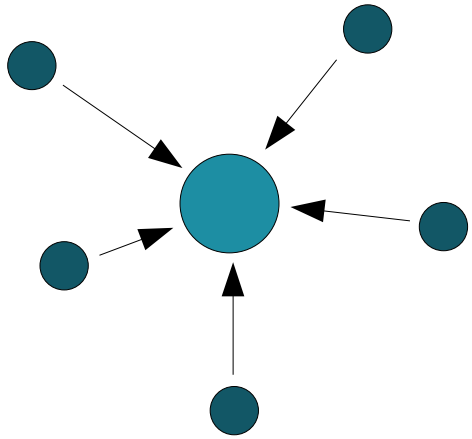
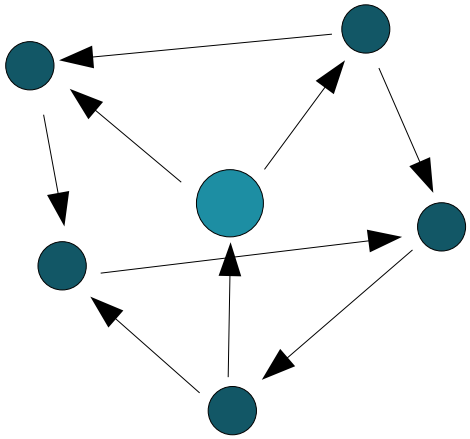
Peer-to-Peer-Technologie

Klient-Server-Architektur

- Bisher:
  - Viele Klienten fragen einen Dienst nach
  - Ein Server erbringt den Dienst
- Nahe liegend, denn
  - Alltägliche Vorstellung von einem Laden
  - Viele Kunden kommen dorthin um einzukaufen
- Leichteres Finden von Gütern/Diensten
  - Wohin muss ich mich wenden, um ein Sofa zu bekommen?
  - Sofa = Möbel; Möbel bekommt man im Möbelgeschäft
  - Anzahl der Möbelgeschäfte relativ klein / schnellere Suche



Neue Sichtweise



- Keine expliziten „Server“, von denen alle Klienten etwas herunterladen
- Jeder Teilnehmer im Netz ist sowohl Anbieter als auch Nachfrager
  - Peer = Kollege (kein Vorgesetzer, kein Untergebener)
- Verteilung der Lasten auf alle
  - Nutzung brach liegender Ressourcen
- Verteilung der Verantwortung (?)

Peer-to-Peer

- Einsatzgebiete
  - Verteilte Verarbeitung
    - Alle Teilnehmer berechnen einen Teil der Gesamtaufgabe, oder
    - verschiedene Teilnehmer bieten (ggf. gleichwertige) Dienste an
    - Erhöhte Ausfallsicherheit
  - Persistenz
    - Daten werden an unterschiedlichen Stellen im Netz gehalten
    - Verteilung der Downloadlasten
    - Sicherheit vor Datenverlust

P2P-Fragestellungen

- Wichtige Fragen
  - Existiert eine bestimmte Datei im Netz?
  - Wer kann diese Datei liefern?
  - Was passiert, wenn der bisherige Lieferant vom Netz geht?
  - Wie lokalisiere ich einen Anbieter?
  - Wie verhindert man eine Flutung des Netzes mit Anfragen?
  - Ist eine zentrale Verwaltung oder Indizierung notwendig?
  - Ist das ein zentraler Ausfallpunkt?

P2P-Fragestellungen

- Wie schnell ist die Datenübertragung?
- Wie erreicht man Fairness?
- Kann man eine Datei auch von mehreren Quellen erhalten und damit die Bandbreite besser ausnutzen?
- Wie integrieren sich neue Teilnehmer?

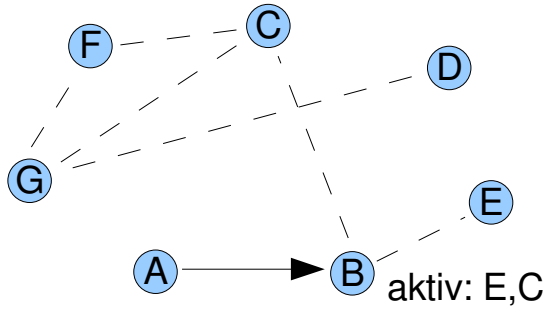
Betrachtungen

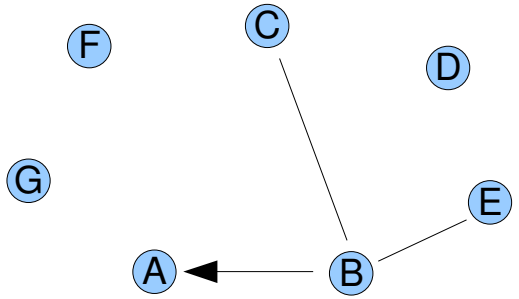
- Grundlegende Technologien
- Technologien und Rahmenwerke zum Aufbau von P2P-Netzen
  - JXTA
  - Pastry
- Netze und Anwendungen
  - insbesondere Tauschbörsen

Gnutella

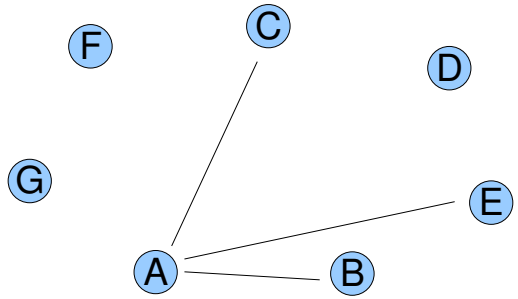
- Entwickelt von Justin Frankel/Tom Pepper (Nullsoft)
  - AOL kauft Nullsoft und stoppt Projekt aus rechtlichen Bedenken
  - von unabhängigen Entwicklern analysiert und weiterentwickelt
- Vollständig dezentrales Netzwerk
  - große Stabilität, keine Ausfälle
  - lange Suchzeiten
  - verhindert rechtliche Angriffe gegen einen zentralen Betreiber
- „Query flooding“
  - Abfrage an viele Punkte des Netzes

Gnutella

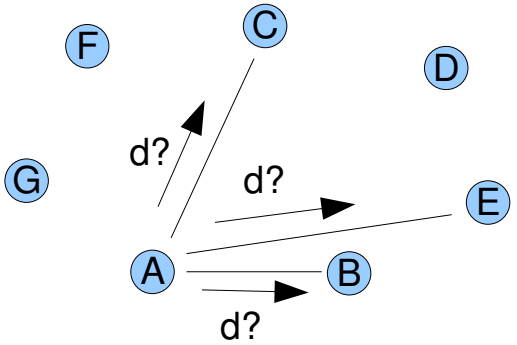


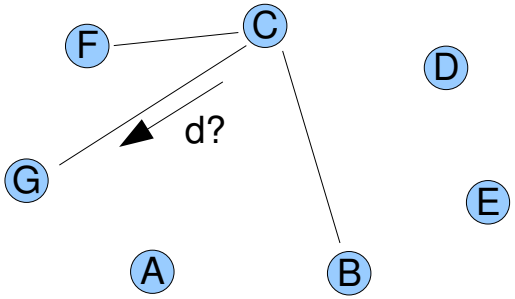


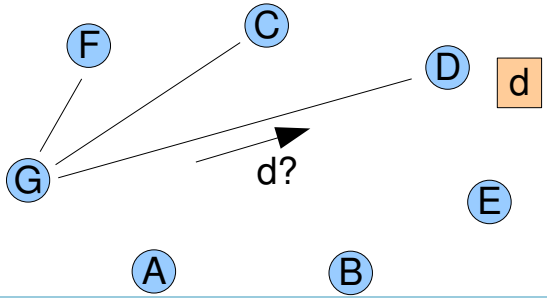
(aus Sicht von B)



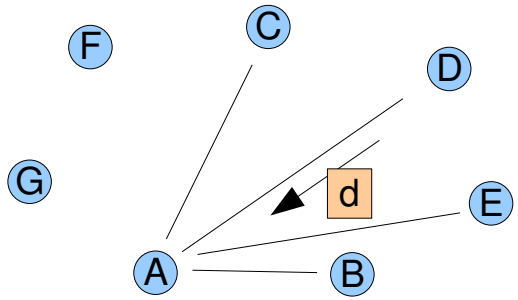
(aus Sicht von A)

