

## Introduction and Application Areas

### Key characteristics of P2P systems:

- Equality (all peers are equal), Autonomy (no central control), Decentralization (no centralized services), Self-organization (no coordination from outside), Shared resources (peers may use resources provided by other peers)

### Key characteristics of Peers:

- Have all the same capabilities, can act as “clients” and as “servers” at the same time, are typically located at the edges of the network (end-to-end principle)

### End-to-End principle:

- Whenever possible, communication protocol operations should be defined to occur at the end-points of a communication system, or as close as possible to the resource being controlled

### Differences between Client/Server systems and P2P systems:

Properties	Descriptions	C/S	P2P
Manageability	How hard is it to keep the system working?	+	-
Information coherence	How authoritative is information in the system?	+	-
Extensibility	How easy is it to grow the systems, to add new resources to it?	=	+
Fault-tolerance	How well can the system handle failures?	-	+
Security	How hard is it to subvert the system?	+/-	-
Resistance to lawsuits	How hard is it for an authority to shut down the system?	-	+
Scalability	How large can the system grow?	+/-	+

### Lookup Problem:

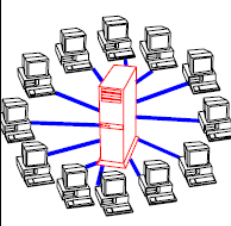
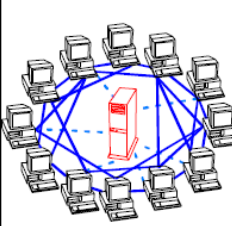
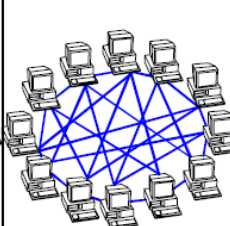
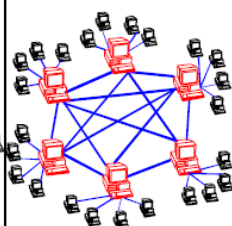
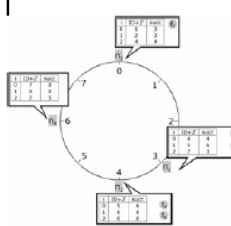
- Locating resources is the central problem for P2P networks in general, and for file sharing in particular
- There are 3 possible solutions for that problem
  - Centralized directory model
    - Perfect example of a hybrid P2P model
    - The index service is provided centrally by a coordinating entity
    - Lookup of existing documents can be guaranteed
    - Index service is “single point of failure”
    - Example: Early Napster
  - Flooded requests model
    - Search request is passed to an predetermined number of peers
    - If they can’t answer the request, they pass it on to various other nodes until a predetermined search depth (TTL) is reached
    - When requesting files have been located, positive search results are sent to the requesting entity
    - Lookup of existing documents can’t be guaranteed
    - System does not scale
    - Example: Gnutella
  - Document routing model
    - Files are not stored on the hard disc of the peers providing them
    - Each peer is assigned responsibility for a set of files
    - When requesting a file a definite function is used to determine associated peer
    - Self organized adaption in the case of entering and leaving peers necessary
    - Example: Chord, Pastry

**P2P Generations, Past and Future**

**General characteristics of 1<sup>st</sup> and 2<sup>nd</sup> generation of P2P systems:**

- Overlay architectures → TCP/IP based
- Decentralized and self organizing (with possibly centralized elements)
- Content is distributed randomly on the network with several replicas
- Content and description is not structured → Content stays at the nodes which bring it into the network
- Initially developed for filesharing
- Content transfer is out-of-band (typically HTTP)

**Architectures of 1st and 2nd generation P2P:**

Client-Server	Peer-to-Peer			
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 Example: WWW	1. Resources are shared between the peers 2. Resources can be accessed directly from other peers 3. Peer is provider and requestor (Servernt concept)			
	Unstructured P2P			Structured P2P
	Centralized P2P	Pure P2P	Hybrid P2P	DHT-Based
	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 Example: Napster	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities Examples: Gnutella 0.4, Freenet	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → dynamic central entities Example: Gnutella 0.6, JXTA	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" Examples: Chord, CAN
				
	1 <sup>st</sup> Gen.		2 <sup>nd</sup> Gen.	

