

Introduction and Application Areas

Introduction

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 “servers” at the same time*
- *Typically located at the edges of the network (end-to-end principle)*

End-to-End principle

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

Differences between Client/Server Systems and P2P Systems:

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

Application Areas

Conventional Classification:

- *File Sharing (Napster, Gnutella, Freenet, etc.)*
- *Grid Computing (SETI@home)*
- *Instant Messaging (ICQ, AIM)*
- *Collaboration (Groove Workspace)*

Classification by Means of Sharing Resources:

- *Information*
- *Files*
- *Bandwidth*
- *Storage space*
- *Processor cycles*

Information

Presence Information:

- *Provides information about which peers and which resources are available in the network*

Document Management:

- *Is usually organized centrally – but a large portion of the documents are created among Desktop PCs without a central repository having any knowledge of their existence*
- *P2P network could be used to create a connected repository from the local data on the individual peers*

Collaboration:

- *P2P groupware avoid additional administrative task and central data management*
 - *All of the data created is stored on each peer and is synchronized automatically*
 - *Users can set up shared working environment for virtual teams (shared spaces)*
 - *Users can invite other users to work in these teams*

Files

File Sharing:

- *Store content at individual nodes instead of one central place*
- *Peers who downloaded files subsequently make them available for other peers*

Lookup Problem:

- *Locating resources is the central problem for P2P networks in general, and for file share in particular*
- *Solutions:*
 - *Centralized directory model*
 - *Perfect example of a hybrid P2P model*
 - *The index services 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 cannot answer the request, the pass it on to various other nodes until a predetermined search depth (TTL)*
 - *When requested files has been located, positive search results are sent to the requesting entity*
 - *Lookup of existing documents cannot be guaranteed*
 - *System does not scale*
 - *Example: Gnutella*
 - *Document routing model*
 - *Files are not stored on the hard disk 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*

Bandwidth

Problem with centralized approach:

- *Spontaneous increases in demand exert a negative influence on the availability of the files since bottlenecks and queues develop*

P2P based approach:

- *Achieve increased load balancing by taking advantage of transmission routes which are not being fully exploited*

Storage Space

Disadvantages of centralized storage:

- *Inefficient use of the available storage*
- *Additional load on the network*
- *Necessity for specially trained personnel*
- *Additional backup solutions*

P2P Storage Networks:

- *A cluster of computers, formed on the basis of existing networks, which share all storage available in the network*
- *Organization:*
 - *Each peer must make available some of its own storage, or pay a fee*
 - *Corresponding to its contribution, each peer is assigned a maximum volume of data which can be added to the network*
 - *Storing the file and searching for it in the network takes place in the manner described in the document routing model*

Processor Cycles

Using P2P applications to bundle processor cycles:

- *Forming a cluster of independent, networked computers in which a single computer is transparent and all networked nodes are combined into a single logical computer → Achieve computing power which even the most expensive super computers can scarcely provide.*
- *Also known as **Grid Computing***
- *Example: SETI@home*

P2P Generations, Past and Future

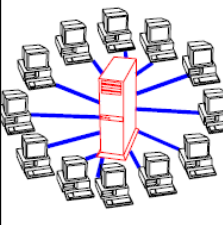
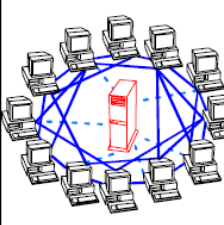
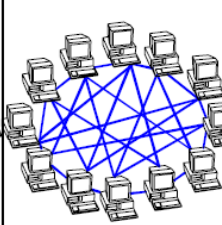
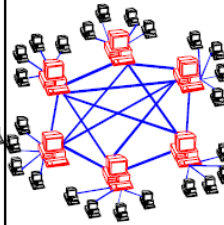
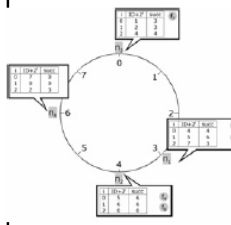
General Characteristics of P2P Systems

