# 7 Peer-to-Peer-Architekturen

Dieses Kapitel behandelt anwendungsspezifische und architekturelle Aspekte von Peer-to-Peer-Systemen. Ziel ist es, das Peer-to-Peer-Paradigma in seinen unterschiedlichen Ausprägungen anhand weit verbreiteter bzw. technologisch interessanter Systeme darzustellen, daraus architekturelle Muster abzuleiten und dem Leser/der Leserin ein umfassendes Verständnis dieses Ansatzes zu vermitteln. Mittels dieses Wissens soll es ermöglicht werden, abzuschätzen, ob der Peer-to-Peer-Ansatz für ein konkretes Anwendungsproblem verwendet werden kann.

## 7.1 Einführung

In verteilten Systemen, die wie das WWW in Internet-Größenordnung operieren, treten häufig die Limitationen des Client-Server-Modells zu Tage: Ressourcen sind auf einem oder einer kleinen Anzahl von Netzknoten konzentriert und um durchgehenden Zugriff (24 Stunden, 7 Tage) bei akzeptablen Antwortzeiten gewährleisten zu können, müssen ausgeklügelte Lastverteilungs- und Fehlertoleranz-Algorithmen angewendet werden. Wer kennt nicht den Fall, wenn sich das World Wide Web beim Zugriff auf beliebte Server zu Spitzenzeiten eher als „World Wide Wait“ beschreiben ließe? Neben dieser Konzentration der Serverlast kommt es zu einem ähnlichen Problem in Bezug auf die Netzwerk-Bandbreite: Selbst wenn noch Serverkapazität vorhanden wäre, um Anfragen zu behandeln, ist dies sehr oft auf Grund der beschränkten, zur Verfügung stehenden Netzwerk-Bandbreite nicht mehr möglich, denn immer noch stellt die Netzwerk-Bandbreite die knappste Ressource dar.

Was liegt also näher als zu versuchen, sowohl Server-Last als auch Bandbreitenverbrauch gleichmäßig unter den an einem verteilten System teilnehmenden Knoten aufzuteilen, um so diese Engpässe zu umgehen. Dies ist der Peer-to-Peer-Systemen zugrunde liegende Denkansatz. Im Prinzip löst dieser Ansatz die beschriebenen Probleme. Im Vergleich zu Client-Server-Systemen „bezahlt“ man dafür allerdings mit höherer Komplexität der verwendeten Algorithmen und zusätzlichem Sicherheitsaufwand. Wie in jedem Anwendungsfall einer Software-Architektur muss auch hier einer Entscheidung für eine bestimmte Architektur eine genaue Analyse der Vor- und Nachteile der Kandidatenarchitekturen vorausgehen, um die notwendigen Aufwände abschätzen zu können. Für etliche Bereiche, in denen Client-Server momentan die Standardarchitektur darstellt, ist der Peer-to-

Peer-Ansatz allerdings eine sehr gute Alternative, um Skalierbarkeitsprobleme zu lösen.

Im Peer-to-Peer-Ansatz agiert jeder Netzknoten sowohl als Client als auch als Server (*servent*) und stellt einen Teil der im gesamten Peer-to-Peer-System vorhandenen Information bzw. Funktionalität zur Verfügung. Jeder teilnehmende Knoten „bezahlt" seine Teilnahme, indem er Zugriff auf seine Ressourcen erlaubt. Dies bedeutet nun aber nicht, dass in einem Peer-to-Peer-System jeder beliebig auf die Ressourcen anderer Knoten Zugriff hat, sondern nur im Rahmen des durch die Konfiguration Erlaubten.

Eine recht grobe, aber eingängige Definition wird von Clay Shirkey (The Accelerator Group) gegeben:

> *Peer-to-peer is a class of applications that take advantage of resources—storage, cycles, content, human presence—available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, peer-to-peer nodes must operate outside the DNS and have significant or total autonomy of central servers.*

Mit dieser Definition kann man Peer-to-Peer-Systeme also als eine Art „Internet auf Applikationsebene über dem Internet" ansehen. Um schnell beurteilen zu können, ob es sich bei einer betrachteten Architektur um eine Peer-to-Peer-Architektur handelt, kann die Beantwortung folgender einfacher Fragen als eine Art „Lackmus-Test" verwendet werden:

- Unterstützt das System variable Verbindungen und temporäre Netzwerkadressen?
- Erlaubt es den Blattknoten im System signifikante Autonomie?

Etwas genauer können Peer-to-Peer-Systeme durch folgende Eigenschaften charakterisiert und von anderen Architekturen für verteilte Systeme unterschieden werden:

- Es gibt keine zentrale Koordination, d.h., es gibt keinen zentralen Knoten, der die Interaktionen der Peers untereinander steuert oder koordiniert.
- Es gibt keine zentrale Datenbasis. Dies bedeutet, dass jeder Peer einen Teil der im Gesamtsystem vorhandenen Daten speichert und zur Verfügung stellt, aber kein Peer den Gesamtdatenbestand verwaltet.
- Kein Peer hat eine globale Sicht auf das System. Jeder Peer kennt also nur seine „Nachbarschaft", d.h., die Peers, mit denen er interagiert.
- Das globale Verhalten (das Gesamtverhalten) des Systems entsteht aus der Kombination der lokalen Interaktionen zwischen den Peers.
- Peers sind autonom.
- Peers und Verbindungen sind nicht notwendigerweise verlässlich. Da man nicht die Glaubwürdigkeit jedes einzelnen Peers kennt und Peers auch offline sein können, muss das System Mechanismen zur Verfügung stellen, um dies zu kompensieren.