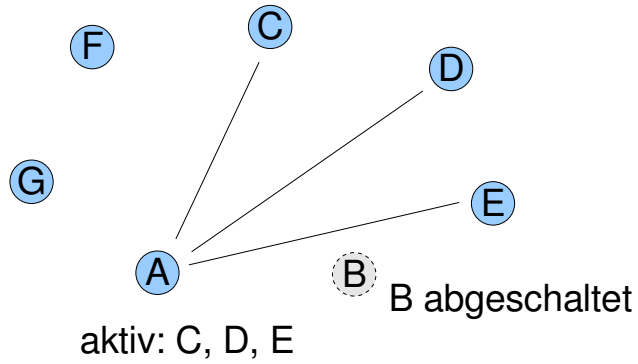






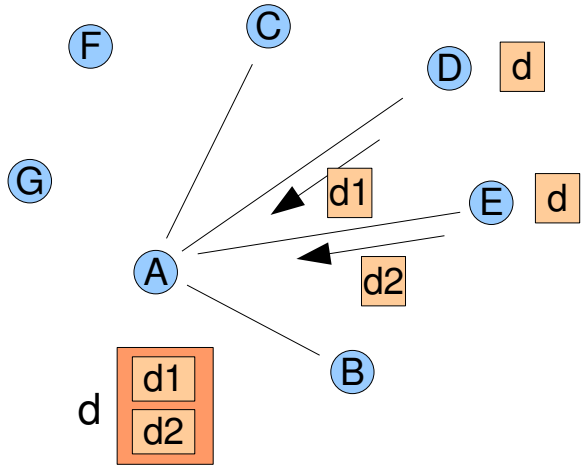
Gnutella





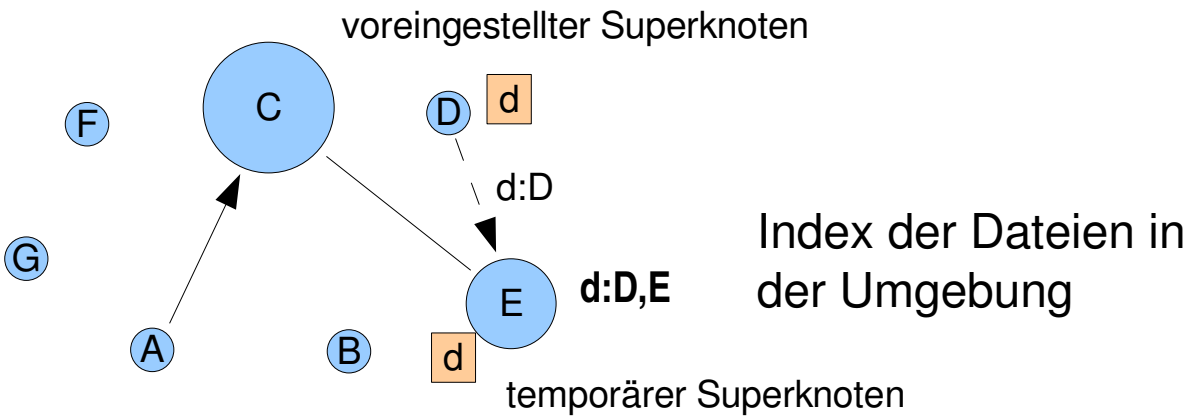
Übertragung „scheibchenweise“

- wenn mehrere Knoten dieselbe Datei besitzen



FastTrack

- Protokoll der zweiten Generation
  - basiert auf Gnutella
  - spezialisiert auf den Austausch von mp3-Dateien
  - nicht völlig dezentral
- Speziell: Superknoten (*Supernodes)*



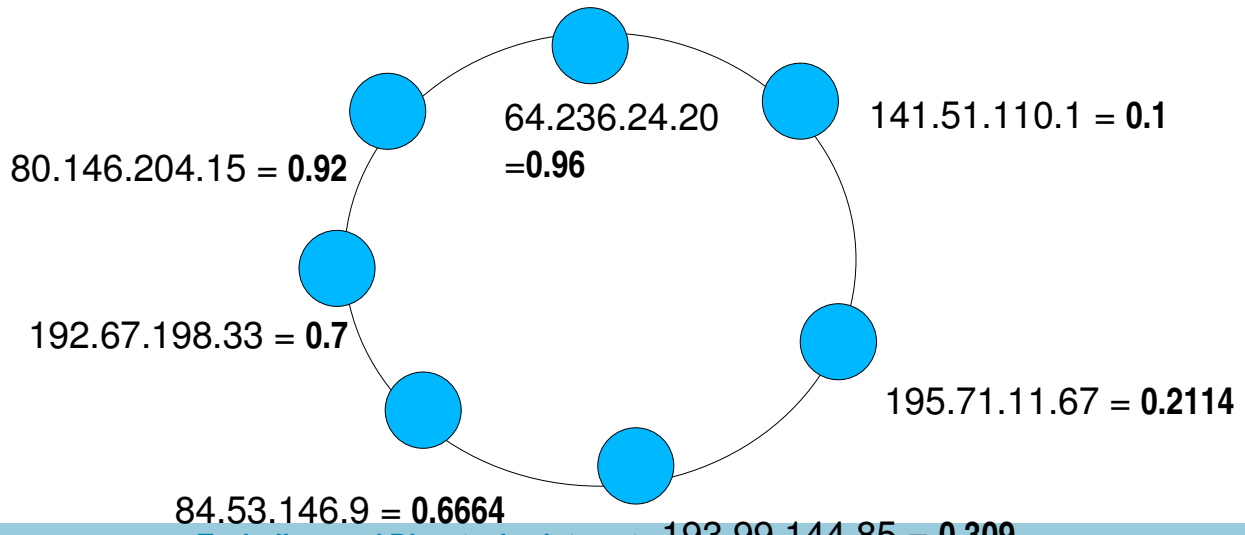
FastTrack

- Superknoten
  - jeder kann Superknoten werden (abhängig von Verbindung und Rechnergeschwindigkeit)
  - Temporäre Superknoten melden sich bei den voreingestellten Superknoten
  - Knoten melden ihre Dateilisten an den nächsten Superknoten
  - Klienten starten ihre Suchanfragen an ihrem Superknoten
- Datenübertragung via HTTP
- Identifikation der Dateien über Hashwert (UUHash)
  - verwendet mehrere kleine Blöcke, nicht die ganze Datei
  - ist anfällig für „Sabotage“ (defekte Dateien in Umlauf bringen, deren Hashwert gleich ist)

Verteilte Hashtabellen

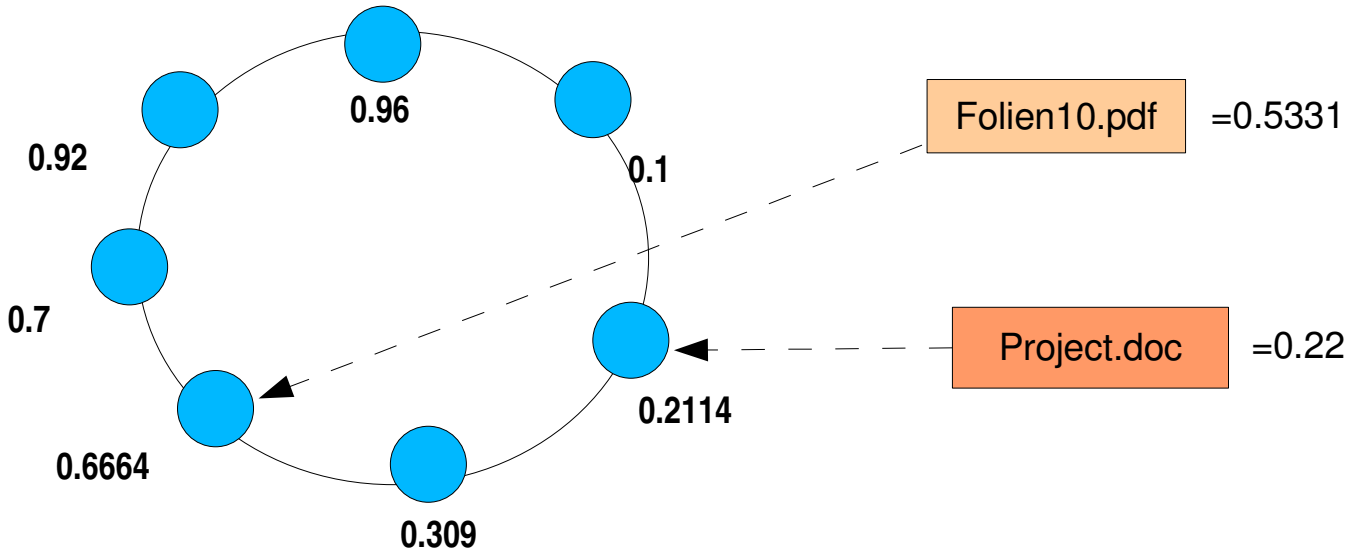
- Information wird im Netz verteilt
- Zugriff über eine Hashfunktion
- Knoten sind in einem abstrakten Netz organisiert
  - z.B. Hyperkubus, Ring
  - Adressen z.B. im (realen) Intervall [0, 1]

