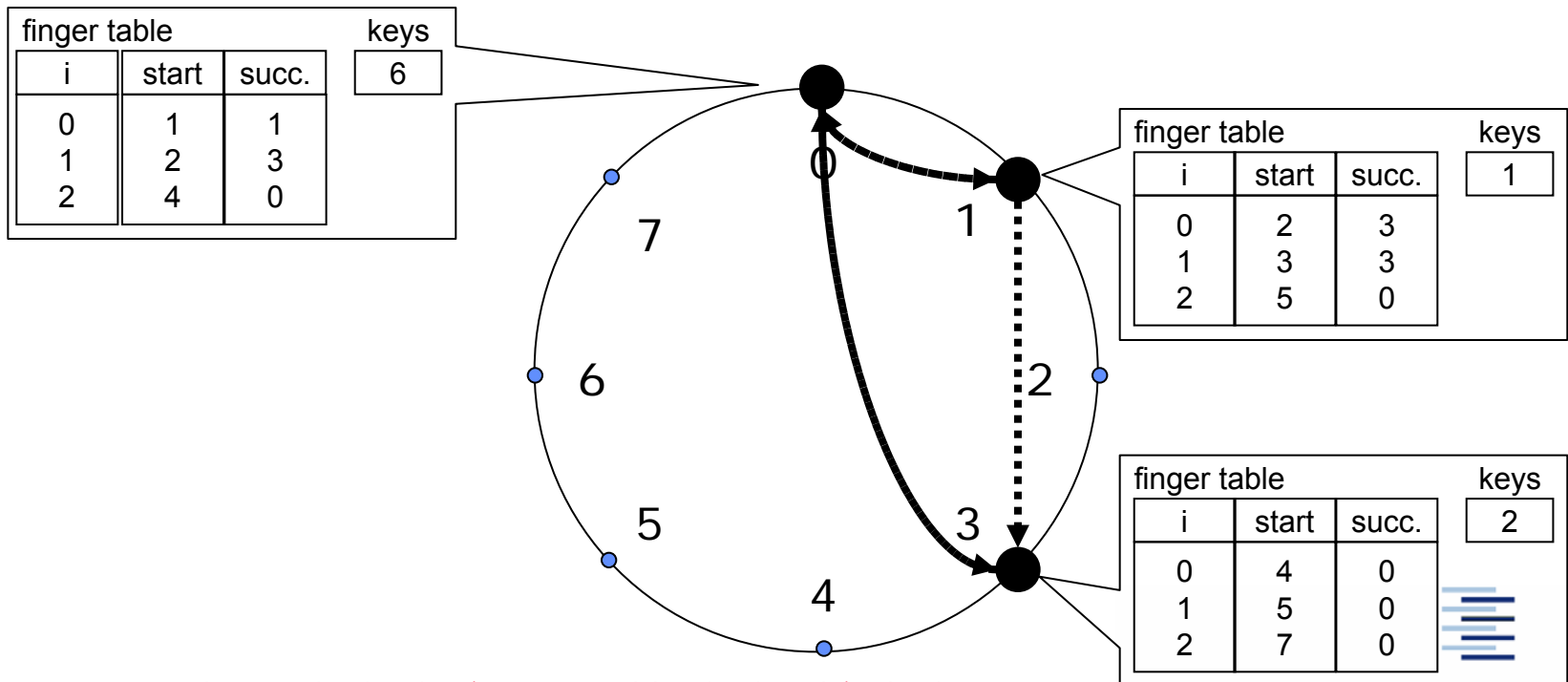
- ▶ **Erweiterte Routeninformationen:**
 - ▶ Speichere Links für z Nachbarn, leite Suche nach k zu weitesten bekannten Vorgänger von k
 - ▶ For $z = N$: vollständig vermaschtes Routing System
 - ▶ Lookup-Effizienz: $O(1)$
 - ▶ Knotenzustände: $O(N)$
 - ▶ Weiterhin geringe Skalierbarkeit durch lineare Zustandszahl
- ▶ **Skalierbares Routing:**
 - ▶ Mischung von nahen und fernen Links:
 - ▶ Akkurates Routing in Nachbarschaft des Knoten
 - ▶ Schneller Routing-Fortschritt über weite Distanzen
 - ▶ Beschränkte Anzahl von Links per Knoten



Chord: Routing

Chord's RoutingTabelle: *finger table*

- Speichert $\log(N)$ Links per Knoten
- Überdeckt exponentiell wachsende Distanzen:
 - Knoten n : Eintrag i zeigt auf $\text{successor}(n + 2^i)$ (*i-th finger*)