- Die gesamten im System gespeicherten Daten müssen verfügbar sein. Diese Forderung bedeutet, dass trotz verteilter Speicherung, möglichen Verbindungsausfällen, unbekannter Glaubwürdigkeit der Peers etc. die gesamten Daten jederzeit und von jedem Peer aus zugreifbar sein müssen.

Anhand dieser Definition lässt sich schon die inhärente Komplexität von Peer-to-Peer-Systemen erahnen, andererseits aber auch ihre extreme Flexibilität und Fehlertoleranz. Der Peer-to-Peer-Ansatz umgeht zwar viele Limitationen von Client-Server-Systemen, erfordert dafür allerdings mehr Aufwand für Suchoperationen, Knotenorganisation und -interaktion, Sicherheit usw. Da diese Funktionalitäten aber durch existierende Peer-to-Peer-Infrastrukturen abgedeckt werden, sind sie auf Applikationsebene nur mehr bedingt relevant.

Es ist weiters interessant anzumerken, dass nach obiger Definition einige bekannte Peer-to-Peer-Systeme eigentlich nicht als solche zu betrachten sind. Zum Beispiel umgeht Napster einen Teil der Komplexität durch eine zentrale Datenbank, die durchsucht werden kann und Referenzen auf Peers enthält, die die gesuchten Daten anbieten. Es wird also eigentlich ein Client-Server-System für die Suche und Knotenverwaltung verwendet und nur die Datenübertragung selbst ist Peer-to-Peer. Damit werden natürlich die Skalierungsprobleme von Client-Server-Systemen nicht umgangen. Gnutella, ein anderes bekanntes Peer-to-Peer-System, entspricht zwar der obigen Definition und löst die aufgeworfenen Probleme auf einfache Weise, tut dies allerdings mit Hilfe eines sehr viel Bandbreite kostenden Such- und Koordinationsmechanismus. Dies sind architekturelle Entscheidungen, welche die Entwickler der jeweiligen Peer-to-Peer-Systeme getroffen haben und die die jeweiligen Systemeigenschaften wesentlich bestimmen. Soll aufbauend auf einer Peer-to-Peer-Plattform eine Applikation entwickelt werden, so sind dadurch natürlich auch die Eigenschaften der Applikation teilweise bereits festgelegt und im Applikationsentwurf muss darauf Rücksicht genommen werden.

In den folgenden Abschnitten wird der Peer-to-Peer-Ansatz anderen verwandten architekturellen Modellen gegenübergestellt. Darauf aufbauend wird ein generelles architekturelles Modell für Peer-to-Peer-Systeme definiert. Danach werden einige architekturell interessante Peer-to-Peer-Systeme vorgestellt und ihre Vor- und Nachteile diskutiert, die sich aus den zur Erfüllung der Peer-to-Peer-Definition notwendigen Kompromissen in den jeweiligen Systemen ergeben. Abschließend wird die Anwendbarkeit von Peer-to-Peer-Systemen in Bezug auf bestimmte Anwendungsdomänen diskutiert.

## 7.2 Einordnung des Peer-to-Peer-Ansatzes

In diesem Abschnitt wird der Peer-to-Peer-Ansatz anderen verwandten Ansätzen gegenübergestellt, um so eine intuitive, aber schärfere Abgrenzung dieses Paradigmas zu ermöglichen. Neben einer konzeptuellen Gegenüberstellung wird im Schlussteil dieses Abschnitts versucht, diese Abgrenzung mittels einiger Beispiele zur Anwendbarkeit noch genauer zu verdeutlichen.

### 7.2.1 Peer-to-Peer vs. Event-basiert

In einem Event-basierten System (EBS) interagieren die einzelnen Komponenten (Peers), indem sie „Ereignisse“ (Events) generieren und empfangen. Um Events empfangen zu können, muss eine Komponente bestimmte Events oder Event-Muster subskribieren. Tritt ein passender Event bzw. ein passendes Event-Muster auf, werden die Komponenten von diesem Ereignis unterrichtet. Dieses Kommunikationsmuster ermöglicht sehr flexible Interaktionen zwischen lose gekoppelten Komponenten. Ein Anwendungsbeispiel wäre eine Groupware-Applikation, in der sich Teilnehmer laufend an- und abmelden und auf bestimmte Ereignisse reagieren, z.B. eine Börsenapplikation, die aus unterschiedlichen Quellen mit Kurs-Events versorgt wird, auf welche die einzelnen Teilnehmer mit „Buy“- oder „Sell“-Events reagieren können.

Die Gemeinsamkeiten von Peer-to-Peer- und Event-basierten Systemen sind der symmetrische Kommunikationsstil (jede Komponente kann Events subskribieren und generieren) und die dynamische Bindung zwischen Event-Produzenten und -Konsumenten, d.h., es gibt keine fixen Client-Server-Beziehungen. Während in einem Peer-to-Peer-System „aktive“ Abfragen durchgeführt werden, wird in einem Event-basierten System auf Notifikationen gewartet („passive“ Abfragen). Um die Flexibilität zu erhöhen, bieten Event-basierte Systeme außerdem komplexe Musterdefinitionssprachen, in denen sich zeitliche Abhängigkeiten zwischen Events definieren lassen sollten.

Typischerweise haben Event-basierte Systeme zwei Knotentypen, nämlich einerseits Event-Produzenten und -Konsumenten und andererseits spezielle Infrastrukturknoten, die für Event-Verteilung und die Verwaltung von Subskriptionen zuständig sind (üblicherweise in verteilter Kooperation). Die Infrastrukturknoten existieren nicht per se, sondern müssen beim Aufsetzen der Infrastruktur installiert werden und bekannt sein. Im Gegensatz dazu haben echte Peer-to-Peer-Systeme nur einen Knotentyp, der auf jedem teilnehmenden Knoten identisch ist. Außerdem müssen die Knoten nicht a priori installiert und konfiguriert werden, sondern die Struktur des Systems wird sukzessive im Laufe des Betriebs von den teilnehmenden Knoten aufgebaut.