**Characteristics of the overlay topology:**

- Completely independent from physical network, due to abstraction layer TCP/IP
- May include hierarchies (hub network)
- May include centralized elements (star network)
- Separate addressing scheme

**Basic Bootstrapping:**

- Mostly not part of the protocol specification
- It's necessary to know at least one participant of the network. Otherwise no participation possible for a new node
- The address (TCP) of an active node can be retrieved by different means
  - Bootstrap cache: Try to establish one after another connection to a node seen in a previous session
  - Bootstrap server: Connect to a "well-known host" which almost always participants or ask a bootstrap server to provide a valid address of at least one active node
  - Broadcast on the IP layer: use multicast channels or IP broadcasting

**Characteristics of centralized P2P:**

- The central server is the bootstrap server
- The central entity is necessary to provide the service
- All signaling connections are directed to central entity
- The central entity is used to find content, log on to the overlay, register, update the routing tables and update shared content information

**Drawbacks of centralized P2P:**

- Single point of failure → easily attackable
- Bottleneck → potential of congestion
- Central server in control of all peers

**Advantages of centralized P2P:**

- Fast and complete lookup (one hop lookup)
- Central managing / Trust authority
- No keep alive messages necessary beyond content updates

**Definitions of decentralized P2P (pure P2P):**

- Any terminal entity can be removed without loss of functionality
- No central entities at all employ in the overlay
- Peers establish connections between each other randomly

**Basic characteristics of decentralized P2P:**

- Bootstrapping
  - No registration is necessary
- Routing
  - Completely decentralized
  - Reactive protocol → routes to content providers are only established on demand, no content announcements
  - Requests: flooding / Responses: routed
- Signaling connections
  - Stable as long as neighbors don't change
  - Based on TCP
  - Keep alive messages and content search
- Content transfer connections
  - Temporary out-of-band connections over HTTP

**Drawbacks of decentralized P2P:**

- High signaling traffic because of decentralization
- Slow nodes may become bottleneck
- Overlay topology is not optimal as there is no coordinator and no complete view available
- If not adapted to physical structure, delay and total network load increases zigzag routes and loops

**Advantages of decentralized P2P:**

- No single point of failure
- Can be adapted to physical network
- Can provide anonymity
- Can be adapted to special interest groups

**Driving forces behind P2P:**

- Personal computers have capabilities comparable to servers
- Bandwidth is plentiful and cheap

**Reasons against P2P:**

- Law suits against users, software patents, intellectual property, digital right management, best-effort services insufficient for most applications, P2P requires flat rate access, lack of trust, still low bandwidth at end nodes, interoperability, commercialization as the end of P2P, ...

**Distributed Hash Tables (1)**

**Distributed indexing:**

- Data and nodes are mapped into the same address space
- Intermediate nodes maintain routing information for target nodes
  - Efficient forwarding to “destination” (content – not location)
  - Definitive statement of existence of content
- Problems:
  - Maintenance of routing information is required
  - Fuzzy queries not primarily supported (e.g. wildcard searches)

**Comparison of lookup concepts:**

System	Per Node State	Communication Overhead	Fuzzy Queries	No false negatives	Robustness
Central Server	$O(N)$	$O(1)$	✓	✓	✗
Flooding Search	$O(1)$	$O(N^2)$	✓	✗	✓
Distributed Hash Tables	$O(\log N)$	$O(\log N)$	✗	✓	✓

**Challenges for designing DHTs:**

- Equal distribution of content among nodes → crucial for efficient lookup of content
- Permanent adaption of faults, joins, exits of nodes
  - Assignments of responsibilities to new nodes
  - Re-assignment and re-distribution of responsibilities in case of node failure or departure

**Association of data with IDs:**

- Direct storage
  - Content is stored in responsible nodes of the key
  - Inflexible for large content, ok if small amount of data (< 1 KB)
- Indirect storage
  - Nodes in the DHT stores tuples like (key, value)
    - Key = hash(“my data”)
    - Value = real storage address of content (IP, Port)
  - More flexible, but one step more to reach content

**Joining of new node:**

- 1) Calculation of node ID

- 2) New node contact DHT via arbitrary node
- 3) Assignment of a particular hash range
- 4) Copying K/V pairs of hash range (usually with redundancy)
- 5) Binding into routing environment

**Departure of a node:**

- 1) Partitioning of hash range to neighbor nodes
- 2) Copying K/V pairs to corresponding nodes
- 3) Unbinding from routing environment

**Comparison DHT vs. DNS:**

	DNS	DHT
<b>Mapping</b>	Symbolic name → IP Address	Key → value
<b>Topology</b>	Is built on a hierarchical structure with root servers	Does not need a special server
<b>Name space</b>	Names refer to administrative domains	Does not require a special name spaces ( → not bound to particular applications or services)
<b>Functionality</b>	Specialized to search for computer names and services	Can find data that are independently located of computers