General characteristics of 1st and 2nd 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 file sharing*

- Content transfer is out of band (typically HTTP)

Architectures of 1st and 2nd Generation P2P:

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

Peer:

- Node actively participating in the overlay
- Content provider and content requestor and router in the overlay
- Identifiable by a General Unique ID

Characteristics of the overlay topology:

- Completely independent of physical network, due to abstraction layer TCP/IP
- May include hierarchies (Hub network)
- May include centralized elements (Star network)
- Separate addressing scheme

Basic routing behavior:

- Request messages:
 - Header consists of: Hop-counter, TTL, and GUID
 - TTL determines along how many hops a message may be forwarded
 - Are flooded in the overlay network: Every node forwards every incoming message to all neighbors except the neighbor, it received the message from
 - Request message terminate if the same message with same GUID is received more than once (→ Loop) or if Hop-counter = TTL
- Response message:
 - Header consists of: Hop-counter, TTL and GUID
 - Are routed back on the same way to the requestor, the request message was transmitted to the responding peer
 - Consequences:
 - Every peer has to store the GUID of each request for a certain amount of time

- No flooding to save resources

Basic Bootstrapping:

- Mostly not part of the protocol specification
- 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
 - 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

Centralized P2P Networks

Definition of centralized P2P:

- All peers are connected to central entity
- Peers establish connections between each other on demand to exchange user data
- Central entity is necessary to provide the service
- Central entity is some kind of index/group database
- Central entity is lookup/routing table

Basic characteristics of centralized P2P:

- The central server is the bootstrap server
- Central entity can be established as a server farm, but one single entry point = single point of failure
- All signaling connections are directed to central entity
- Central entity is used to:
 - Find content
 - Log on to the overlay
 - Register
 - Update the routing tables
 - Update shared content information
- Communication:
 - Peer and Central Entity: P2P protocol
 - Peer and Peer: HTTP

Example: Napster

- Participants:
 - Napster Hosts/Peers
 - Client Service: Login and Data/Download requests
 - P2P Service: Data transfer
 - Napster index service: Pure server
- Procedure:
 - 1) Connect to Napster server
 - 2) Upload your list of files to server
 - 3) Query index server with a list of keywords to search the full list with
 - 4) Select “best” of correct answers

5) Connect to providing host/peer

- Drawbacks:
 - Single Point of Failure → Easily attackable
 - Bottleneck
 - Potential of congestion
 - Central server in control of all peers
- Advantages:
 - Fast and complete lookup (one hop lookup)
 - Central managing/trust authority
 - No keep alive necessary, beyond content updates

Decentralized P2P Networks

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:
 - Via bootstrap-server (host list from a web server)
 - Via peer-cache (from previous sessions)
 - Via well-known host
 - No registration
- 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 do not change
 - Based on TCP
 - Keep-alive
 - Content search
- Content transfer connections
 - Temporary
 - Based on HTTP
 - Out of band transmission

Example: Gnutella 0.4

- Participants:
 - Gnutella peers/servants
 - Router service
 - Flood incoming requests (keep alive and content)
 - Route responses for other peers (keep alive and content)
 - Data/download requests
 - Lookup service

- Initialize data and keep alive requests
- “Server” service
 - Serve data requests over HTTP
- Procedure
 - 1) Connect to at least one active peer (address received from bootstrap)
 - 2) Explore your neighborhood (PING/PONG commands)
 - 3) Submit query with a list of keywords to your neighbors
 - 4) Select “best” of correct answers
 - 5) Connect to providing host/peer
- Drawbacks
 - High signaling traffic, because of decentralization
 - Modem nodes (slow nodes) may become bottlenecks
 - Overlay topology not optimal, as
 - No complete view available
 - No coordinator
 - If not adapted to physical structure delay and total network load increases zigzag routes and loops
- Advantages:
 - No single point of failure
 - Can be adapted to physical network
 - Can provide anonymity
 - Can be adapted to special interest groups