### 7.2.2 Peer-to-Peer vs. Push

Push-Systeme können als Spezialfall von Event-basierten Systemen (EBS) betrachtet werden. Während bei EBS die Notifikation im Vordergrund steht, liegt bei Push-Systemen der Fokus auf Informationsverteilung „just in time“: Sobald Information verfügbar ist, wird sie an Subskribenten verteilt. Dabei handelt es sich im Gegensatz zu EBS um große Informationsmengen, während EBS Events von kleiner Größe (einige hundert Byte) verteilen. Die Push-Systemen zugrunde liegende Idee ist, das Pull-Interaktionsmuster des WWW, bei dem der Benutzer aktiv Informationen suchen muss, durch aktive Informationsverteilung an den Benutzer, entsprechend seiner Vorlieben, zu ersetzen.

In Push-Systemen wird Information durch Sender in Kanäle vorgruppiert (z.B. Nachrichten, Wetter, Sport), die dann von Konsumenten subskribiert werden können. Je nach Kanal ist die verteilte Information entweder frei verfügbar oder der Empfänger muss entsprechend einem Geschäftsmodell für die verteilten Informationen bezahlen (pre-paid, pay-per-view oder ähnliche).

Im Gegensatz zum symmetrischen Kommunikationsstil von Peer-to-Peer-Systemen weisen Push-Systeme ein sehr asymmetrisches Interaktionsmuster auf: Kommunikation wird in der großen Mehrzahl der Fälle nur vom Broadcaster initiiert. Dies ergibt sich aus dem Ziel der zeitgerechten Informationsverteilung, wohingegen Peer-to-Peer-Systeme primär für Suchaufgaben verwendet werden. Durch Subskription von Kanälen gibt der Benutzer bereits a priori bekannt, an welcher Art von Information er/sie interessiert ist. Dadurch wird im Unterschied zu Peer-to-Peer und EBS eine (relative) statische Bindung zwischen Produzenten und Konsumenten hergestellt. Diese geringere Flexibilität ermöglicht dafür andererseits wieder größere Kontrolle. Zusätzlich wird in Push-Systemen häufig auf beiden Seiten einer Kommunikation entsprechend der Profile der Teilnehmer gefiltert, um einerseits unnötige Übertragungen zu vermeiden und andererseits nur vom Teilnehmer gewünschte Information zu übermitteln.

Push-Systeme benötigen eine spezielle Verteilungsinfrastruktur, um zu skalieren. Diese Infrastruktur sollte für Produzenten und Konsumenten möglichst transparent sein, d.h., von diesen verwendet, aber nicht wahrgenommen werden. Diese Infrastruktur muss vor Inbetriebnahme des Systems installiert und im Laufe der Systemlebensdauer gewartet werden.

Ein Push-System besteht aus drei Arten von Knoten (Produzent, Konsument, Verteilungsknoten), im Unterschied zum einheitlichen Knotentyp in Peer-to-Peer-Systemen.

### 7.2.3 Peer-to-Peer vs. Mobile Agents

Ein „Mobile Agent“ besteht aus Daten und Code. Er bewegt sich frei („aus eigener Entscheidung“) in einem Netzwerk, um eine Aufgabe für seinen Besitzer zu erfüllen. Ein Agent (im Folgenden wird immer impliziert, dass es sich um einen mobilen Agenten handelt) kann dazu mit anderen Agenten zusammenarbeiten und kann im Laufe seines Lebenszyklus „intelligenter“ werden, beispielsweise seine Auswahlstrategie zur Ermittlung des nächsten zu besuchenden Knotens verbessern.

Es können grob zwei Arten von Mobilität unterschieden werden:

- Schwache Mobilität: Code und Daten werden zwischen den Netzwerkknoten transferiert, d.h., der Agent wird vom Laufzeitsystem gestoppt und mit seinen persistenten Daten auf einen neuen Knoten transferiert und dort neu gestartet.
- Starke Mobilität: Code, Daten und Execution Stack werden zwischen den Netzwerkknoten transferiert, d.h., der Agent wird vom Laufzeitsystem gestoppt und mit seinem aktuellen Zustand auf einen neuen Knoten transferiert. Dort setzt der Agent genau an der Stelle seine Abarbeitung fort, an der er unterbrochen wurde.

Da starke Mobilität ungleich höhere Anforderungen an das Mobile-Agent-System stellt, implementieren kommerzielle Agentensysteme üblicherweise schwache Mobilität.

Agentensysteme und Peer-to-Peer-Systeme ähneln sich in den Bereichen Suche und Navigation: Bei Peer-to-Peer-Systemen muss entschieden werden, an welche(n) Knoten ein Such-/Update-Request weitergeleitet werden soll, wohingegen in einem Mobile-Agent-System bestimmt werden muss, wohin der Agent geschickt wird. In diesem Sinne könnte man Agenten als „aktive" Variante eines Peer-to-Peer-Systems sehen („active queries").

Auf Grund ihrer Eigenschaften bieten Agentensysteme zwar ein sehr hohes Maß an Flexibilität, dies allerdings zu einem Preis, der die praktische Einsetzbarkeit massiv beeinflusst. Sie benötigen nämlich auf jedem teilnehmenden Knoten ein Laufzeitsystem, das üblicherweise ziemlich umfangreich ist und den Agenten eine einheitliche Umgebung auf den unterschiedlichen Systemplattformen bieten muss. Des Weiteren stellt Security ein primäres Problem dar: Es muss sowohl das System bzw. die Plattform vor dem Agenten – der ja auch ein Virus oder ein Trojanisches Pferd sein könnte – als auch der Agent vor dem „gastgebenden" System geschützt werden, da dieses ja die Daten bzw. das Verhalten des Agenten verändern könnte.