**Selected DHT algorithm: Pastry:**

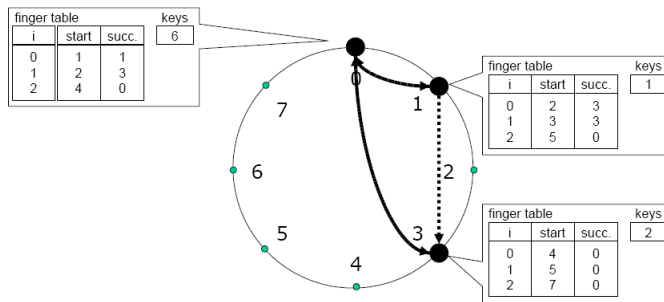
- Identifier space
  - Each node and data item has unique  $l$ -bit ( $l$  typically 128) identifier
  - Keys are located on the node whose node ID is numerically closest to the key
  - Pastry identifiers are string of digits to the base  $2^b$  (typically  $b = 4$ )
- Leaf set
  - Contains nodes close in the ID space
  - $|L|/2$  numerically closest larger node IDs and  $|L|/2$  numerically closest smaller node IDs (typically  $|L| = 16$ )
- Routing table
  - $l/b$  rows with  $2^b - 1$  entries
  - Row  $n$ : nodes that share a  $n$ -digit prefix but whose  $n+1$  digit is different
  - On average, only  $\log_2 b(N)$  rows are populated
- Neighborhood set  $M$ 
  - Nodes close in network locality
- Routing procedure
  - 1) Forward message to a node who share with the key a prefix that is at least one digit ( $b$  bits) longer than the prefix that the key share with the current node
  - 2) If no such node exists, forward the message to a node who is numerically closer to the key
  - 3) Forward message to a node in the leaf set who is numerically closest to the key
- Node arrivals
  - New node with node ID  $X$  asks nearby node  $A$  to route special message to key  $X$
  - Message is routed to node  $Z$ ,  $X$  obtains leaf set from  $Z$  and  $i$ -th row of routing table from  $i$ -th node along the route from  $A$  to  $Z$

**Distributed Hash Tables (2)**

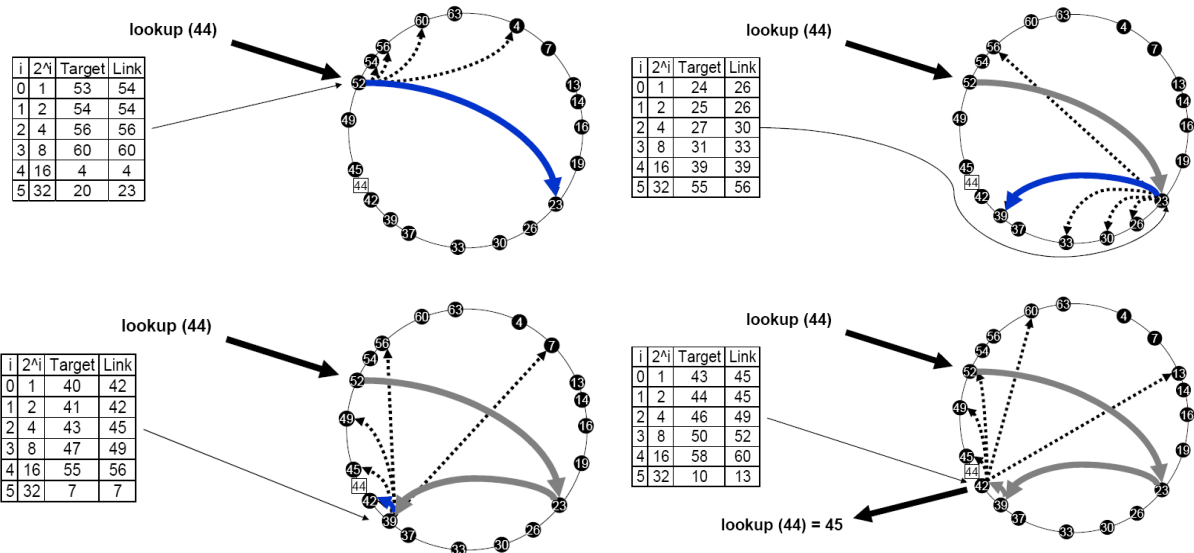
**Selected DHT algorithm: Chord:**

- Keys and Node IDs topology

- K/V pairs are managed by the clockwise next node
- Routing
  - Primitive routing
    - Each node has link to clockwise next node → forward query for key X until successor(X) is found
    - Advantages: simple and little node state  $O(1)$
    - Drawbacks:
      - Poor lookup efficiency:  $O(1/2 * N)$  on average
      - Node failure breaks circle
  - Advanced routing
    - Every node stores link to z next neighbors → forward query for k to farthest known predecessor of k
  - Scalable routing
    - Mix of short and long distance links required
    - Advantages
      - Accurate routing in node's vicinity
      - Fast routing progress over large distances
      - Bounded number of links per node
- Finger table (routing table)
  - Stores  $\log(N)$  links per node
  - Covers exponentially increasing distances → at node n the i-th entry points to successor( $n+2^i$ )



- Routing algorithm
  - Each node n forwards query for key k clockwise to farthest finger preceding k until  $n = predecessor(k)$  and  $successor(n) = successor(k)$