Past and Future

From ARPANET to P2P:

- 1960: Establishment of ARPANET
 - Every host treated equally
 - Virtual network matched the physical network to a large extent
 - Client/Server mode with no decentralized search and storage
 - Central steering committee to organize the network
- 1979: UseNet Protocol
 - Newsgroup organization to organize content
 - Self organizing approach to add and remove newsgroup servers
 - Application itself is still a Client/Server application
- 1990: General public joins Internet
 - Lot of applications are following the Client/Server approach (Email, WWW, FTP)
 - Security concerns resulted in a partitioned Internet by firewalls

Driving Forces behind P2P:

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

Napster:

- May 1999: Introduction of Napster
- Users establish a virtual network, entirely independent from physical network and administrative authorities and restrictions
- December 1999: RIAA (Recording Industry Association of America) files a lawsuit against Napster Inc.
- July 2001: Napster has to stop the operation of the Napster server → Network breaks down

Gnutella and its relatives:

- *March 2000: Gnutella is released as open source project*
- *Additionally to servant functionality, the peers also take over routing tasks*
- *Fully decentralized, no central lookup server*
- *October 2000: introduction of hierarchical routing layers (2nd generation of P2P) → Increases the scalability significantly*
- *2001: 3rd generation of P2P → Usage of proactive routing algorithms based on distributed hash tables (DHT) → called structured P2P networks*

Traffic measurement:

- *Approximately 70% of Internet Traffic is caused by P2P applications*
- *Also unidentified traffic and data transfers increased. This could also be P2P applications because a lot of filesharing applications use port 80 to send files or try to hide data transfer (e.g. via port hopping).*

Research Challenges in Peer-to-Peer Systems and Applications

Applications:

- *Telephony, Streaming, Intra-organization resource sharing, multiplayer games, P2P-Spam, video conferences, trustworthy computing, ...*

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, ...*

Research Focus:

- *Anonymous but still secure e-commerce, real time P2P data dissemination, distributed search mechanisms, mobile P2P, semantic queries, realistic P2P simulator, reduction of signaling traffic, reliable messaging, incentives market mechanisms, ...*

Distributed Hash Tables (1)

Distributed Management and Retrieval of Data

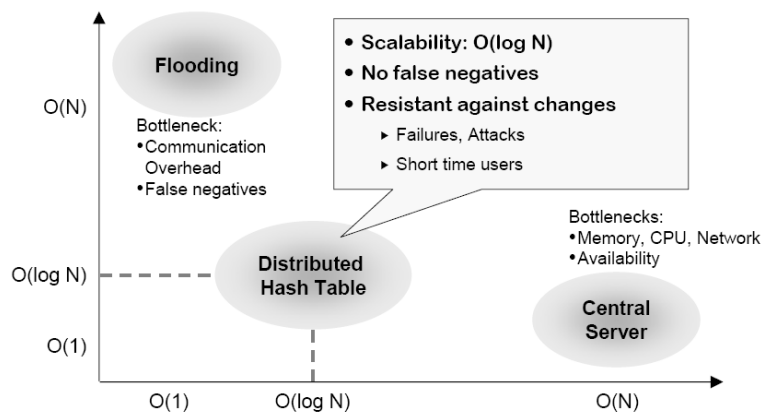
Essential challenge in (most) P2P systems:

- *Location of data item among systems distributed → where shall the item be stored by the provider and how does a requester find the actual location of an item?*
- *Scalability: keep the complexity for communication and storage scalable*
- *Robustness and resilience in case of faults and frequent changes*

3 different strategies to store and retrieve data items in a distributed system:

- *Central server*
 - *Procedure:*
 - 1) *Node A (provider) tells server that it stores item D*
 - 2) *Node B (requester) asks server S for the location of D*
 - 3) *Server S tells B that node A stores item D*
 - 4) *Node B requests item D from node A*