Auf Grund der Ähnlichkeiten könnten Peer-to-Peer-Systeme in einigen Bereichen Agentensysteme ersetzen, da sie mehr auf verteiltes Datenmanagement abzielen und geringere Systemanforderungen als Agentensysteme stellen (eine Abfrage zu schicken, erfordert weniger Netzbandbreite, als den Code eines Agenten zu übertragen, die notwendige Security ist viel geringer, ebenso wie die Systemanforderungen).

### 7.2.4 Peer-to-Peer vs. Distributed Databases

Ein weiterer verwandter Ansatz zu Peer-to-Peer-Systemen sind verteilte Datenbanken („distributed databases" – DDBs). Wie bei Peer-to-Peer-Systemen geht es um verteiltes Datenmanagement, basierend allerdings auf einem völlig anderen Ansatz: Bei DDBs werden große Datenbanken auf physikalisch verteilte Netzwerkknoten aufgeteilt, um dadurch eine effizientere Durchführung von Abfragen durch Parallelisierung zu ermöglichen.

Im Unterschied zu Peer-to-Peer-Systemen steht allerdings die Gewährleistung der bekannten Transaktionseigenschaften (atomicity, consistency, isolation, duration – ACID) im Vordergrund. Um dies zu ermöglichen, existieren eine Vielfalt an Update-Strategien („*lazy* vs. *eager*") und spezielle Commit-Protokolle (Zwei-Phasen-Commit). Üblicherweise handelt es sich dabei allerdings um Ansätze mit zentraler Koordination.

Im Gegensatz dazu liegt der Schwerpunkt von Peer-to-Peer-Systemen auf der Daten-Verteilung. Dennoch kommen viele Ansätze, die auch in den Peer-to-Peer-Bereich Einzug gehalten haben, aus dem Bereich der DDBs, die sehr ähnliche Probleme zu lösen hatten, vor allem in den Bereichen verteilte Indizierungsstruk-

turen und deren Skalierbarkeit und lokale Autonomie der teilnehmenden Knoten. Als Beispiele für Ansätze im Bereich Skalierbarkeit seien hierzu nur die LH*-Familie skalierbarer Hash-Index-Strukturen (Litwin u. Neimat 1997), Snowball (Vingralek et al. 1998), ein skalierbares Speichersystem für Workstation-Cluster oder Fat-Btree (Yokota et al. 1999), ein skalierbarer B-Baum für parallele Datenbanken genannt. Mariposa (Stonebraker et al. 1996), eine verteilte relationale Datenbank, wiederum versucht auch den Autonomie-Aspekt zu behandeln und basiert auf einem ökonomischen Modell.

Peer-to-Peer-Systeme müssen ebenfalls beide Bereiche, Skalierbarkeit und Autonomie, berücksichtigen und können daher Konzepte aus dem Gebiet der DDBs übernehmen bzw. anpassen, wodurch die Entwicklung schneller vorangetrieben werden kann.

### 7.2.5 Anwendungsbereiche

Nach dieser Gegenüberstellung stellt sich nun die Frage, wie denn die einzelnen Ansätze gegeneinander einzuordnen sind. Wie so oft ist keine absolut eindeutige Antwort möglich. Dennoch soll anhand Tabelle 7.1 eine grobe, anwendungsspezifische Klassifizierung versucht werden, die es ermöglichen soll, die Anwendbarkeit der einzelnen Paradigmen für konkrete Problemstellungen abzuschätzen.

**Tabelle 7.1.** Anwendungsbereiche verteilter Architekturen

| Anwendungsfall | Architektur | Beispiel | Alternativen |
|---|---|---|---|
| Fallweise Suche | Suchmaschine | Andere Stadt: Wo ist die nächste Autovermietung? | Peer-to-Peer System, Mobile Agents |
| Benachrichtigung (Ereignis-Muster) | Event-basiertes System | Aktienkurse fallen unter einen Schwellwert | Push System |
| Systematische Suche (regelmäßig) | Peer-to-Peer System | Regelmäßige Suche nach Jobprofilen | Suchmaschine, Mobile Agents |
| Stetiger Informationsfluss | Push-System | Nachrichtenkanal | Event-basiertes System |

Des Weiteren kann man diese Systeme, wie in Abb. 7.1 gezeigt, anhand der notwendigen Interaktionsinitiative klassifizieren.


**Abb. 7.1.** Interaktionsklassen

Bei Event-basierten Systemen und Push-Systemen geht die Initiative vom System aus. D.h., nach Definition bzw. Konfiguration übernimmt das System die Initiative und liefert selbsttätig kontinuierlich Informationen entsprechend der Konfiguration. Mobile Agents bzw. Peer-to-Peer-Systeme hingegen erfordern Aktivitäten seitens des Anwenders, bevor Aktionen gesetzt werden. Wird nun eine dieser Architekturen für eine verteilte Anwendung in Betracht gezogen, sollten die Kommunikationsmuster der Anwendung genau analysiert werden, um die am besten passende Architektur anwenden zu können.

## 7.3 Das Peer-to-Peer-Modell

Nach dieser Abgrenzung des Peer-to-Peer-Modells gegenüber verwandten Ansätzen soll dieser Abschnitt das eigentliche hinter Peer-to-Peer (P2P) stehende architekturelle Modell beschreiben und erklären.