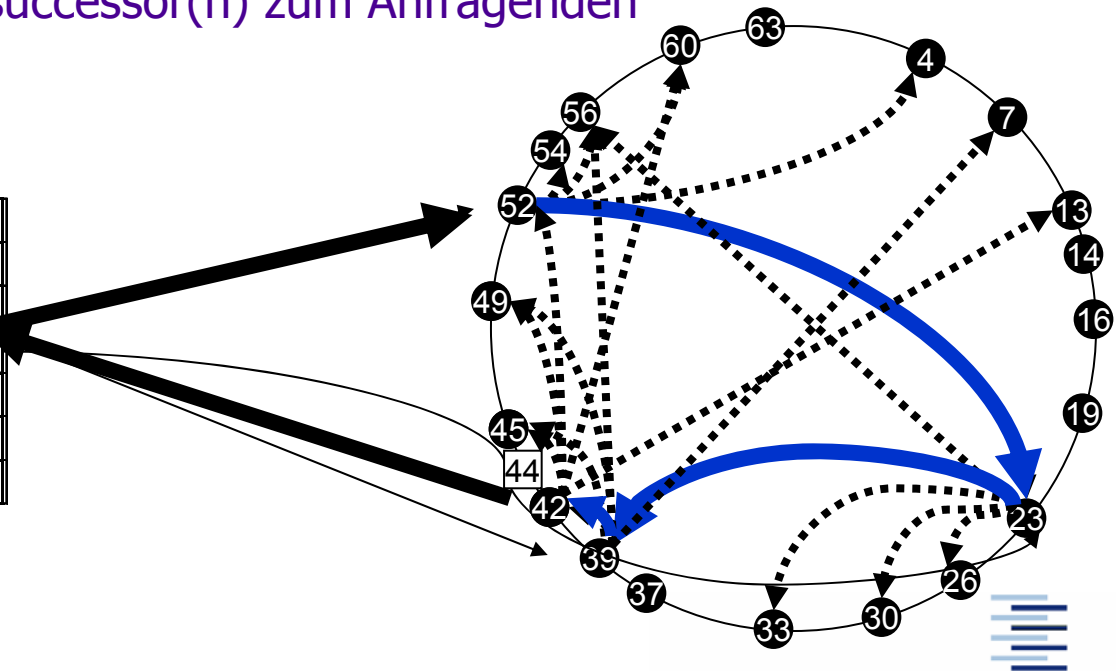


Chord: Routing

Chord's Routing Algorithmus:

- ▶ Jeder Knoten n leitet Suche für key k im Uhrzeigersinn
 - ▶ zum entferntesten finger vor k
 - ▶ bis $n = \text{predecessor}(k)$ und $\text{successor}(n) = \text{successor}(k)$
 - ▶ Rückgabe $\text{successor}(n)$ zum Anfragenden

i	2^i	Target	Link
0	1	1	2
1	2	3	4
2	4	5	6
3	8	7	9
4	16	11	13
5	32	17	19



Chord: Selbstorganisation

- ▶ Handhabung veränderlicher Netzwerkbedingungen
 - ▶ Knotenfehler
 - ▶ Netzwerkfehler
 - ▶ Ankunft neuer Knoten
 - ▶ Verschwinden bisheriger Knoten
- ▶ Erhalt konsistenter Systemzustände
 - ▶ Halte Routing Informationen aktuell
 - ▶ Routing Korrektheit liegt in den richtigen Nachfolgerinformationen
 - ▶ Routing Effizienz liegt in richtigen finger tables
 - ▶ Fehlertoleranz für alle Operationen erforderlich



Chord: Knotenbeitritt

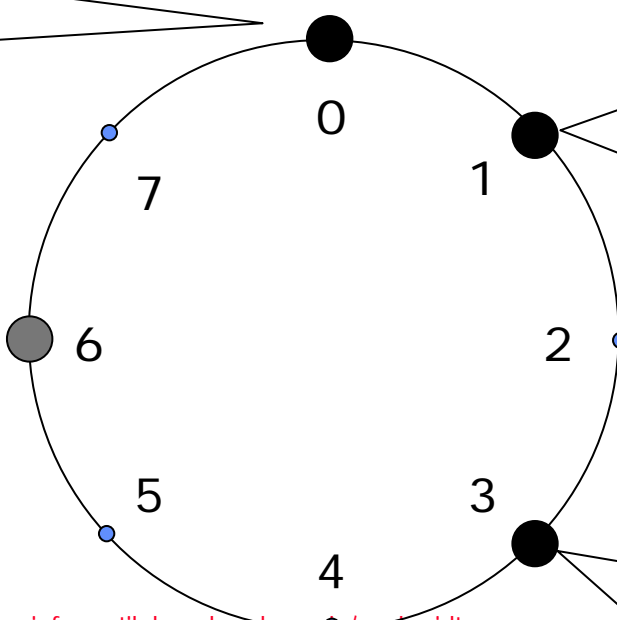
- Knoten bildet seine ID
- Kontaktiert bestehenden Chord Knoten (vom Bootstrapping)
- Sucht eigene ID und erfährt so seinen Nachfolger
- Erfragt (key, value) Paare vom Nachfolger
- Konstruiert finger table mittels Standard-Routing/lookup()

finger table			keys
i	start	succ.	6
0	1	1	
1	2	3	
2	4	0	

finger table			keys
i	start	succ.	1
0	2	3	
1	3	3	
2	5	0	

finger table			keys
i	start	succ.	6
0	7	0	
1	0	0	
2	2	3	

finger table			keys
i	start	succ.	2
0	4	0	
1	5	0	
2	7	0	



Chord: Zusammenfassung

► Komplexität

- Nachrichten per lookup: $O(\log N)$
- Speicherplatz per Knoten: $O(\log N)$
- Nachrichten pro Managementaktion (join/leave/fail): $O(\log^2 N)$

► Vorteile

- Theoretische Modelle und Beweise der Komplexität
- Einfach & flexibel

► Nachteile

- Keine Kenntnis der (Underlay-) Distanz
- Chord Ringe können in der Realität auseinander fallen

► Viele publizierte Verbesserungen

- z.B. Proximität, bi-directionale Links, Lastausgleich, etc.



References

- Andy Oram (ed.): *Peer-to-Peer*, O'Reilly, 2001.
- R. Steinmetz, K. Wehrle (eds.): *Peer-to-Peer Systems and Applications*, Springer LNCS 3485, 2005.
- C.Plaxton, R. Rajaraman, A. Richa: Accessing Nearby Copies of Replicated Objects in a Distributed Environment, Proc. of 9th ACM Sympos. on parallel Algor. and Arch. (SPAA), pp.311-330, June 1997.
- I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. Proc. of the 2001 ACM SigComm, pp. 149 – 160, ACM Press, 2001.