Beispielhafte Zuordnung



Verteilte Hashtabellen

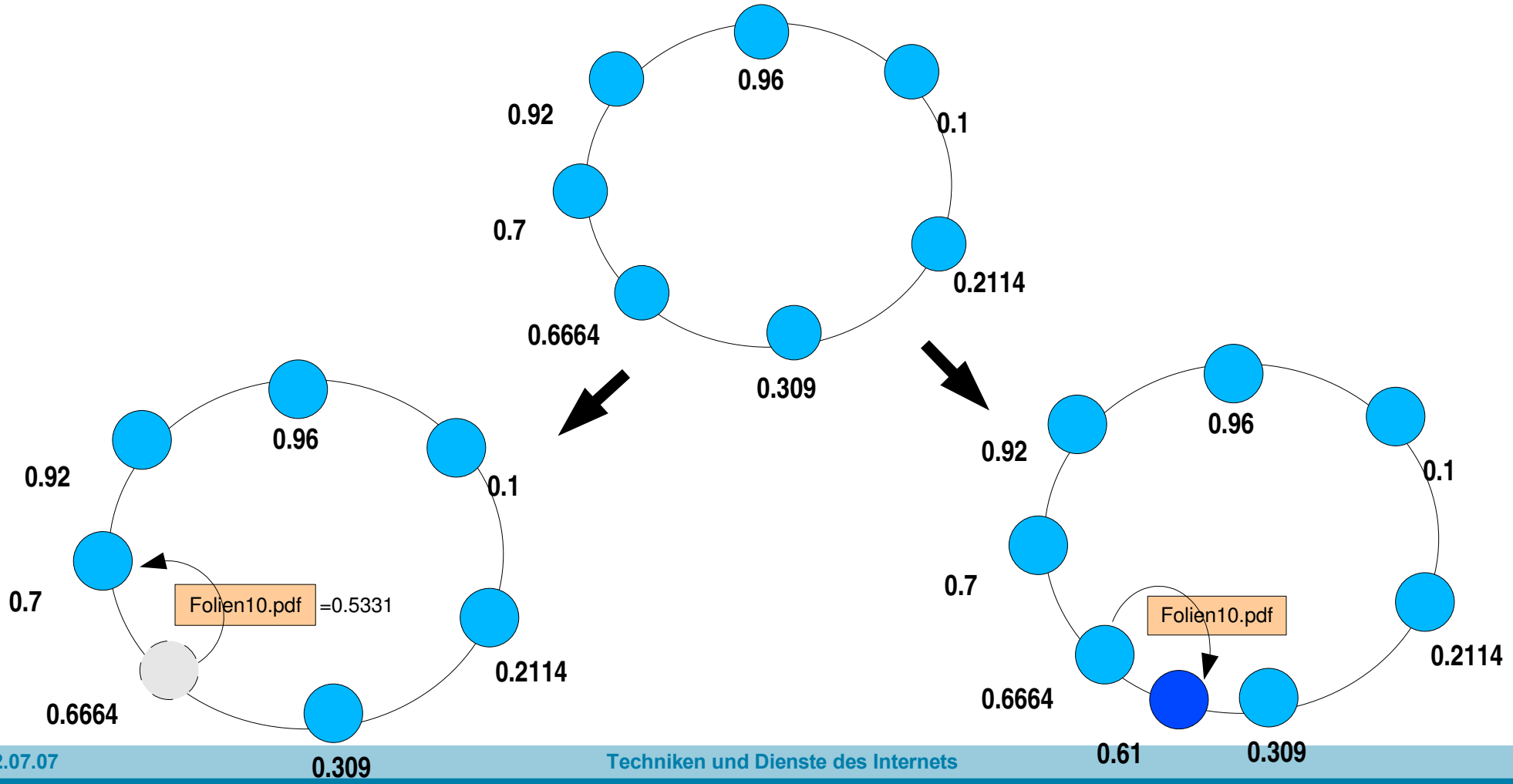
- Daten bekommen ebenfalls einen Schlüssel aus demselben Bereich
- Daten werden am nächstgelegenen Knoten gespeichert



Schlüsselbasiertes Routing

Verteilte Hashtabellen

- Beitritt und Austritt



Verteilte Hashtabellen

- Problem
  - Wie findet man die IP-Adresse zu einem gegebenen Schlüssel (um die Daten zu finden)?
    - aus dem Hash nicht mehr rekonstruierbar (keine Umkehrabbildung!)
- Ansatz
  - Knoten kennen jeweils ihre Nachbarn
    - links/rechts im Kreis
    - entlang der Kanten in einem n-dimensionalen Würfel
  - Annäherung über mehrere Stufen

Chord

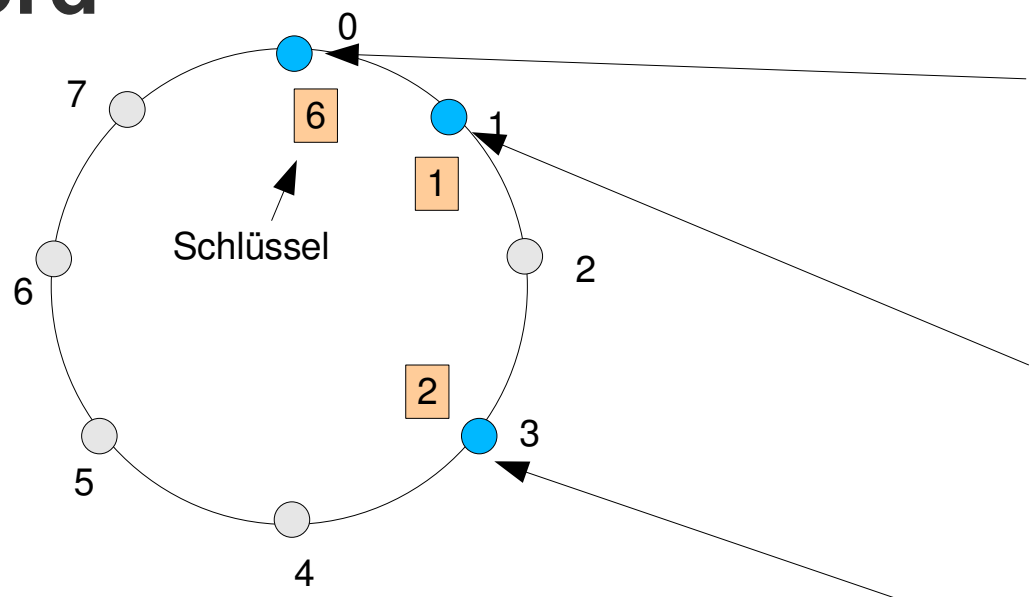
- Entwicklung am MIT
- Eigenschaften
  - Ringstruktur
  - Werteraum ganzzahlig, $[0..2^m)$; Identifikatoren mit Modul 2^m
  - Daten werden immer dem zugehörigen Knoten oder dem Nachfolger zugewiesen
  - Verlässt Knoten n den Ring, werden alle seine Daten auf seinen Nachfolger verschoben
  - Betritt Knoten n den Ring, werden alle Daten von seinem Nachfolger auf ihn verschoben (außer, sie treffen exakt den Nachfolger)

Chord

- Problem, wenn jeder Knoten nur seinen Nachfolger kennt
  - wäre in ungünstigen Fällen ineffizient, könnte den Durchlauf aller Knoten bewirken
- In Chord hat jeder Knoten Zusatzinformationen
  - Zeigetabelle (finger table) mit m Einträgen (bei max. 2^m Knoten)
  - k,n: Knotenindizes
  - k: aktueller Knoten

Zeile 1 -> n = Nachfolger(k+1)
Zeile 2 -> n = Nachfolger(k+2)
Zeile 3 -> n = Nachfolger(k+4)
Zeile 4 -> n = Nachfolger(k+8)
...
Zeile m -> n = Nachfolger($k+2^{m-1}$)

Chord



Start	Intervall	Nachf.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	0

Start	Intervall	Nachf.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0

Start	Intervall	Nachf.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

Knoten 3 sucht Nachfolger des Schlüssels „1“

- 1 gehört ins Intervall [7,3) -> dritter Eintrag in Tabelle -> Nachfolger 0
- Knoten 3 bittet Knoten 0 um Adresse von „1“
- Knoten 0: 1 gehört ins Intervall [1,2) -> erster Eintrag -> Nachfolger 1 -> gefunden (exakter Treffer)