Der P2P-Ansatz ist keineswegs neu, sondern war bereits im ursprünglichen Design des Internets (Routing, DNS etc.) vorhanden. Neu ist hingegen, dass dieser Ansatz nun auch auf anderen Abstraktionsebenen angewendet wird. Bezüglich P2P lassen sich folgende Abstraktionsebenen unterscheiden:

- **Netzwerkebene:** Grundlegende Routing-Dienste, die es ermöglichen, Requests in applikationsunabhängiger Weise über das physikalische Netzwerk zu routen.
- **Datenzugriffsebene:** Suche und Änderung von Ressourcen mit Hilfe applikationsspezifischer Zugriffsstrukturen.
- **Dienstebene:** Kombination und Erweiterung von Funktionalitäten der Datenzugriffsebene, um höherwertige Dienste zur Verfügung zu stellen. Die Bandbreite dieser Dienste kann von einfachem File-Sharing bis hin zu komplexen Geschäftsprozessen reichen.
- **Benutzerebene:** Gruppierung von Benutzern („Communities“) und Unterstützung von Benutzer-Interaktionen unter Verwendung der Dienstebene für Community-Management und Informationsaustausch.

Das P2P-Paradigma kann interessanterweise unabhängig auf jeder dieser Ebenen auftreten. Die Kombination dieser Ebenen in einem konkreten System und die Menge der auf den einzelnen Ebenen verwendeten P2P-Konzepte charakterisieren ein konkretes P2P-System. Tabelle 7.2 listet einige Beispiele für die Anwendung von P2P-Architekturen auf den unterschiedlichen Ebenen auf.

**Tabelle 7.2.** Anwendung des P2P-Paradigmas auf unterschiedlichen Systemebenen

| Ebene | Applikations-domäne | Dienst | Beispielsystem |
|---|---|---|---|
| Netzwerkebene | Internet | Routing | TCP/IP, DNS |
| Datenzugriffsebene | Overlay-Netzwerke | Ressorcen-Lokation | Gnutella, Freenet |
| Dienstebene | P2P-Applikationen | Nachrichten-dienste, verteilte Bearbeitung | SETI@Home, Napster, Groove |
| Benutzerebene | Benutzer-Communities | Kollaboration | eBay, Ciao |

Für die Netzwerkebene des Internets wurde bereits darauf hingewiesen, dass hier auf P2P-Ansätze, z.B. um Pakete zu routen, zurückgegriffen wird. Diese Funktionalität könnte theoretisch auch zum Auffinden von Ressourcen verwendet werden. Moderne Lokationssysteme wie Gnutella oder Freenet sind jedoch als so genannte Overlay-Netzwerke (Datenzugriffsebene) konzipiert und bilden somit eine höhere funktionale Ebene in Bezug auf die Unterstützung applikationsspezifischer Identifikationsmechanismen für Ressourcen, semantisches Routing und zusätzliche Dienste wie „Group-Membership“, „Trust“, oder Authentifizierung. Overlay-Netzwerke sind vielleicht der grundlegende Beitrag von P2P-Systemen im Bereich verteilter Systeme.

Auf nächsthöherer Ebene werden Dienste in P2P-Manier angeboten. In Napster ist beispielsweise die Datenzugriffsebene (Suche) zentralisiert und nur die Dienstebene (Dateiübertragung) P2P. Da sehr viele soziale und ökonomische Prozesse im Grunde genommen auf P2P-Interaktionen fußen, zeigt sich dies auch in den auf Benutzerebene vorhandenen Systemen wie beispielsweise eBay oder Empfehlungssysteme wie Ciao (www.ciao.com). Diese Systeme sind sowohl auf Datenzugriffs- als auch Dienstebene zentralisiert, Benutzerinteraktionen jedoch folgen einem P2P-Muster.

Eine andere Möglichkeit der Klassifikation von P2P-Systemen wäre zum Beispiel auch die Betrachtung der generellen Anwendbarkeit: P2P-Applikationen (Gnutella, Freenet) versuchen ein relativ eng begrenztes Problem zu lösen, P2P-Plattformen wie JXTA (Gong 2001) hingegen stellen Architekturen, Dienste und Komponenten zur Entwicklung von P2P-Systemen zur Verfügung und P2P-Algorithmen wiederum stellen allgemeine, universell anwendbare Mechanismen bereit.

### 7.3.1 Charakteristische architekturelle Parameter

Ein P2P-System lässt sich entlang dreier Designdimensionen charakterisieren: Strukturiertheit, Hierarchiegrad und Kopplungsgrad. Die Position eines konkreten Systems in Bezug auf diese Achsen bestimmt im Wesentlichen alle Systemeigenschaften und legt zum Großteil auch die für die Implementierung des P2P-Systems möglichen Architekturen fest.

#### *7.3.3.1 Strukturiertheit*

In unstrukturierten P2P-Systemen halten Peers keine Information über die Ressourcen anderer Peers. Dies ist zum Beispiel in Gnutella der Fall, das Anfragen einfach so lange weiterleitet und Antworten liefert, bis die Lebensdauer einer Anfrage abläuft. Dieser Ansatz ermöglicht zwar einen extrem hohen Grad an Unabhängigkeit und Fehlertoleranz, dies aber auf Kosten von Effizienz und vorhersagbarem Systemverhalten.

In strukturierten P2P-Systemen hingegen speichern Peers lokal Wissen über die von anderen Peers verwalteten Ressourcen, wodurch eine zielorientierte Suche ermöglicht wird, dabei jedoch zusätzlicher Aufwand für die Verwaltung dieser Information notwendig wird. Ein Beispiel für die hier gemeinte Information sind Routing-Tabellen, die auf dem aktuellen Stand gehalten werden müssen.