- **Advantages:**
 - Search complexity of $O(1)$
 - Complex and fuzzy queries are possible
 - Simple and fast
- **Problems:**
 - No scalability: $O(n)$ node state in the server and $O(n)$ network and system load of server
 - Single point of failure or attack
 - Non-linear increasing implementation and maintenance cost
 - Not suitable for systems with massive numbers of users
- **Flooding search**
 - No information on location of a content
 - Content is only stored in the node providing it
 - Necessity to ask as much systems as possible/necessary
 - Procedure:
 - 1) Node B (requester) asks neighboring node for item D
 - 2) Nodes forward request to further nodes (breadth-first search/flooding)
 - 3) Node A (provider of item D) send D to requesting Node B
- **Distributed indexing:**



- Data and nodes are mapped into the same address space
- Intermediate nodes maintain routing information to target nodes
 - Efficient forwarding to “destination” (content – not location)
 - Definitive statement of existence of content
- **Problems:**
 - Maintenance of routing information 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	$O(\log N)$	$O(\log N)$	✗	✓	✓

Fundamentals of Distributed Hash Tables

Challenges for designing DHTs:

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

Sequence of operations:

- 1) Mapping of nodes and data into same address space
 - Peers and content are addressed using flat identifiers
 - Mapping of data and nodes into an address space (with hash function)
 - Association of parts of address space to DHT nodes → Nodes are responsible for data in certain parts of the address space
 - Often with redundancy (overlapping of parts)
 - Real (underlay) and logical (overlay) topology are (mostly) uncorrelated
 - Association may change since nodes may disappear
- 2) Storing/Looking up data in the DHT
 - Start lookup at arbitrary node of DHT
 - Routing to requested data item (key)
 - Key/Value pair is delivered to requester
 - Requester analyzes K/V pair (and downloads data from actual location in case of indirect storage)

Association of Data with IDs:

- Direct storage
 - Content is stored in responsible node for 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 is often real storage address of content: (IP, Port) = (127.0.0.1, 4611)
 - More flexible, but one step more to reach content

DHT Mechanisms

Joining of a new node:

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

Failure of a node:

- Use of redundant K/V pairs
- Use of redundant/alternative routing paths
- K/V usually still retrievable if at least one copy remains

Departure of a node:

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

DHT Interfaces

Generic interface of distributed hash tables:

- Provisioning of information
 - Publish(key, value)
- Requesting of information (search for content)
 - Lookup(key)

Comparison DHT vs. DNS:

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

Properties of DHTs:

- Use of routing information for efficient search for content
- Keys are evenly distributed across nodes of DHT
 - No bottlenecks
 - A continuous increase in number of stored keys is admissible
 - Failures of nodes can be tolerated
 - Survival of attacks possible
- Self-organizing system
- Simple and efficient realization
- Supporting a wide spectrum of applications
 - Flat (hash) key without semantic meaning
 - Value depends on application

Selected DHT Algorithm: Pastry

Identifier space:

- Each node and data item has unique I -bit identifier
 - I typically 128
 - Can be calculated from IP address or public key, and data item using secure hash function
- 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 $b = 4$)

Routing information:

- Leaf Set L
 - Nodes close in the ID space
 - $|L|/2$ numerically closest larger node IDs, and $|L|/2$ numerically closest smaller node IDs
 - $|L|$ typically 16

- *Routing Table R*
 - *l/b rows with $2^b - 1$ entries*
 - *Row n: Nodes that share an n-digit prefix, but whose n+1st digit is different*
 - *On average, only $\log_2 b(N)$ rows are populated*
- *Neighborhood Set M*
 - *Nodes close in network locality*

Pastry Routing Table:

NodeId 10233102			
Leaf set			
	SMALLER	LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232
Routing table			
-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	
Neighborhood set			
13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