- *Node arrival*
  - 1) *New node picks ID*
  - 2) *Contact existing node*
  - 3) *Construction of finger table*  
*Iterate over finger table rows. For each row query entry point for successor*
  - 4) *Construction of successor list*  
*Add immediate successor from finger table and request successor list from successor*
  - 5) *Retrieve K/V pairs from successor*
- *Complexity*
  - *Messages per lookup:  $O(\log(N))$*
  - *Memory per node:  $O(\log(N))$*
  - *Messages per management action (join/leave/fail):  $O(\log_2(N))$*
- *Advantages*
  - *Theoretical models and proofs about complexity*
- *Disadvantages*
  - *No notion of node proximity and proximity based routing optimizations*
  - *Chord rings may become disjoint in realistic settings*

**Power of Two Choices:**

- *Use of one hash function for all nodes ( $h_0$ ) and multiple hash functions for data ( $h_1 \dots h_d$ )*
  - *Data is stored at one node*
  - *Data is stored at one node and other nodes store pointers*
- *Inserting data*
  - 1) *Results of all hash functions are calculated*
  - 2) *Data is stored on the retrieved node with the lowest load*
- *Data retrieving*
  - *Without pointers*  
*Results of all hash functions are calculated and all of the possible nodes are requested in parallel. One of them will answer*
  - *With pointers*  
*Request only one of the possible nodes. Node can forward request directly to the final node*
- *Advantages: simple*
- *Disadvantages*
  - *Message overhead at inserting data*
  - *With pointers: additional administration of pointers*
  - *Without pointers: message overhead at every search*

**Virtual Servers:**

- *Each node is responsible for several intervals*
- *Rules for transferring a virtual server*
  - 1) *The transfer of an virtual server makes the receiving node not heavy*
  - 2) *The virtual server is the lightest that makes the heavy node light*
  - 3) *If there is no virtual server whose transfer can make a node light, the heaviest virtual server from this node would be transferred*
- *One-to-one*
  - 1) *Light node picks a random ID*
  - 2) *Contacts the node  $x$  responsible for it*
  - 3) *Accepts load if  $x$  is heavy*

- *One-to-many*
  - 1) *Light nodes report their load information to directories*
  - 2) *Heavy node H gets the information by contacting a directory*
  - 3) *H contacts the light node which can accept the excess load*
- *Many-to-many*
  - 1) *Many heavy and light nodes rendezvous at each step*
  - 2) *Directories periodically compute the transfer schedule and report it back to the nodes, which then do the actual transfer*
- *Advantages*
  - *Easy shifting of load*
- *Disadvantages*
  - *Increased administrative and message overhead*
  - *Much load is shifted*

**Thermal-Dissipation-based Approach:**

- *Several nodes are responsible for one interval. A fixed constant  $f$  indicates how many nodes have to act within one interval at least*
- *Procedure*
  - 1) *First node takes random position*
  - 2) *A new node is assigned to any existing node*
  - 3) *Node is announced to all other nodes in the same interval*
  - 4) *Copy of documents of interval → more fault tolerant system*
- *Algorithm*
  - *$2*f$  different nodes in the same interval and nodes are overloaded*
    - *Interval is divided into two intervals*
  - *More than  $f$  but less than  $2*f$  nodes in the same interval*
    - *Release some nodes to other intervals*
  - *Interval borders may be shifted between neighbors*

**Comparison between load balancing strategies:**

- *Without load balancing*
  - *Simple and original but bad load balancing*
- *Power of Two Choices*
  - *Simple, but there are nodes without any load*
- *Virtual Server*
  - *No nodes without any load but higher maximal load than in Power of Two Choices*
- *Thermal-Dissipation*
  - *No node without any load and best load balancing but more effort necessary*

**Redundancy vs. Replication:**

- *Redundancy*
  - *Each data item is split into  $m$  fragments*
  - *$K$  redundant fragments are computed*
  - *Any  $m$  fragments allow to reconstruct the original data*
- *Replication*
  - *Each data item is replicated  $k$  times*
  - *$K$  replicas are store on different nodes*



**“Stabilize” Function:**

- 1) *N asks its successor for its predecessor p*
  - 2) *N checks if p equals n*
- *N refreshes random finger x by (re)locating successor*

## Grids and P2P Systems

**Grid computing:**

- *Application of several computers to a single problem at the same time (usually scientific or technical problem) that requires a great number of processing cycles or access to large amounts of data.*

**Service grids:**

- *Services are offered to the user by many different computers which are organized in a grid architecture*

**Knowledge grid:**

- *Grid is used to share knowledge resources (data mining)*

**Globus Project / Globus Toolkit:**

- *International association dedicated to developing fundamental technologies to build large-scale grid applications, building large scale test-beds for grid research*
- *Globus Toolkit: Open source toolkit for building computing grids*

**Global Grid Forum (GGF):**

- *Standardization initiative by community of users and developers*
- *Promotes the adaption of grid and building of communities*

**Grid service characteristics:**

- *Distributed and network-enabled*
- *Represents computational resources*
- *Dynamic service creation → transient and persistent services*