#### *7.3.3.2 Hierarchiegrad*

Das Design eines P2P-Systems weist einen bestimmten Grad an hierarchischen Strukturen auf. Typische Modelle sind:

- Zentralisiertes Modell
  - Es existiert ein globaler Index an einer zentralisierten Stelle (möglicherweise repliziert).
  - Der globale Index ist ein „*single point of failure*“.
  - Interaktionen nach der Indexabfrage geschehen direkt zwischen den einzelnen Peers.
  - Beispiel: Napster (Napster homepage 2001, Dr. Scholl 2001)
- Dezentrales (verteiltes) Modell
  - Es existiert kein globaler Index, der an einer zentralen Stelle konzentriert wäre (kein „*single point of failure*“).
  - Peers werden nicht von einer zentralen Stelle koordiniert, sondern das globale Systemverhalten ergibt sich aus der Kombination der lokalen Interaktionen der Peers.
  - Peers interagieren direkt miteinander, z.B. Gnutella (Gnutella Homepage 2001; Clip2 2001), über eine dynamisch aufgebaute Kette von Mediatoren, z.B. Freenet (Clarke et al. 2001; Freenet Project 2001), oder in einer virtuellen Baumstruktur, z.B. P-Grid (Aberer 2001; Aberer et al. 2002).
- Hierarchisches Modell
  - Mischung aus zentralisiertem und dezentralisiertem Modell: Normale Peers und hierarchische Super-Peers, die den globalen Index verwalten.
  - Beispiel: FastTrack

#### *7.3.3.3 Kopplungsgrad*

In stark gekoppelten P2P-Systemen existiert zu jedem Zeitpunkt genau eine Gruppe, der alle teilnehmenden Peers angehören, und nur einzelne Peers können sich dieser Gruppe anschließen bzw. sie verlassen. In dieser Art von Systemen, wie beispielsweise Chord (Dabek et al. 2001), wird Peers beim Eintritt in die Gruppe eine statische, logische Identifikation zugewiesen, die die Rolle des Peers in Bezug auf die Gruppe genau festlegt, d.h., welche Daten er hält oder die Art und Weise wie Nachrichten behandelt werden. In diesen Systemen sind die für Informationen möglichen Schlüsselintervalle mit jenen für die Peer-Identifikation zur Verfügung stehenden identisch. Diese Systemstruktur beschränkt natürlich eindeutig die Möglichkeiten Peer-Populationen, die sich unabhängig von einander entwickeln, zu vereinigen bzw. zu trennen.

Lose gekoppelte Systeme hingegen bieten diese Möglichkeiten. Peers können zwar zu einem bestimmten Zeitpunkt eine bestimmte Rolle innehaben, diese Rolle und damit auch die logische Peer-Adresse können sich jedoch zeitlich dynamisch ändern. Gnutella ist ein typischer Vertreter für ein lose gekoppeltes P2P-System.

### 7.3.4 Weitere Kriterien

Weitere Kriterien, die für die Anwendbarkeit bzw. Klassifikation eines konkreten Systems in Betracht gezogen werden können, sind:

Skalierbarkeit des Systems

- Robustheit und Fehlertoleranz
- Sicherheitsaspekte („Vertrauen“ in andere Peers, Authentizität etc.)
- Wie werden Updates behandelt (Effizienz, Konsistenz etc.)?

## 7.4 Beispiele für Peer-to-Peer-Architekturen

Im Folgenden werden einige bekannte bzw. technisch interessante Peer-to-Peer-Systeme exemplarisch beschrieben, um die zugrunde liegenden Konzepte und die Unterschiede zwischen den einzelnen Systemen zu verdeutlichen, die zum Teil entscheidenden Einfluss auf die Anwendbarkeit haben.

### 7.4.1 Napster

Napster (Napster 2001) ist das wohl bekannteste Peer-to-Peer-System. Es wurde als erstes Peer-to-Peer-System weit über die Grenzen der Informatik hinaus als Musiktauschbörse bekannt, dann wegen Copyright-Verstößen verurteilt und hat bis jetzt den Neustart nicht wieder geschafft.

Technisch gesehen handelt es sich bei Napster um ein relativ simples Peer-to-Peer-System: Der Napster-Server hält eine zentrale Datenbank der angebotenen

MP3/WMA-Musikdateien, die Dateinamen mit IP-Adressen von Peers verknüpft. Clients (man beachte die Terminologie) melden sich bei diesem Server an und schicken eine Liste von Dateien, die sie zur Verfügung stellen. Neubenutzer müssen dazu zuerst einen Account beim Napster-Server einrichten.

Jeder Client kann Suchanfragen an den Napster-Server stellen und erhält als Suchergebnis eine Liste an Clients, die der Suchanfrage entsprechende Dateien anbieten. Aus dieser Liste kann ein Client ausgewählt werden, um von diesem die gewünschte(n) Datei(en) herunterzuladen.

Abb. 7.2 zeigt die Architektur von Napster.

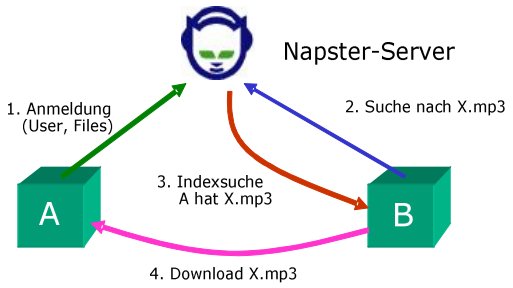


**Abb. 7.2.** Architektur und Kommunikationsmodell von Napster

Bei Napster handelt es sich also um kein reines Peer-to-Peer-System, sondern vielmehr um eine Kombination des Client-Server- und des Peer-to-Peer-Ansatzes. Das Kommunikationsprotokoll von Napster wurde nie publiziert. Es existieren lediglich Protokolldefinitionen, die „re-engineered“ wurden (Dr. Scholl 2001). Auf Grund der zentralen Datenbank von Napster (auch wenn sie in Wirklichkeit auf vielen Servern repliziert war), stellt diese einen Flaschenhals dar, der die Skalierbarkeit von Napster erheblich beschränkt. Der Vorteil dieser Architektur ist allerdings, dass dies eine einfache Kontroll- und Steuermöglichkeit bietet, was in vielen verteilten Systemen notwendig ist bzw. gewünscht wird. Außerdem gibt es mit einer solchen Architektur geringere Probleme bezüglich der Konsistenz des (zentralen) Index als bei verteilten Indexansätzen.