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 nodes exists, forward message to a node who is numerically closer to the key
- 3) Forward message to a node in the leafset 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

Node failures:

- Leaf set nodes periodically exchange keep alive messages
- If a node does not respond for a time T, it is declared dead
- Members of the leaf set are notified and update their leaf set

Common API for Structured P2P Overlay:

- *forward(RouteMessage)*
 - called just before a message is forwarded
- *deliver(Id, Message)*
 - called when a message is received
- *update(NodeHandle, boolean)*
 - called when a node's leafset changes
- *route(Id, Message, NodeHandle)*
 - send a message to a node numerically closest to an Id
 - NodeHandle can serve as a hint

Distributed Hash Tables (2)

Selected DHT Algorithm: Chord

Keys and Node IDs Topology:

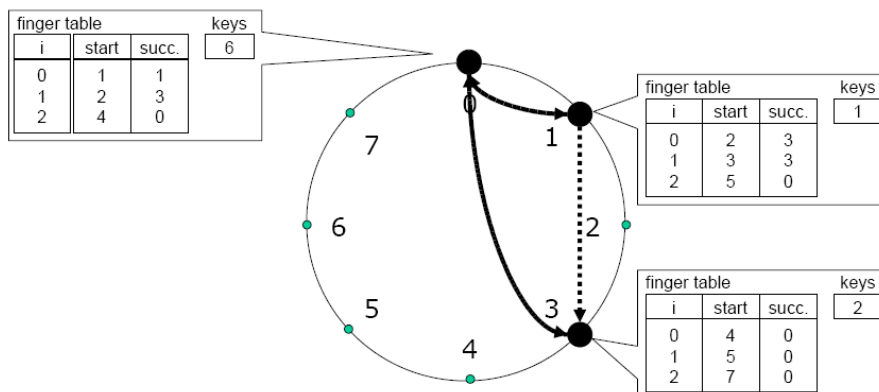
- l -bit identifiers, i.e. integers in range $0 \dots 2^l - 1$
- (Key, Value) Pairs are managed by clockwise next node

Routing:

- Primitive routing
 - Each node has link to clockwise next node
 - Forward query for key x until $\text{successor}(x)$ is found
 - Return result to source of query
 - Advantages
 - Simple
 - Little node state $O(1)$
 - Drawbacks
 - Poor lookup efficiency: $O(1/2 * N)$ hops on average
 - Node failure breaks circle
- Advanced routing
 - Every node stores links to z next neighbors
 - Forward queries for k to farthest known predecessor of k
 - For $z = N$ there is a fully meshed routing system
 - Drawbacks
 - Lookup efficiency: $O(1)$
 - Per-node state: $O(n) \rightarrow$ Poor scalability
- 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

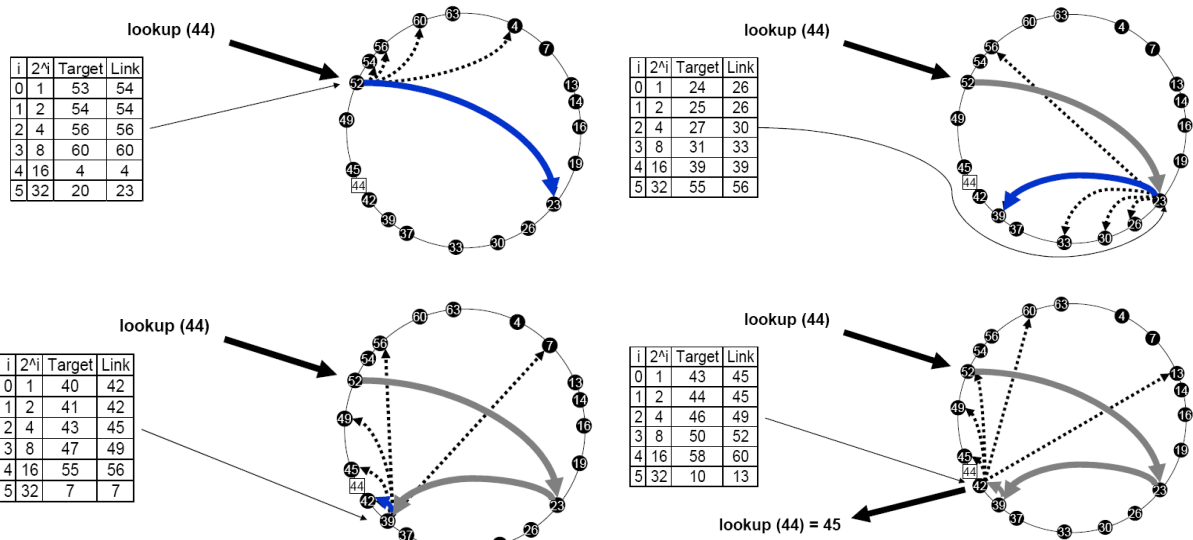
Chord's routing table (finger table):