**Akogrimo Project:**

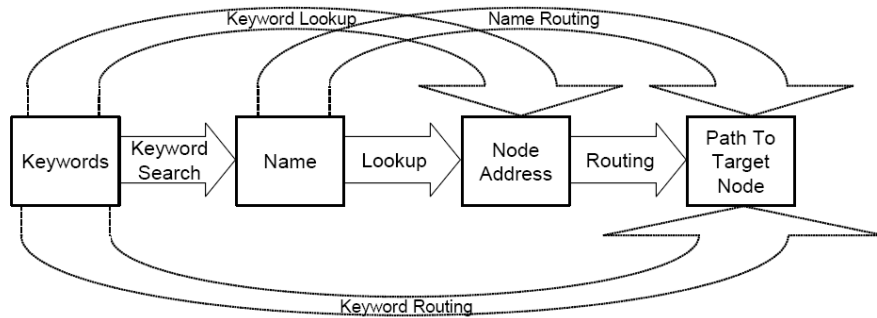
- *Wants to define and realize a mobile grid architecture*
- *Develops new business models for the use of mobile grids*
- *Business scenario: Motivation for travelers and insurance company to know whether consultation of medical facilities is required → early diagnosis*

**Comparison Grid vs. P2P:**

- *Commonalities*
  - *Motivation: Pooling and organizing of resources shared between virtual communities connected via the internet*
  - *Resource sharing: Resources can be located anywhere in the system and are made transparently available*
  - *Overlay structures*
- *Differences*
  - *Target Applications*
    - *Grid: Scientific applications used in a professional context with moderate size, stable and identifiable user set*
    - *P2P: Consumer applications with large scale, dynamic and unknown users*
  - *Resources*

- Grid: Resource pools
- P2P: Single resources
- Structure:
  - Grid: Multipurpose service based infrastructure
  - P2P applications: vertically integrated
  - P2P platforms: Generic support for discovery, naming and resource aggregation

### P2P Search and Scalability



DHT is very suitable for lookup, however not suitable for distributed search → flooding approach is suitable for distributed search, however not scalable.

#### Integrated keyword routing:

- Superior choice for P2P because reasons to decouple names and addresses as in the web are not applicable for P2P, because:
  - Not hierarchical ownership structure available that should be reflected in the name space
  - No slowly updated centralized search engines requiring a separate, faster name resolution system to allow for network changes

#### Definitions and Metrics:

- Symmetry
  - Complete symmetric topologies are applicable to true P2P systems → trees show centralized control
  - Assists load balancing
- Network diameter
  - Number of hops in the overlay structure required to connect from one peer to the most remote peer
  - Average number of hops between any two peers in the overlay network is termed characteristic path length
- Bisection width
  - Number of connections from one part of the overlay to the other part
  - Due to load balancing methods, the maximum throughput of the overlay is proportional to the bisection width
- Node degree
  - Number of overlay links each peers has to maintain
  - Higher node degree preferable due to improved fault tolerance
- Wire length
  - Average round trip delay of an overlay link
  - Low wire length achieves a close-by location of the physical node

- *Extensibility*
  - *Ease of adding resources and growing a system*
- *Scalability*
  - *Strict: Efficiency converges to non-zero value at increasing scale*
  - *Pragmatic: Efficiency/Overhead resource consumption behavior compared to reference system at growing scale*
  - *Is not limited to resources but also other functions, etc.*

**SHARK (Symmetric redundant hierarchy adaption for routing of keywords):**

- *Implements a metadata search functionality → uses a multidimensional metadata structure*
- *Evaluations:*
  - *4 orders of magnitude less traffic than Gnutella*
  - *Logarithmic growth in average node degree*
  - *Logarithmic growth in average number of levels*

## **Web Services and Peer-to-Peer**

**Web Services:**

- *Targeted at machine-to-machine communication*
- *Driven mainly by industry and not academia*
- *Great potential for combination with P2P*

**Key characteristics of web services:**

- *Loose coupling, simple usage/integration, independent of programming language and operating system*

**Techniques for binding the process to the client:**

- *Stubs*
  - *Generated out of WSDL at compile time*
  - *Create local representation of the service*
- *Dynamic proxy*
  - *Local interface definition is needed at compile time*
  - *Generates local representation of the service at run time*
- *Dynamic invocation*
  - *No local representation is needed*
  - *Calls are created completely at run time*

**What can P2P learn from web services?**

- *Security: Apply XML security standards*
- *Service registration: Register and find services*
- *XML: Attach helpful metadata to resources*
- *Interoperability: Standardization*
- *Service orchestration*

**What can web services learn from P2P?**

- *Decentralization: Eliminate central elements like UDDI*
- *Transport Protocols: HTTP is not sufficient for all interaction scenarios → look at flexible P2P communication protocols*
- *Client/Server architecture: Consider scalability as an important attribute of dependability*

- *Bandwidth: XML increases bandwidth usage → intelligent search algorithms must be applied*
- *Security: Decentralization needs new ways for securing access and communication*
- *Maintenance: Maintaining distributed systems is a complex task → dependable maintenance with self-sufficient peers becomes even more complex*