Das Kommunikationsmodell von Napster existiert im Internet im Grunde in vielerlei Ausprägung, z.B.: Suche in einer Suchmaschine, die URLs auf die Treffer liefert (C-S-Teil) und danach Download der Treffer (P2P-Teil).

### 7.4.2 Gnutella

Gnutella ist eines jener zahlreichen Beispiele, die die Kapazität des Internets für die schnelle Verbreitung innovativer Ansätze belegen. Ursprünglich wurde Gnutella als „Quick-Hack“ zum Austausch von Kochrezepten von Nullsoft-Entwicklern (Winamp) entwickelt und unter der GNU General Public License (GPL) als Free Software am Nullsoft-Web-Server platziert. Schon nach wenigen

Stunden wurde Gnutella auf Anordnung von AOL (Besitzer von Nullsoft) wieder vom Web-Server genommen. Jedoch reichte diese kurze Zeit bereits aus, dass eine sehr große Anzahl an Benutzern Gnutella heruntergeladen hatte. Das Gnutella-Protokoll wurde „reverse engineered“ und bald gab es eine Anzahl an Gnutella-Clients, die sich seitdem großer Beliebtheit erfreuen.

Im Unterschied zu Napster gibt es in Gnutella keinen zentralen Server. Dadurch ist das System um einiges robuster als Napster, da der Ausfall eines Rechners die Systemeigenschaften nicht beeinflusst. Auch sind somit rechtliche Schritte wegen Copyright-Verletzung nicht mehr möglich. Gnutella ist weitgehend selbstorganisierend, d.h., Peers binden sich dynamisch an beliebigen Stellen ins Gnutella-Netzwerk ein. Dazu muss die Adresse mindestens eines Peers, der online ist, bekannt sein. Um dieses Henne/Ei-Problem zu lösen, sind in der Gnutella-Peer-Software Peers mit hoher Verfügbarkeit vorkonfiguriert, welche auf Anfrage Peer-Listen liefern, aus denen Peers kontaktiert werden können. Die Erstellung der Listen ist durch einfaches Mithören der Verkehrsdaten möglich. Insofern ist auch Gnutella in gewissem Sinne zentralisiert, wenn auch nur der Start-up eines Peers.

Nachrichten werden, basierend auf einem „Constrained Broadcast“-Ansatz, durch das Netz der Gnutella-Peers geroutet. Dabei halten sie typischerweise Verbindungen mit 4 anderen Peers aufrecht, die sich aber im Lauf der Zeit ändern können (im Zuge des Routings von Nachrichten lernt jeder Peer laufend neue Peers kennen). Gnutella-Protokoll-Pakete haben typischerweise eine Lebensdauer (TTL) von 7, welche von jedem Peer verändert, typischerweise um 1 verringert werden kann. Da die Struktur des Gnutella-Netzwerks ein zufälliger Graph ist, besitzt jedes Paket eine eindeutige Identifizierung, um Paket-Schleifen erkennen zu können. Um Verbindung zu anderen Gnutella-Peers aufnehmen zu können, muss mindestens ein anderer Peer bekannt sein. Für diesen „Bootstrap“ gibt es, wie bereits erwähnt, spezielle Peers, die Listen von Gnutella-Peers anbieten. Dies ist jedoch nicht in der Gnutella-Protokoll-Spezifikation genormt (Clip2 2001). Das Gnutella-Protokoll umfasst die in Tabelle 7.3 angegebenen 5 Pakettypen.

**Tabelle 7.3.** Paket-Typen in Gnutella

| Pakettyp | Beschreibung | Enthaltene Information |
|---|---|---|
| Ping | Ankündigung der eigenen Verfügbarkeit und Finden weiterer Peers | Keine |
| Pong | Antwort auf ein Ping | IP-Adresse und Portnummer des antwortenden Peers sowie die Anzahl und die Größe (in kB) der lokal zur Verfügung gestellten Information. |
| Query | Suchanfrage | Suchkriterium, minimale Netzwerk-Bandbreite, die antwortende Peers (Treffer) haben sollten; |
| QueryHit | Wird von Peers retourniert, welche die gewünschten Daten haben. | IP-Adresse, Portnummer und Netzwerkbandbreite des antwortenden Peers, Anzahl der Suchresultate und Treffer der Suche |
| Push | Datei-Download-Anfrage für Peers hinter Firewalls | Peer-Id, Index der angeforderten Datei, IP-Adresse und Port, an den die Datei geschickt werden soll. |

Diese Nachrichtenarten werden von allen Gnutella-Peers mittels eines beschränkten Broadcast-Mechanismus verteilt: Empfängt ein Peer ein Paket, dann reduziert er zuerst dessen Time-to-live-Feld (TTL). Wenn die TTL dann noch größer als 0 ist und die eindeutige Identifikation des Pakets dem Peer noch unbekannt ist (Schleifen-Erkennung), dann wird das Paket an alle dem Peer bekannten Peers weitergeschickt. Zusätzlich überprüft der Peer, ob er selbst auf die Nachricht reagieren sollte. Beispielsweise mit einem *Pong* als Antwort auf ein empfangenes *Ping*. Wurde hingegen ein *Query*-Paket empfangen, dann überprüft der Peer seinen lokalen Speicher, ob passende Daten vorhanden sind. Ist dies der Fall, schickt er ein *QueryHit*-Paket. Alle Antwortpakete werden entlang des gleichen Pfades zurückgeschickt, auf dem sie empfangen wurden.