Chord

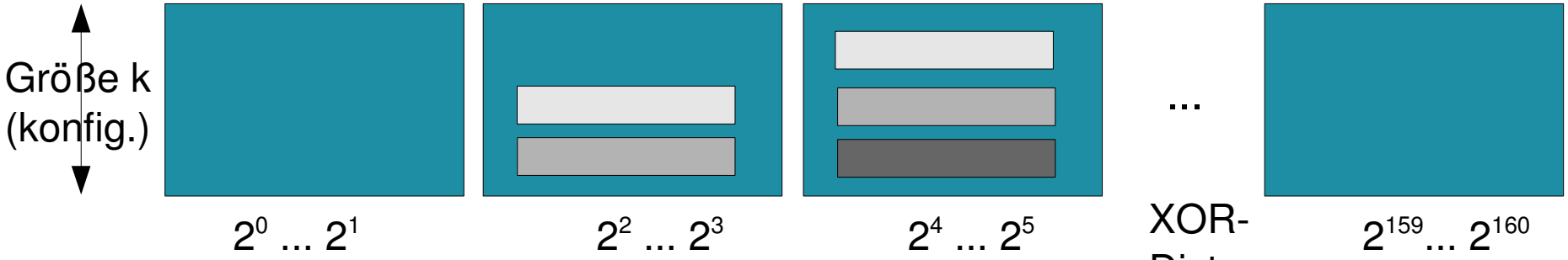
- Abstand zum gesuchten Knoten wird fortgesetzt halbiert
- Die Anzahl der abzufragenden Knoten ist O(logN) in einem N-Knoten-Netz
  - unter Standardannahmen (Gleichverteiltung, keine gezielte Wertewahl)
- Das Hinzufügen oder Wegnehmen von Knoten in einem Chord-Netz erfordert $O(\log^2 N)$ Nachrichtentransfers
  - Schlüsselumordnung und Zeigetabellen-Aktualisierung
  - Vereinfachung durch Vorgängerzeiger

Kademlia

- P2P-Abstraktionsprotokoll (overlay)
  - Definition eines Netzwerks
  - Kommunikation über UDP
  - Verwendung einer verteilten Hashtabelle
- Jeder Knoten bekommt eine Knoten-ID (node ID), 160 Bit
- Wesentliche Stärke
  - Distanzberechnung über XOR-Funktion (Ergebnis als Zahl interpretiert)
  - XOR ist Metrik
    - Zeige: xor(x,x)=0; xor(a,b)=xor(b,a); xor(x,z)<=xor(x,y) + xor(y,z)
  - kann Routinginformationen lernen
  - Parallele Anfragen

Kademlia

- Kennenlernen neuer Peers als Tripel (IP, Port, ID)
- Einsortieren der Tripel in Töpfe für Intervalle



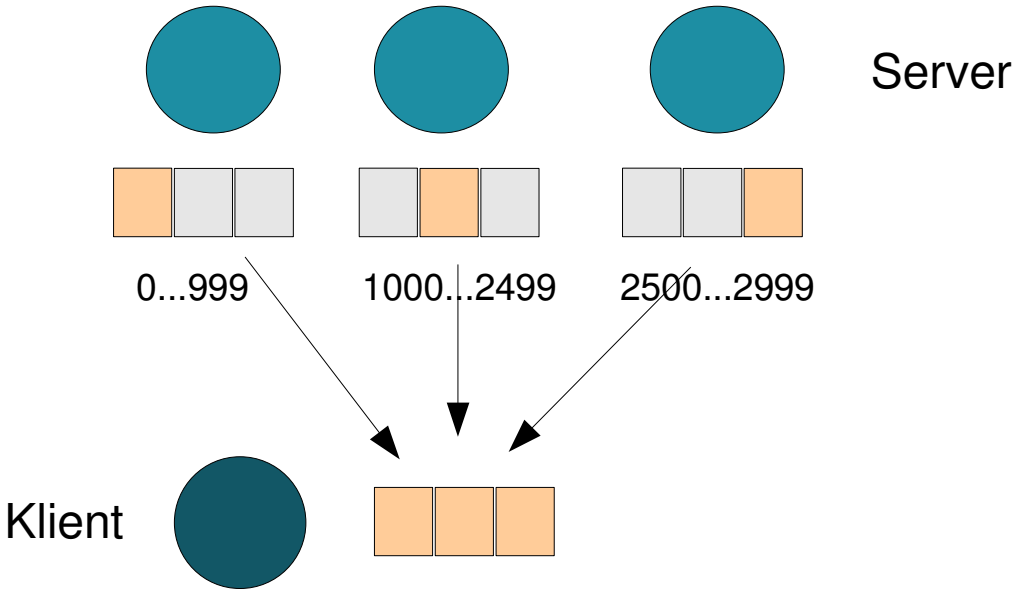
- Sortieren der Tripel
  - kürzlich gesehen: Kopf
  - am längsten nicht gesehen: Ende
- wenn Topf voll: Anpingen des Kontakts am Ende
  - wenn er sich meldet, an den Kopf stellen, sonst verwerfen

Kademlia-Protokoll

- find_node(ID)
  - Anfrager kann parallele Anfragen starten: nimmt ein paar der Werte in eigenem „ID-nahen“ Topf als Empfänger
  - Empfänger liefert bis zu k Tripel zurück, die aus dem zugehörigen Intervall-Topf stammen
- find_value(ID)
  - wie find_node, es sei denn, der Wert liegt vor: dann den Wert liefern
- Entdeckt ein Knoten einen für einen seiner gespeicherten Werte „geeigneteren“ Knoten, dann kopiert er den Wert dorthin (store)
  - nicht verschieben - aus Persistenzgründen

MFTP

- Multisource File Transfer Protocol
  - patentiert in den USA: Patentnummer 6 339 785
  - Klient kann von mehreren Servern Teile der Datei herunterladen und sie dann zusammensetzen
  - Server können bevorzugt ausgewählt werden



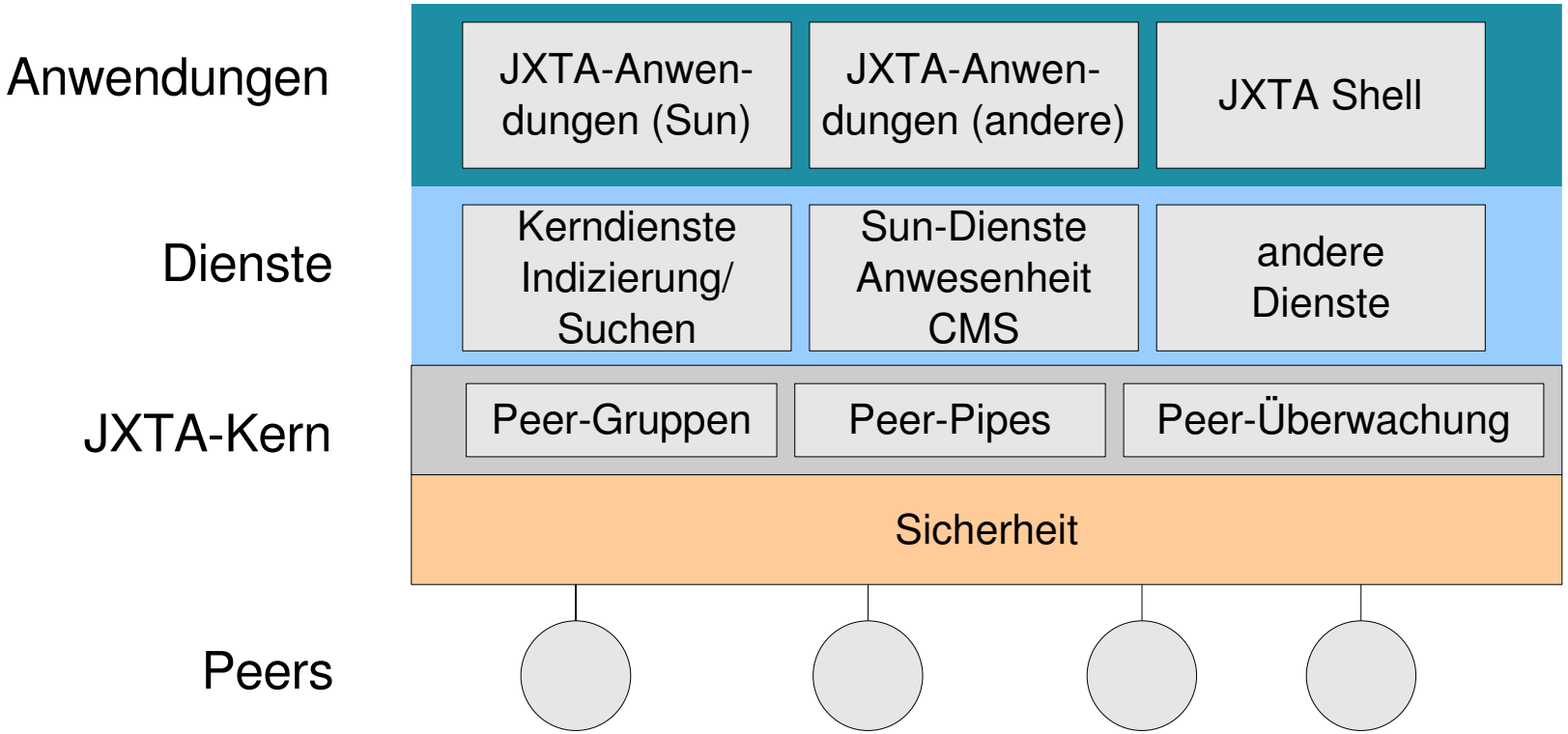
Eigentlich keine wirklich neue Erfindung: Das Laden ab Byte n ist im FTP-Standard bereits festgelegt (REST); das Laden kann ggf. mit ABOR abgebrochen werden