### **Peer-to-peer Market Management**

- *Focus on the economic aspect of P2P networks → Support of real world commercial applications*

#### **Main problems:**

- *Key idea of P2P systems is idealistic (Peers offer services to other peers and each peer contributes as much as it uses from other peers)*
- *Peers are autonomous entities*
  - *Cooperation is unlikely to happen without appropriate incentives for peers to share their resources → Can lead to major degradation of overall performance*
- *Freeriders: Peer which benefits from the effort from other peers, e.g. by downloading or searching for file without contributing any resources or performing any task itself*
- *Existing solutions have weaknesses*
  - *Mainly filesharing oriented*
  - *Sometimes weak security measures*
  - *Not applicable for commercial purpose*
- *Peers may be faulty or even act maliciously*
  - *Frequent joins and leaves of the system*
  - *Loss of messages or stored data*
  - *Deliberate misuse of the system*
  - *Malicious behavior against potential competitors to increase own benefits*
  - *DHTs such as Pastry or Chord have limited support against malicious peers*

#### **The Prisoner's dilemma:**

- *Defection is the dominant strategy, although the total outcome would be higher if both cooperate*

#### **The Tragedy of the Commons:**

- *When individuals overuse the public good in order to maximize their own utility, they do not take into account the external costs (negative externality) that have to be borne by everyone*

#### **Market architecture:**

- *Consists of three models. Each of them describes one aspect of the architecture:*
  - *Market model*
  - *Use model*
  - *Peer model*

#### **PeerMart:**

- *Basic concept*
  - *Each service is traded in double action*
  - *Each auction is mapped onto a cluster of broker peers*
- *Broker set*
  - *The n peers numerically closest to a service ID form a broker set*
- *Matching process*
  - *Peers send price offers to a random subset of f broker peers*

- A slotted time is used to tackle message delay between peers
- Brokers forward candidates for a match, matches determined by major decisions

### **Decentralized Accounting**

#### **Trust based incentives patterns:**

- *Collective pattern*
  - *Collective = set of entities with mutual trust and unconditional cooperation*
  - *The incentive for cooperation stems from being member of the same collective*
- *Community pattern*
  - *Community = group of entities whose incentives for cooperation are based on the trust gained by providing services to other entities of the community*
  - *Good reputation is required in order to consume services of other entities*
  - *The consumer remunerates by increasing its trust in the provider*

#### **Trade based incentives patterns:**

- *Trade based pattern*
  - *Barter trade pattern*
    - *Barter trade is defined as the exchange of services. Hence, the consumer remunerates the provider simultaneously providing a service in return*
    - *Conditions / Consequences*
      - *Exchange goods must be of equal value*
      - *Low transaction costs*
  - *Bond based pattern*
    - *The consumer remunerates the provider by handing over a bond. In this regard, an entity provides a service in order to be promised a service in return*
    - *Conditions / Consequences*
      - *Problem of double spending and forgery*
      - *High transaction costs*

#### **P2P trading scheme:**

- *Combine benefits of barter trade and money → use money only if a peer's balance exceeds a certain threshold*

#### **Types of accounting:**

- *Local accounting*
- *Token-based accounting*
- *Remote accounting*
  - *Central accounting*
  - *Hybrid accounting*
  - *Distributed accounting*
    - *Non-redundant accounting*
    - *Redundant accounting*

#### **Token-based accounting:**

- *To create tokens it is necessary that every member of a super peer group signs the token*
- *It's only possible to pay with your own token → foreign tokens have to be traded against own tokens*
- *Wenn zwischen zwei Peers A und B ein Handel zustande gekommen ist, sendet Peer A Peer B eine Liste mit Token welche er ausgeben möchte. Peer B lässt diese Liste dann von Peer C, welcher der*

Accountant von A ist, überprüfen. Möchte A ein Token zum zweiten mal ausgeben, ist dieses Token bereits von der Liste gestrichen und die Transaktion kann abgebrochen werden. Sind die Token alle gültig, wird dies Peer B von Peer C bestätigt. Peer B bestätigt wiederum an A, welcher daraufhin ein unsigniertes Token an B sendet. Nach Erhalt des Token beginnt die Filetransaktion. Sobald diese beendet ist, sendet A dasselbe Token erneut an B, jetzt jedoch signiert und gültig.

**PeerMint:**

PeerMint is a completely decentralized and secure accounting scheme which facilitates market-based management of P2P applications. The scheme applies a structured P2P overlay network to keep accounting information in an efficient and reliable way. Session mediation peers are used to minimize the impact of collusion among peers. A prototype has been implemented as part of a modular Accounting and Charging system to show PeerMint's practical applicability. Experiments were performed to provide evidence of the scheme's scalability and reliability.