Abb. 7.3 zeigt eine einfache Gnutella-Interaktion zum Aufbau der dynamischen Gnutella-Netzwerkstruktur, wenn ein neuer Peer (*A*) sich in eine existierende Gnutella-Struktur einklinkt.

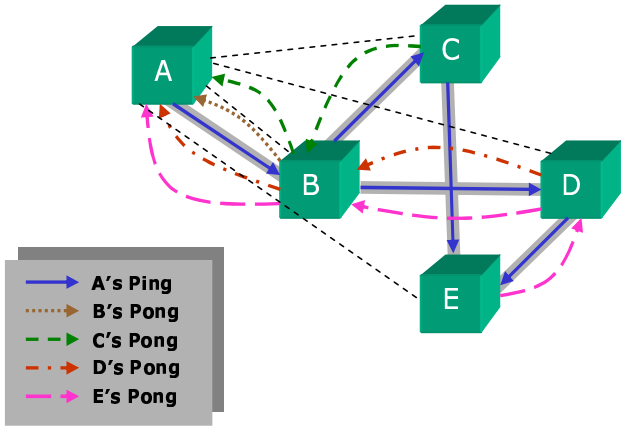


**Abb. 7.3.** Aufbau eines Gnutella-Netzwerks (Ping/Pong)

Der Ablauf ist dabei wie folgt (die breiten grauen Linien zeigen die bestehenden Verbindungen zu Beginn der Interaktion an): Peer *A* kontaktiert Peer *B* und sendet eine *Ping*-Nachricht. *B* antwortet mit einem *Pong* und leitet *A*'s *Ping* weiter, d.h., er leitet es weiter an alle ihm bekannten Peers, in diesem Fall *C* und *D*, die wiederum mit einem *Pong* antworten, usw. Nach einigen dieser Ping/Pong-Interaktionen „kennt" *A* die Peers *B*, *C*, *D* und *E* und ist in die Infrastruktur eingeklinkt.

Wenn *A* nun eine Suche startet und als Antwort ein *QueryHit*-Paket empfängt, versucht er zu dem in dem Paket angegebenen Peer eine direkte Verbindung aufzubauen und die gesuchte Datei herunterzuladen. Dies passiert mittels einer sehr stark vereinfachten HTTP-GET-Anfrage (Clip2 2001). Befindet sich der zu kontaktierende Peer hinter einer Firewall, dann kann eine solche Verbindung nicht hergestellt werden. Stattdessen schickt der Peer ein *Push*-Paket (entlang des Pfa-

des, über den das *QueryHit*-Paket empfangen wurde, denn diese Verbindung existiert sicher, da sich der Ziel-Peer ja bereits im Gnutella-Netzwerk befindet) an den Peer hinter der Firewall, welcher dann von sich aus versucht, eine Verbindung mit dem anfragenden Peer herzustellen (was auf Grund der Firewall-Konfiguration üblicherweise möglich ist) und dem anfragenden Peer damit die Möglichkeit eines Downloads via HTTP GET ermöglicht. Sind allerdings beide Peers hinter Firewalls, dann ist ein Download nicht möglich.

Abb. 7.4 zeigt eine Suchinteraktion in Gnutella.

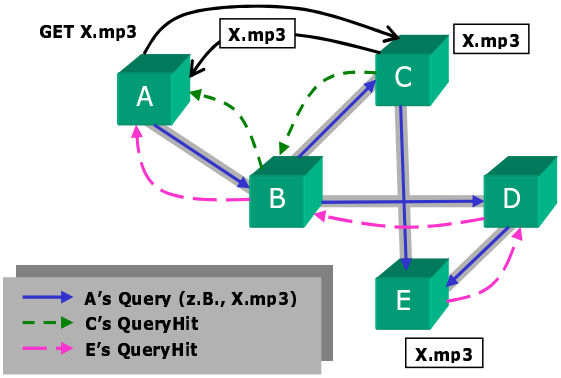


**Abb. 7.4.** Suche in Gnutella (Query/QueryHit/GET)

### *7.4.2.1 Netzwerklast*

Aus Benutzersicht ist Gnutella ein einfaches, aber dennoch sehr effektives Protokoll: Die Trefferraten sind ziemlich hoch, es ist fehlertolerant in Bezug auf den Ausfall von Peers und es passt sich gut an Änderungen der „Peer-Population" an. Aus Netzwerksicht hingegen werden diese positiven Eigenschaften teuer bezahlt, denn Gnutella verwendet eine Form von „beschränktem Broadcast" und jeder Peer, der einen Request empfängt, durchsucht seine lokale Datenbank nach möglichen Treffern.

Wird beispielsweise eine typische Konfiguration mit einer TTL von 7 und durchschnittlich 4 Verbindungen $C$ pro Peer zu Grunde gelegt, dann ergibt sich die Gesamtanzahl an Nachrichten (inklusive Antwortpakete), die von einem einzigen Gnutella-Paket verursacht werden, als:

$$2 * \sum_{i=0}^{TTL} C * (C-1)^i = 26240$$

Kürzlich durchgeführte Experimente (Sripanidkulchai 2001) zeigen, dass die durch Gnutella verursachte Netzwerklast bis zu 3,5 Mbit/s betragen kann (oder 353.296 Abfragen in 2,5 Stunden, wie ein anderes Experiment aus (Sripanidkulchai 2001) belegt). Um diese sehr hohe Netzwerklast zu verringern, schlägt der Autor dieser Studie vor, Caching einzusetzen. Seine Experimente zeigen nämlich,