MFTP

- zusätzliche Beurteilung möglich (als Metadaten, z.B. „lohnt sich“, „ist kaputt“)
- Dateien können über Hashwert (MD4) referenziert werden
  - Dateinamen können sich unterscheiden
  - Hashwert hängt vom Inhalt der Datei ab
- MFTP ist Basis für ed2k-Protokoll (eDonkey2000)

JXTA

- P2P-Entwicklungsumgebung (Rahmenwerk) in Java
- Schwerpunkt liegt auf Anwendungserstellung auf Basis von P2P-Vernetzung

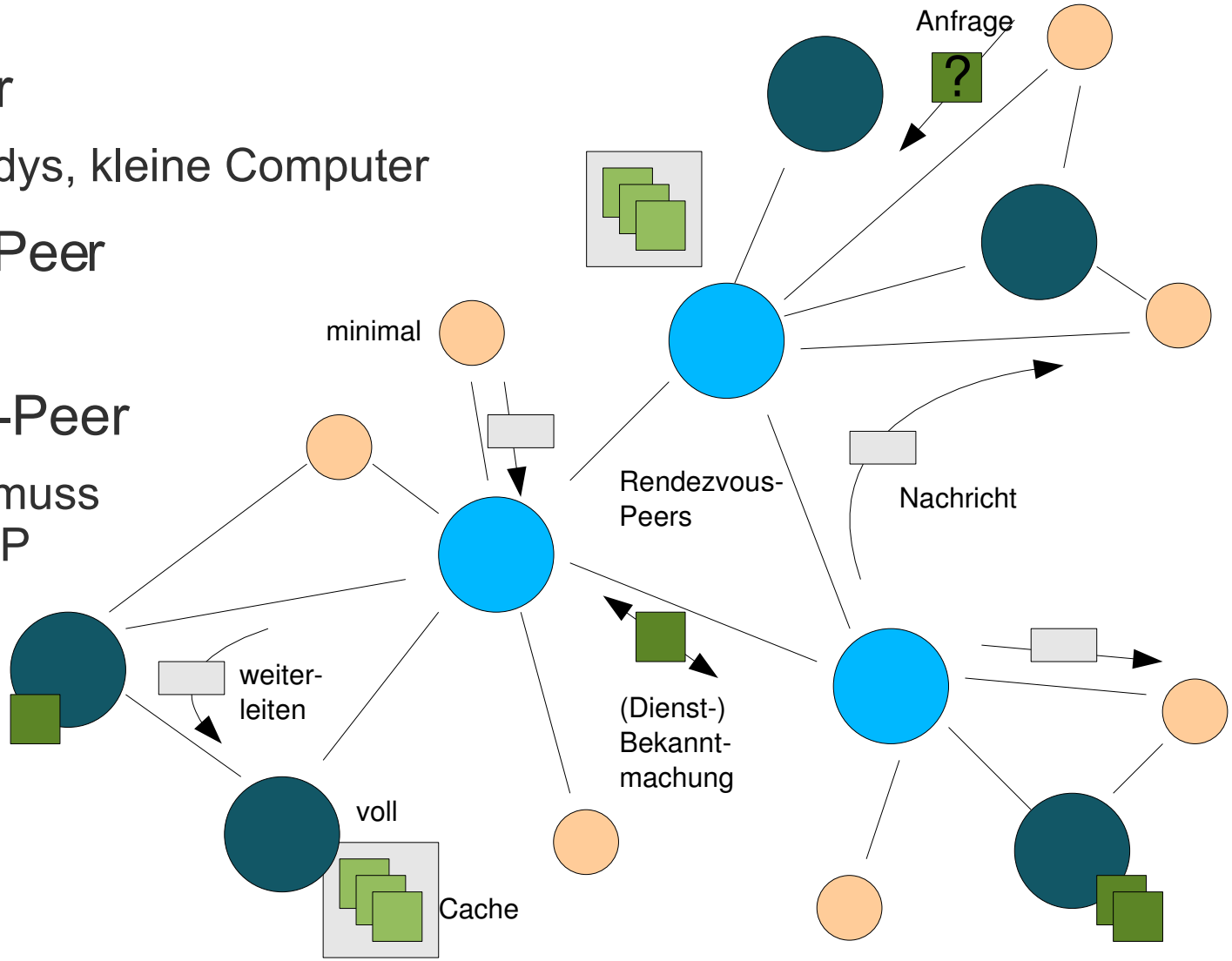


JXTA

- Peer
  - jedes Gerät im Netz, das eines oder mehrere JXTA-Protokolle implementiert
- Peer-Gruppen
  - Sammlung von Peers mit gemeinsamen Satz von Diensten
  - Peers organisieren sich selbständig in Gruppen
  - Peers können mehreren Gruppen angehören

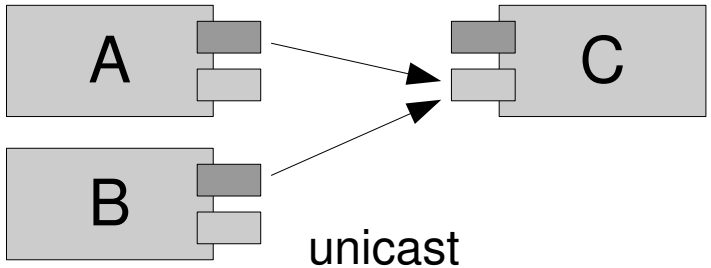
Peer-Typen

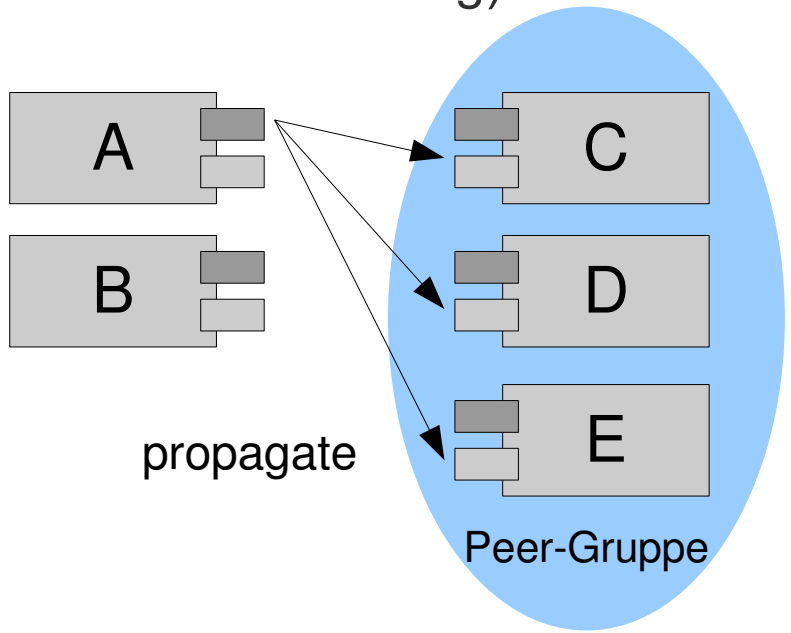
- Minimal-Peer
  - PDAs, Handys, kleine Computer
- Vollwertiger Peer
  - Normalfall
- Rendezvous-Peer
  - jeder Peer muss einen RendP kennen



Pipes

- Pipes
  - asynchrone, unidirektionale Verbindungen zwischen Peers
  - Unicast (Point-to-Point) Pipes und Propagate Pipes
  - Secure Unicast Pipes (sichere und verlässliche Verbindung)





- Als Erweiterung auf Pipes basierend
  - JxtaBiDiPipes, JxtaSocket

Gruppendienste

- Kern-Gruppendienste
  - Discovery (entdecken)
  - Membership (Mitgliedschaft)
  - Access (Gültigkeit des Zugriffs)
  - Pipe (Erzeugen und Verwalten von Pipes)
  - Resolver (Allgemeine Anfragen an andere Peers senden)
  - Monitoring (Überwachung)
- Implementierungen müssen nicht alle Dienste bereitstellen

Bekanntmachung

- Bekanntmachung (Advertisements)
  - Peer Advertisement
  - Peer Group Advertisement
  - Pipe Advertisement
  - Rendezvous Advertisement
  - usw.
- In XML formuliert
- Von Rendezvous-Peers weitergeleitet

Bekanntmachung

- Beispiel für eine Pipe-Bekanntmachung

```
<?xml version="1.0"?>
<!DOCTYPE jxta:PipeAdvertisement>

<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
   <Id>urn:jxta:uuid-716383ab49ec018fd00183....</Id>
   <Type>JxtaUnicast</Type>
   <Name>TestPipe.end1</Name>
</jxta:PipeAdvertisement>
```