- Stores $\log(n)$ links per node
- Covers exponentially increasing distances
 - At node n the i -th entry points to $\text{successor}(n + 2^i)$ (i -th finger)



Chord's routing algorithm:

- Each node n forwards query for key k clockwise
 - To farthest finger preceding k
 - Until $n = predecessor(k)$ and $successor(n) = successor(k)$
 - Return $successor(n)$ to source of query



Self-Organization:

- Handle changing network environment
 - Failure of nodes
 - Network failures
 - Arrival of new nodes
 - Departure of participating nodes
- Maintain consistent system state for routing
 - Keep routing information up to date
 - Failure tolerance required for all operations

Failure tolerance of routing:

- Finger failures during routing
 - Query can't be forwarded to finger
 - Forward to previous finger
 - Repair mechanism:
 - Replace finger with its successor
 - Active finger maintenance
 - Periodically check liveness of fingers
 - Replace with correct nodes on failures
- Successor failures during routing
 - Last step of routing can return failed node to source of query
 - Repair mechanism:
 - Store n successors in successor list \rightarrow if $successor[0]$ fails use $successor[1]$ etc.
 - Active maintenance of successor list
 - Periodic checks similar to finger table maintenance
 - Crucial for correct routing

Node Arrival:

- 1) New node picks ID
 - Random ID
 - ID generated through hash function
 - Based on load on existing nodes
 - Base on geographic location
- 2) Contact existing node
 - Controlled flooding
 - DNS aliases
 - Published through web
- 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 key/value pairs from successor

Summary:

- Complexity
 - Message 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
 - Simple & flexible
- Disadvantages
 - No notion of node proximity and proximity-based routing optimizations
 - Chord rings may become disjoint in realistic settings

Storage Load Balancing in Distributed Hash Tables

Without load balancing, the optimal distribution (equal) of documents across nodes can't be reached. There are significant differences in the load of nodes.

Several techniques to ensure an equal data distribution

- Power of Two Choices
 - Use of one hash function for all nodes (h_0) and multiple hash functions for data ($h_1 \dots h_d$)
 - d = number of possible locations of data items
 - Two options of data storage
 - Data is stored at one node
 - Data is stored at one node and other nodes store a pointer
 - Inserting data
 - 1) Results of all hash functions are calculated
 - 2) Data is stored on the retrieved node with the lowest load
 - 3) (Other nodes stores pointers)
 - Data retrieving
 - Without pointers
 - 1) Results of all hash functions are calculated
 - 2) Request all of the possible nodes in parallel
 - 3) One node will answer

- *With pointers*
 - 1) *Request only one of the possible nodes*
 - 2) *Node can forward the 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 (virtual server)*
 - *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 interval*
 - 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*
- *A simple Address-space and Item Balancing*

- Each node has a fixed set of $O(\log(n))$ possible positions (virtual nodes)
- Each node chooses exactly one of those virtual nodes. This position become active