### Hybrid Peer-to-Peer Systems

**Benefits of hybrid P2P systems:**

- Intrinsically better than "pure" approaches when heterogeneity is inherent in the deployed system
- Synergistic combination of techniques → more strengths and less weaknesses than either technique alone
- Meet easier the tradeoffs in conflicting requirements

**Basic characteristics of hybrid P2P:**

- Bootstrapping: registration of each leaf node at the super peer it connects to, i.e. it announces its shared files to the super peer
- Routing
  - Partly decentralized → leaf nodes send request to a super peer → super peer distributes this request in the super peer layer → if a super peer has information about a matching file share by one of its leaf nodes, it sends this information back to the requesting leaf node
  - Routes to content providers are only established on demand. But content announcements from leaf nodes to their super peers (reactive and proactive)

**Hybrid P2P example: Gnutella 0.6:**

- Higher signaling efficiency than pure P2P
- Same reliability (no single point of failure)
- Network organization: an election mechanisms decides which node becomes a super peer or a leaf node (depending on capabilities, network connection, uptime, etc.)

**Other hybrid architectures:**

- JXTA, Brocade, Shark, Omicron

**Drawbacks of hybrid P2P:**

- Still high signaling traffic because of decentralization
- No definitive statement possible if content is not available or not found
- Modem nodes may become bottlenecks
- Overlay topology not optimal as no coordinator and no complete view available
- If not adapted to physical structure, delay and total network load increases zigzag routes and loops
- Can't be adapted to the physical network completely because of hub structures
- Asymmetric load (super peers have to bear significantly higher load)

## Selected Key Topics in P2P

### PlanetLab:

- Large collection of machines spread around the world for distributed system research
- Institutions join, provide 2 nodes at minimum and in exchange, researchers get a small slice of many machines worldwide → high benefit from a small entry fee
- Supports distributed virtualization → each of over 500 network services running in their own slice
- Carries real user traffic
- Supports experimental validation of new services

### CoDeen:

- Users set their internet caches to a nearby high bandwidth proxy that participates in the system.
- Requests to that proxy are then forwarded to an appropriate member of the system that is in charge of the file (should be caching it) and that has sent recent updates showing that it is still alive. The file is forwarded to the proxy and thence to the client.

### Bloom Filters:

Der Filter lernt zunächst sein Vokabular. Hierzu wird mittels einer Hash-Funktion für jeden vorkommenden Wert (beispielsweise für jedes richtig geschriebene deutsche Wort) ein Hash-Wert ermittelt, beispielsweise als Binärzahl. Diese Zahl muss umso länger sein, je größer das Vokabular ist, damit sich die Hash-Werte in aller Regel auch voneinander unterscheiden.

Die Hash-Werte werden nun nacheinander in ein zunächst mit Nullen gefülltes Bit-Array geschrieben, das dieselbe Länge hat, wie jeder Wert. Dort, wo ein Hash-Wert eine 1 enthält, wird eine 1 in das Array geschrieben, bei einer 0 bleibt der bisherige Wert erhalten. Es handelt sich also um eine binäre Oder-Funktion. Damit nicht sehr bald im Array nur noch Einsen stehen, sollte die Hash-Funktion Werte liefern, die überwiegend Nullen enthalten.

Ein Hash-Wert kann nicht mehr gelöscht werden, weil im Nachhinein nicht mehr bekannt ist, ob eine 1 an einer bestimmten Stelle im Array womöglich in mehreren Hash-Werten aufgetaucht ist.

Soll nun überprüft werden, ob ein beliebiges Wort im Vokabular enthalten ist, wird auch dessen Hash-Wert ermittelt. Hat er irgendwo eine Eins, wo im Array eine Null steht, kann das Wort nicht enthalten sein. Ist dies aber nicht der Fall, muss das Wort dennoch nicht zwingend im Vokabular enthalten sein, denn das übereinstimmende Bitmuster kann durch die Überlappung mehrerer anderer Hash-Werte zustande kommen, oder auch dadurch, dass zwei Wörter den gleichen Hash-Wert haben.

### Properties:

- Space Efficiency
- No space constraints → add never fails, but false positive rate increases steadily as elements are added
  - Longer bit vector and fewer insertions are always better
- Simple operations

### Applications:

- Distributed caching, collaboration in overlay and P2P networks, resource routing, packet routing, measurement infrastructures

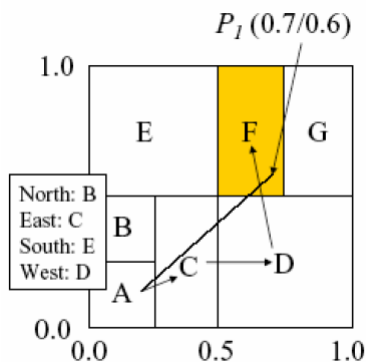
### Bloom Filter Variants:

- Attenuated Bloom Filter
  - Use array of bloom filters to store shortest path distance information
- Counting Bloom Filter