Shared Resource Distributed Index

- SRDI, Suchindex-Dienst für Advertisements
- Peers senden einen Index für ihre Advertisements zu den Rendezvous-Peers über den SRDI
- Anfragen werden nur zwischen den Rendezvous-Peers ausgetauscht
- Aktualisierung der bekannten und aktiven Rendezvous-Peers
  - lose-konsistentes Netzwerk
- Weiterleitung der Indizes über VHT-Technik

JXTA

- Protokolle
  - Peer Discovery
  - Peer Information
  - Peer Resolver
  - Pipe Binding
  - Endpoint Routing
  - Rendezvous
- Komplette API

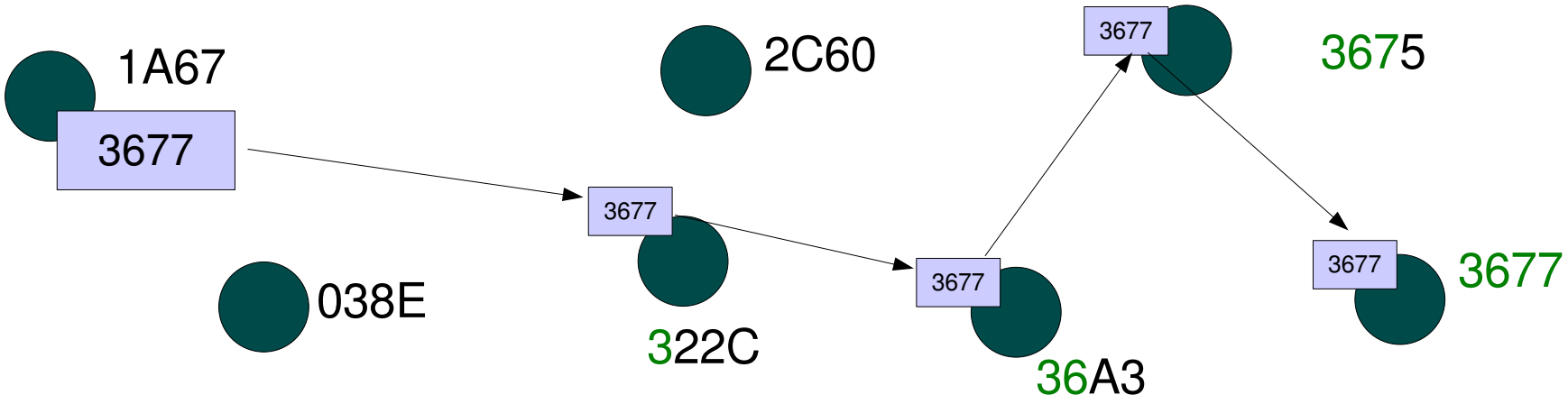
Weitere Informationen unter www.jxta.org oder www.sun.com/software/jxta/

Pastry

- P2P-Infrastruktur von Microsoft
  - „Dezentralisiertes Objektlokalisierungs- und Routingsystem für große P2P-Systeme“
- Grundlagen
  - Knoten, jeder mit einer 128 Bit langen ID (nodeID, wird zufällig ausgewählt)
    - zweckmäßig aus dem Hashwert des eigenen öffentlichen Schlüssels oder der IP-Adresse
  - kann die Lieferung der Daten zum Ziel garantieren, solange eine gewisse Menge von Knoten aktiv bleiben

Pastry: Grundlagen

- Routing nutzt Ziffern des Schlüssels
  - versucht, beim nächsten Hop die Kette übereinstimmender Ziffern der nodeIDs zu verlängern
  - kein solcher Knoten bekannt, dann zumindest jener mit kleinerem arithmetischem Abstand
  - Suchzeit O(log N) mit N=Anzahl der Knoten



Knotentabelle

- Dreiteilige Tabelle
  - Routing (Menge R)
    - für jede Präfixlänge eine Zeile
    - bekannte Knoten-IDs mit entsprechender Übereinstimmung im Präfix
    - Basis 2b (meist b=4, also hex; ergibt 32 Zeilen)
  - Nachbarschaft (Menge M)
    - nächstgelegene Knoten (in einer zusätzlich gegebenen Metrik); normalerweise nicht für Routing benutzt
  - Blätter (Menge L)
    - zur Hälfte mit numerisch nächstgelegenen größeren Knoten-IDs, zur Hälfte mit kleineren; wird für Routing benötigt

Nachbarschaft

- Zusätzliche Eigenschaft von Pastry: Nachbarschaft
  - eigene Metrik, gibt Auskunft über die Nähe zu anderen Knoten
  - z.B. gewonnen aus
    - Subnetznummer
    - geografischer Abstand
    - Hops
  - übliche Bedingungen
    - reflexiv (d(A,A)=0), symmetrisch (d(A,B)=d(B,A)), Dreiecksungleichung (d(A,B) <= d(A,C) + d(C,B))
- Annahme: „nahe“ Rechner sind in derselben Organisation

Knotentabelle

- Beispieltabelle (für b=2 und 16-Bit-IDs)

Basis 2^2 -> Ziffern 0,1,2,3

$|L| = 2^b$ oder 2^{b+1}
(Konfiguration)

grün: übereinstimmende Ziffern
rot: erste unterschiedliche Ziffer

$|M| = 2^b$ oder 2^{b+1}
(Konfiguration)

Knoten-ID: 10233102

Blätter

kleiner als diese ID		größer als diese ID	
10233000	10233001	10233120	10233122
10233021	10233033	10233230	10233232

Routing

02212102	1	22301203	31203203
0	11301233	12230203	13021022
10031203	10132102	2	10323302
10200230	10211302	10222302	3
10230322	10231000	10232121	3
10233001	1	10233232	
0		10233120	
		2	

Nachbarschaft

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Routing

- Eigene Knoten-ID=A, gesucht: passender Knoten für Nachricht mit Schlüssel D
  - Liegt D im Bereich von L? Dann Knoten von L mit geringstem Abstand auswählen.
  - Wenn nicht, Routingtabelle nutzen. Zeile gemäß gemeinsamem Präfix wählen, Spalte gemäß folgender Ziffer von D.
  - Wenn dort kein Knoten vermerkt ist, Knoten aus L, R oder M wählen, der numerisch am nächsten liegt (bei mind. gleich langem Präfix)
  - Nachricht dorthin schicken
- Nachricht wird zu einem Knoten geleitet,
  - dessen ID in mind. einer Ziffer mehr mit D übereinstimmt oder
  - der zumindest näher liegt

Hinzufügen von Knoten

- Aufgaben
  - Eigene Knotentabelle muss generiert werden
  - Andere Knotentabellen müssen aktualisiert werden
- Neuer Knoten heiße X
  - X „kennt“ bereits A in seiner Umgebung (gemäß der Nachbarschaftsmetrik)
- X bittet A, eine Nachricht („join“) an X zu schicken
  - Nachricht wird bei Knoten Z eintreffen, der X am nächsten liegt

Hinzufügen von Knoten

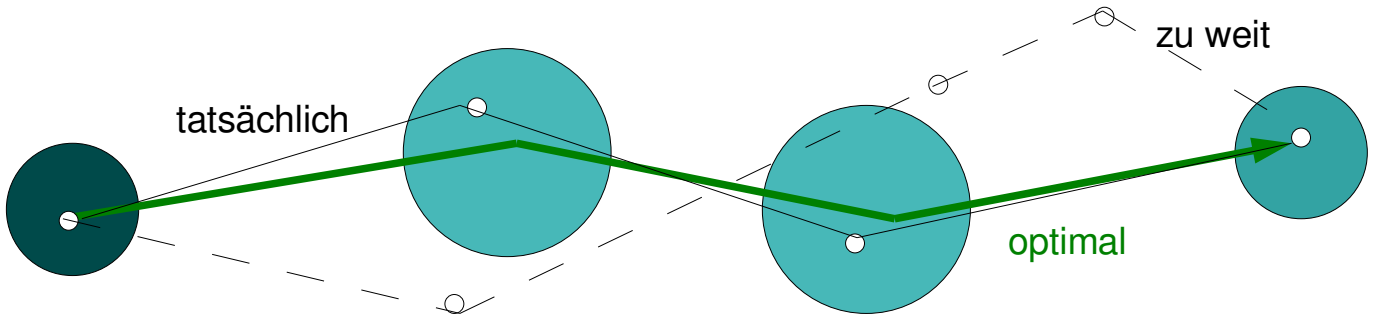
- Erstellung der Knotentabelle für X
  - Alle Knoten auf dem Weg schicken ihre Knotentabellen an X
  - X übernimmt M von A (denn: A liegt nach Voraussetzung in der Umgebung)
  - X übernimmt L von Z (denn: ID liegt nahe bei X)
  - X übernimmt jeweils geeignete Zeile für R von Knotentabellen der Knoten auf dem Weg zu Z (denn: schrittweise längerer Präfix)
  - X schickt seine neue Tabelle an alle Knoten in L, R, M