Comparison between load balancing strategies

- Without load balancing
 - Simple and original
 - Bad load balancing
- Power of two choices
 - Simple
 - Low load
 - Nodes without any load
- Virtual server
 - No nodes without any load
 - Higher maximal load than Power of Two Choices
- Thermal-Dissipation
 - No nodes without any load
 - Best load balancing
 - More effort

Reliability in Distributed Hash Tables

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

Reliability of Data in Chord:

- The reliability of data is an application task
- Replicate inserted data to the next f other nodes
- Chord informs application of arriving or failing nodes
- Advantages
 - After failure of a node its successor has the data already stored
- Disadvantages
 - Nodes stores f intervals \rightarrow more data load
 - After breakdown of a node a new successor has to be found and the data has to be replicated to the next node \rightarrow more message overhead at breakdown
 - Stabilize function has to check every successor list \rightarrow more message overhead

Multiple nodes in one interval:

- A fixed positive number f indicates how many nodes have to act within one interval at least

- Nodes have only to store pointers to nodes from the same interval
- In case of failure, no copy of data needed because data are already stored within same interval
- Procedure
 - 1) First node takes a random position
 - 2) A new node is assigned to any existing node
 - 3) Node is announced to all other nodes in the same interval
- Effects
 - Reliability of data
 - Better load balancing
 - Higher security
- Advantages
 - In case of failure it is not necessary to copy data
 - Rebuild intervals with neighbors only if critical
 - Request can be answered by different nodes
- Disadvantages
 - Less number of intervals as in original Chord

Grids and P2P Systems

Overview and Terminology

Idea to share/combine resources from different locations → Resource sharing in networked infrastructures → Resource pool which forms a Virtual Organization (VO)

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.

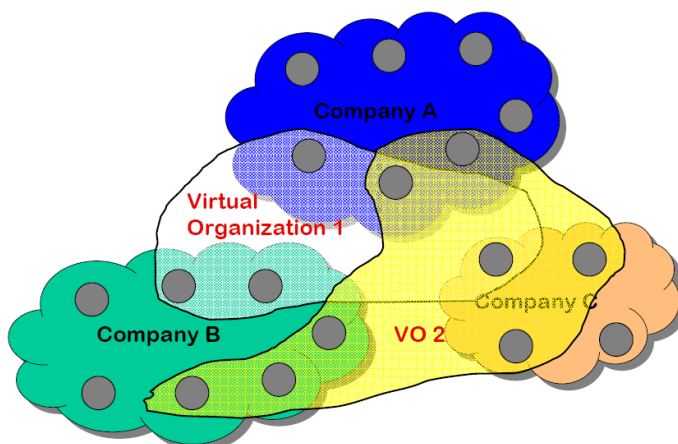
Service grids:

- 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)

Virtual Organizations



Topologies:



Lifecycle of a VO:

- 1) Identification
 - Identify capabilities of the needed services
 - Use them for service discovery
 - Select services and also organizations
- 2) Formation
 - Set up relationship enabling the interoperation of the services
 - Generate workflow descriptions
 - Establish the orchestration/aggregation service
- 3) Operation
 - Services within the VO can communicate
 - The generated workflows get executed
 - Control mechanisms supervise the operation
- 4) Dissolution
 - Clean up everything from the formation phase

Grids

Layered Hourglass Model:

- Fabric Layer
 - Common access to shared resources via a unified interface
- Connectivity Layer
 - Communication & authentication protocols
- Resource Layer
 - Resource management operations
- Collective Layer
 - Coordination of different resources
- Application Layer
 - Application modules for VO

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 grind and building of communities
- Developed standards
 - Open Grid Services Architecture (OGSA)
 - Well defined architecture and interfaces
 - Basic concept: Service-oriented architecture → everything is a service → described with the Web Services Description Language (WSDL)
 - Open Grid Service Infrastructure (OGSI)
 - Infrastructure layer for OGSA. OGSI takes the statelessness issues (along with others) into account by essentially extending Web services to accommodate grid computing resources that are both transient and stateful.