- Each entry in the filter need not be a single bit but rather a small counter
- Delete operation possible → decrementing counter
- Spectral Bloom Filter
  - Extend the data structure to support estimates of frequencies
- Compressed Bloom Filter
  - When the filter is intended to be passed as a message
- Generalized Bloom Filter
  - Two type of hash functions → one which resets bits to 0 and one which sets bit to 1
  - Start with an arbitrary vector
  - In case of collision between the two hash functions, bit is reset to 0 → produces either false positive or false negatives
- Space-Code Bloom Filter
  - Made of  $l$  groups of hash functions each group viewed as a traditional bloom filter

**Selected DHT algorithms: CAN**

- Uses a  $d$ -dimensional Cartesian coordinate space ( $d$ -torus) → each node owns a zone on the torus
- Each node stores IP address and coordinate zone of adjoining zones → node's routing table
- Routing: How to route from node A to point P1?
  - Draw straight line from point in A's zone to P1
  - Follow straight line using neighbor pointers



- In a  $d$ -dimensional space, partitioned into  $n$  equal zones, each node maintains  $2d$  neighbors and the average routing path length is  $(d/4) * (n^{1/d})$
- CAN Node Joining
  - 1) New node finds a node already in CAN
  - 2) New node chooses random point P and sends message to node whose zone contains P (node N)
  - 3) Node N splits its zone and allocates half to new node, transfer of K/V pairs
  - 4) New node learns neighbor set from N
  - 5) N updates its neighbor set to include new node
- CAN Node Departure
  - Graceful node departure
    - Node explicitly hands over zone and (key, value) pairs to one of its neighbors
    - Merge to form valid zone if possible, if not, two zones are temporarily handled by smallest neighbor
  - Node failure
    - Each node periodically sends messages to each of its neighbors
    - Nodes that detects failure initiates takeover mechanisms
    - Takeover mechanism ensures node with smallest volume takes over the zone



## Business P2P Applications

- P2P applications often lack revenue generation → revenue currently only indirect (e.g. ads)

### Revenue Models for Instant Messaging:

- Application Style
  - License fees
  - Optional professional services
- Service Style
  - Subscription fees
    - Undifferentiated → not efficient
    - Fees per log on → not very efficient, hard to realize in pure topology
    - Usage dependent → efficient, only problem-free in C/S topology

### Revenue Models for Digital Content Sharing:

- Application Style
  - License fees
  - Consulting services
- Service Style
  - Legal Owner is not identical with provider
    - Membership/Subscription fees / Fees per log on / Matchmaking fees → legally problematic
  - Legal Owner is not identical with provider but the owner receives compensation
    - Billing step implemented into content exchange → mediator is aggregating as middleman → no clear economic value for owners
  - Legal Owner is identical with provider
    - Differentiated charging and owner is compensated
    - Providers don't sell object but limited rights to its usage

### Revenue Models for Grid Computing:

- Application Style
  - License fees
  - Professional services
- Service Style
  - Compensating the mediator
  - Compensating the provider

### Revenue Models for Collaboration:

- Application Style
  - Licensing models
  - High demand for professional services
- Service Style
  - Undifferentiated → not efficient
  - Fees for buddy list / catalogue service / etc. → not very efficient and hard to realize in pure topology
  - Transaction-based fees → efficient, but only problem-free in C/S topology

## Mobile and Collaborative P2P Systems

### Ad-hoc networks:

- *Self configuring, infrastructure free, wireless, unpredictable terminal mobility, limited radio transmission range*

**General problems:**

- *Low data rates*
- *Temporary loss of connection*
- *High delay and jitter*
- *Limited resources of mobile devices (battery power, computational power, memory, bandwidth, etc.)*
- *Application decoupled from networks*

**Structural Differences of Ad-Hoc Nets vs. the Internet:**

- *No dedicated router → routing via end systems*
  - *Non-predictable router behavior*
  - *Continuous changes of topology*
- *No global reachability → groups of local networks*
- *Environmental effects and scarce resources*
- *Reactive behavior*

**Design Requirements for Mobile P2P:**

- *Unnecessary transmissions have to be avoided*
- *Loops of the physical layer have to be avoided*
- *Low bandwidth in regions with a high node density has to be expected*
- *Low signal quality in regions with low node density has to be expected*

**adPASS:**

- *Disseminate digital advertisements according to user preferences*
- *Bonus point reward for all people carrying the ad to a buyer*
- *Procedure:*
  - 1) *Vendors disseminate digital ads via radio to customers*
  - 2) *Customers pass on the ad when meeting in the street*
  - 3) *Customers returns to store and buys the product*
  - 4) *Vendor informs mediator about the bonus points*
  - 5) *Customers sync their bonus points via internet*

**Building Blocks for Mobile P2P Systems:**

- *Presence awareness service (keep alive message)*
- *Message exchange service*
- *Information filtering service*
- *Information distribution service (subscribe for specific information)*
- *Security service*
- *Identity management service*
- *Service for incentive schemes*
- *Reputation service*
- *User notification service*