Entfernen von Knoten

- Situation: Knoten meldet sich nicht mehr
- Nachbar fordert bei anderen Knoten in der Menge L deren Blättermenge L an
  - füllt damit eigene Menge L wieder auf
- Fehler in Routingtabelle
  - Knoten in derselben Zeile nach passendem Eintrag für die Spalte fragen; ggf. nächste Zeile noch nutzen
- Nachbarschaftsmenge
  - periodisches Abfragen
  - andere Knoten in Nachbarschaft um Zusendung deren Menge M bitten; entsprechend auffüllen

Lokalität

- Idee der Lokalität durch die Nachbarschaftsmetrik
  - bekannte Knoten sind dem aktuellen Knoten so nahe wie möglich
- Knoteneinfügeverfahren bewahrt Lokalität
- Ergebnis
  - Präfix wird mit jedem Schritt verlängert, zugleich
  - wird die Nachricht in jedem Schritt an einen (für den jeweiligen Schritt!) möglichst nahen Knoten geschickt
  - Gesamtstrecke minimiert sich



Pastry

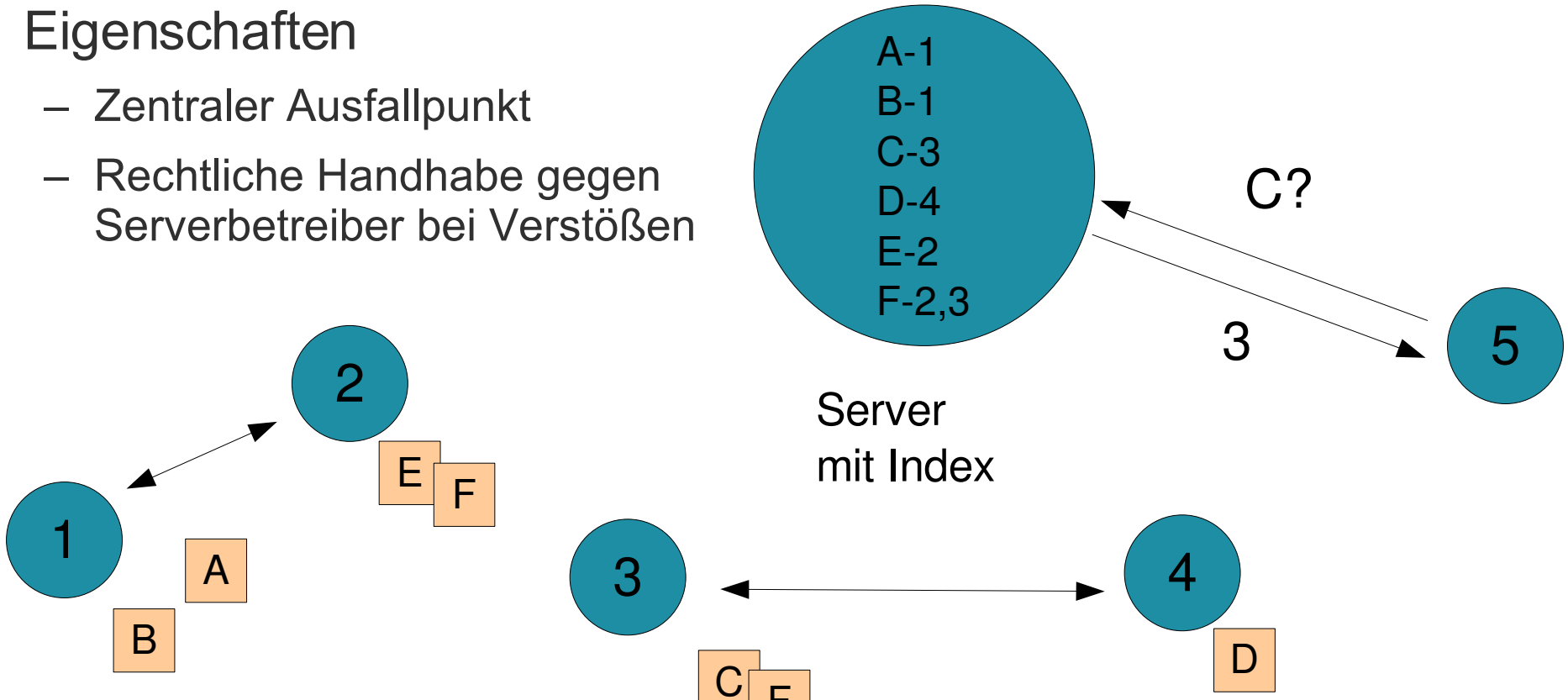
- Anwendungen
  - PAST
    - fileID als Hash aus Dateiname und Eigner einer Datei
    - repliziert Dateien auf den k Knoten-IDs, die der fileID am nächsten liegen
  - SCRIBE
    - publish/subscribe (loses Kommunikationssystem)
    - TopicIDs als Pastry-Schlüssel; werden auf Knoten mit naher nodeID gespeichert

P2P-Software

- Erste Generation: Zentrale Dateiliste
  - Napster
- Zweite Generation: Dezentrale Indizierung, VHT
  - Gnutella-/FastTrack-/Kademlia-Netze
- Dritte Generation: Anonymisierung
  - Freenet
  - Entropy

Napster

- Einsatz: Musikdateienbörse
- Zentrale Vermittlung
- Eigenschaften
  - Zentraler Ausfallpunkt
  - Rechtliche Handhabe gegen Serverbetreiber bei Verstößen



KaZaA

- Protokoll basiert auf FastTrack-Protokoll
  - proprietär, AES-256-Verschlüsselung
- „Teilnahmestufe“ als Anreiz, Dateien zum Tausch anzubieten
  - Download senkt, Upload erhöht die Stufe
  - jedoch ausgehebelt durch „schwindelnde“ inoffizielle Klientenimplementierungen
- Kritikpunkt
  - Primär zum unlizenzierten Tausch von Musikdaten genutzt
  - Installiert „Spyware“ und „Adware“
    - dient dazu, den Anwender mit Werbebotschaften zu versorgen

eDonkey2000

- Netzwerk und Anwendung
- Ziel: P2P-Dateitausch
- Baut auf dem MFTP-Protokoll auf
- ed2k-Links
  - Links ohne Lokationsangabe
  - **ed2k://|file|The_Two_Towers-The_Purist_Edit-Trailer.avi|14997504|965c013e991ee246d63d45ea71954c4d|/**

    Dateiname

    Größe

    MD4-Hash
- Indizierung über spezielle Server (beliebig hinzufügbar)

eDonkey2000

- Dateiaustausch zwischen Peers
  - auch von Bruchstücken
  - bereits während des Downloads
- Anwendung eDonkey2000
  - läuft per Voreinstellung nicht auf ed2k-, sondern auf *Overnet*-Netzen (weiteres P2P-Netz auf Kadamlia-Basis)
  - Kritik: freie Version installiert Spyware und Adware

eMule

- P2P-Anwendung
  - Open-Source-Software, verschiedene Plattformen
- Nutzt eDonkey-Netz
  - neue Modifikation nutzt Kad Network (Kademlia-Basis)
- Spezialitäten
  - Zwingt den Anwender dazu, seine erhaltenen Bruchstücke anzubieten
  - Begrenzt die Download-Rate auf das Vierfache der Rate für Uploads
- Belohnungssystem für Uploads
  - „Leecher“ (*leech*=Blutegel, also „Nur-Sauger“) werden mit geringen Bandbreiten oder langer Warteschlange bestraft (bis hin zur Aussperrung)

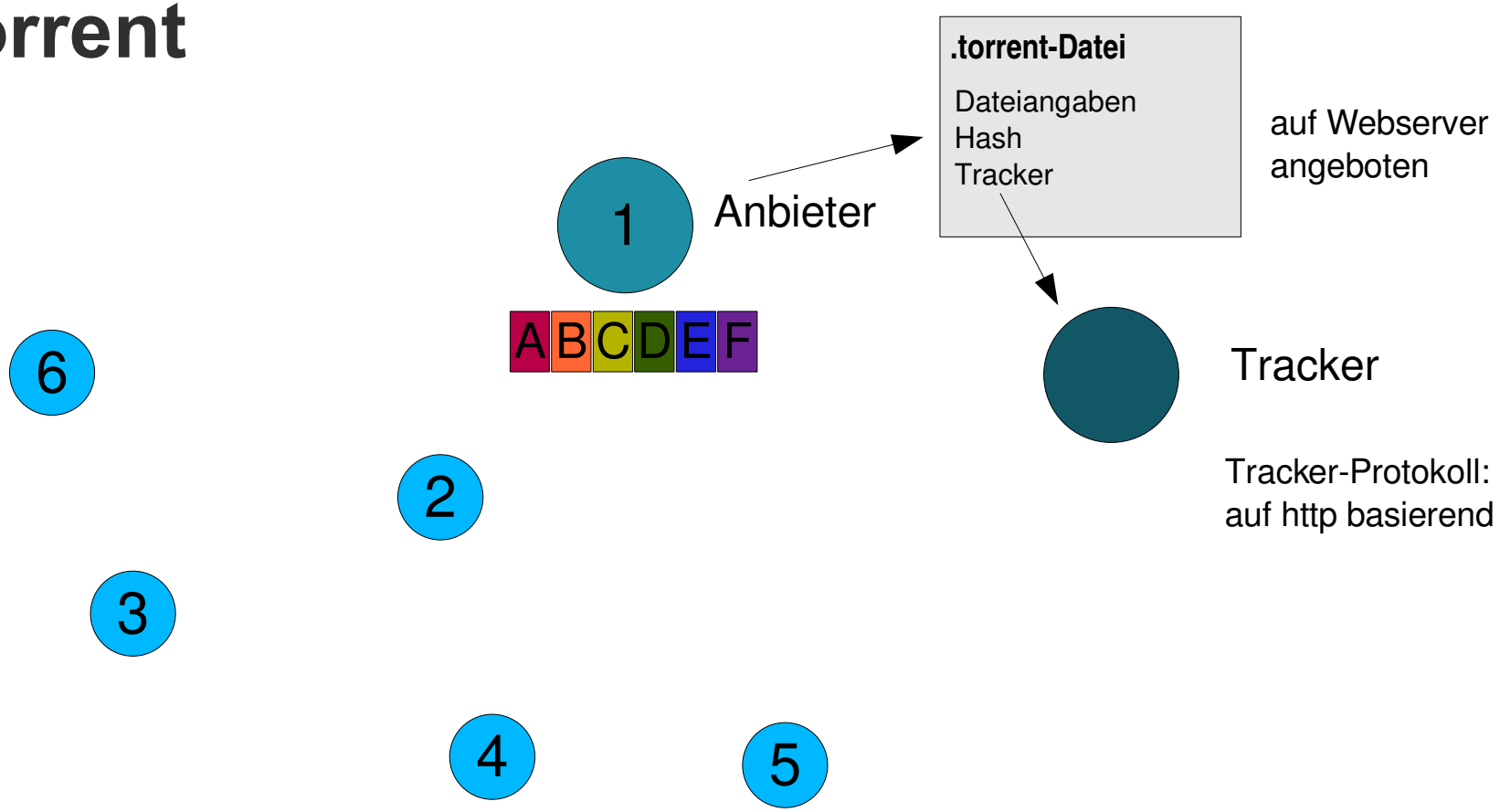
eMule

- Warum den Upload verweigern?
  - Tauschbörsen funktionieren nur nach dem Geben-und-Nehmen-Prinzip
  - Ist das unlizenzierte Herunterladen urheberrechtlich geschützter Daten bereits rechtlich angreifbar, dann aber erst recht das Hochladen (-> eigentlicher Urheberrechtsbruch)
- eMule wird häufig zum Herunterladen sehr großer Dateien (ganze CDs) verwendet
  - lange Warteschlangen

BitTorrent

- Referenzimplementierung in Python
- Open Source
- Einsatz
  - Geeignet vor allem für große Dateien (>100MB)
  - Dateien werden als Bruchstücke (etwa 256 KB) verteilt
  - SHA1-Hash für Datenintegrität

BitTorrent



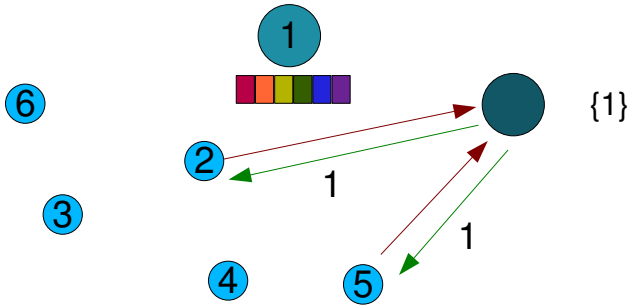
Tracker-Protokoll:
auf http basierend

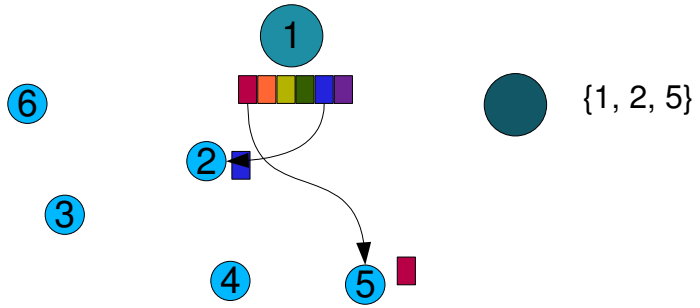
2-6 möchten Datei herunterladen

Säer (seed) – hat volle Kopie
Sauger (leech) – hat nur Bruchstücke

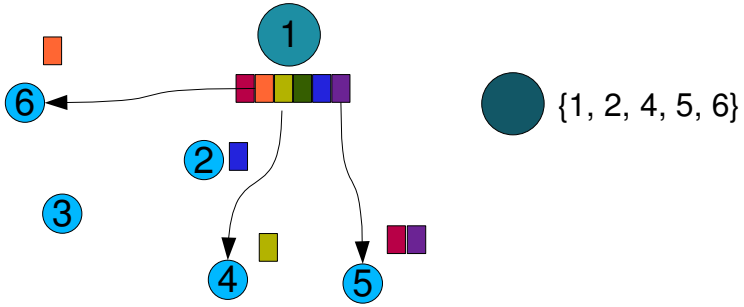
Jeder Sauger wird nach Erhalt aller Stücke zum Säer

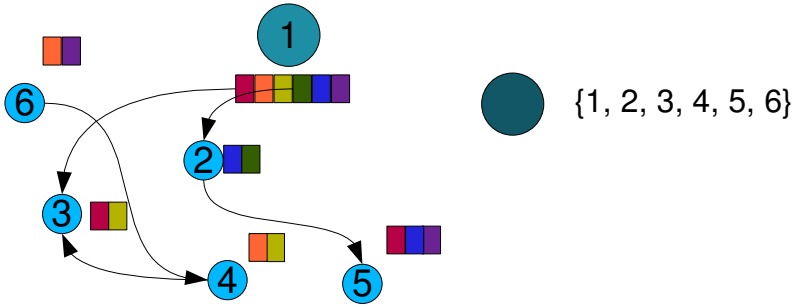
BitTorrent

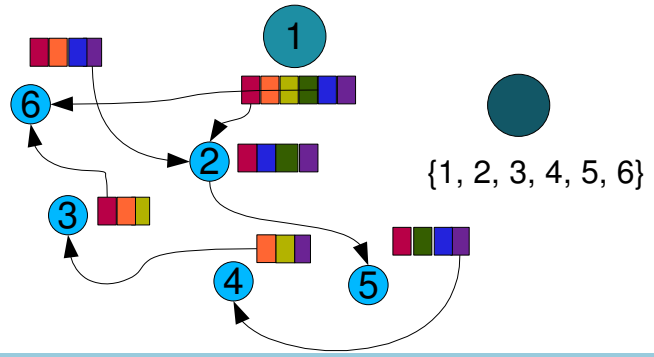




(Trackeranfragen
nicht mehr gezeigt)







- Geringe Belastung für Anbieter
- Tracker verfolgt, wer Teile der Datei geladen hat
- Peers fragen untereinander nach Bruchstücken

BitTorrent

- Tracker gibt nur an, wer die Datei herunterlädt
  - liefert bei Anfrage eine zufällige Auswahl A
- Jeder Interessent fragt bei seinen Peers in A nach Dateifragmenten
- Die seltensten zuerst herunterladen
  - führt dazu, dass die Originaldatei schnell vom ursprünglichen Anbieter heruntergeladen wird
  - seltene Teile werden schnell repliziert (Robustheit!)
- Steuerung
  - Abwürgen (choking) des Uploads
  - Diejenigen bevorzugen, die auch von sich herunterladen lassen
  - Freigabe (unchoke), um Bandbreite nicht brach liegen zu lassen

BitTorrent

- Vorteil für Anbieter
  - kann gewöhnlich wegen evtl. Uploadbegrenzung keinen Download anbieten
  - hier jedoch: Verteilung des Downloads auf viele Schultern
  - Original-Säer kann nach erfolgtem Upload vom Netz gehen
- Problem
  - Tracker: führt Buch über die Klienten und ist von großem Interesse für die Verfolgung von Urheberrechtsbrüchen
- Ausweg
  - Trackerlose Systeme: Jeder Klient wird zu einem „Lightweight tracker“
    - basiert auf einer VHT nach Kademlia, um BT-Peer-Informationen zu verteilen

Zusammenfassung

- Peer-to-Peer
  - bessere Ausnutzung freier Ressourcen
  - Replikation von Daten
- Forschung
  - schnelles Auffinden der Daten
  - Selbstkonfiguration
- Tauschbörsen: Wohin geht die Reise?
  - Illegalität?
  - Neue Vertriebswege?