Standardization requirements:

- Service orientation to virtualize resources
- Defined standard behavior and interfaces for services enabling interaction
- System comprises of (typically few) persistent services and (potentially many) transient services

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:

- E-health and tourism domains
- Motivation for travelers and insurance company to know whether consultation of medical facilities is required → Early diagnosis
- Economic Potential
 - Insurance company
 - Understandable product
 - Differentiation
 - Travelers
 - Individual needs reflected (customization)
 - Medical advice in a homelike manner
 - Mobile network operators
 - Network access
 - Further services (customer management, billing)
 - Content providers
 - Expose content in pieces only (keep control)

Comparisons

Grid and 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*

Grid and P2P → Converging Concepts:

- *Grids are developing into an all purpose computational resource infrastructure*
 - *Accounting, IPR protection, scalability, dependability, trust, fault-tolerance, self-organization, self-configuration, self-healing*
 - *P2P mechanisms for the Grid!*
- *P2P is developing into more complex systems, providing sophisticated services with higher quality of service*
 - *Grid-like services! → But a more platform based approach is still necessary to become a Grid*

Summary

- *Grid developed in scientific community addressing resource sharing in Virtual Organizations*
- *P2P developed by file sharing community address the free exchange of content*

P2P Search and Scalability

Introduction and Motivation

P2P lookup by unique ID:

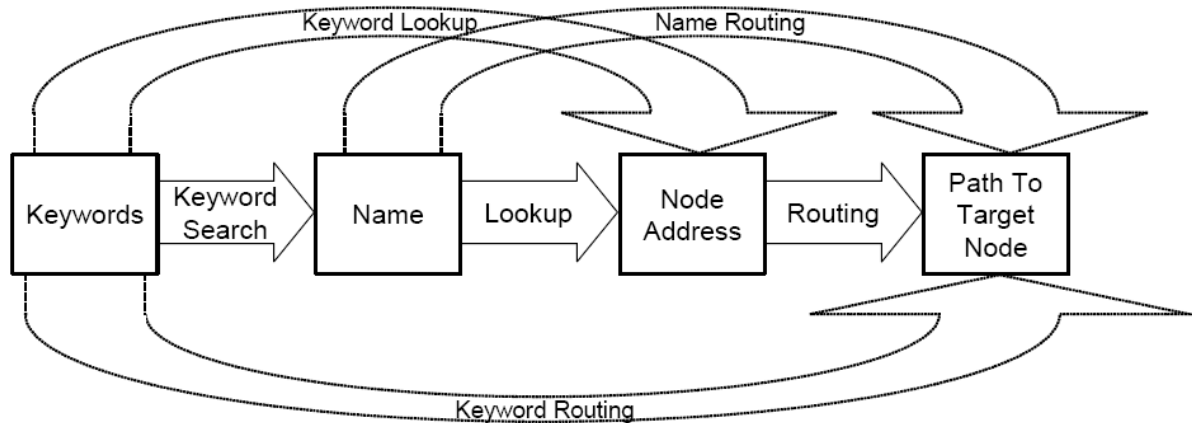
- *DHT very suitable for lookup, however not suitable for distributed search*

P2P Search by keyword or metadata:

- *Flooding approach suitable for distributed search, however not scalable*

Search and lookup:

- *Search: Refers to a wide range of operations and values stored in the network (uni- and multidimensional search, full text search, aggregate operations)*
- *Lookup: Refers to finding the node hosting data for a particular ID*



Search design options:

- Integrated keyword routing
 - Superior choice for P2P
 - More efficient
 - Dynamic rerouting based on keywords reflects the fast changing nature of P2P networks
 - 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
- Separate keyword search and name routing

Definition 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

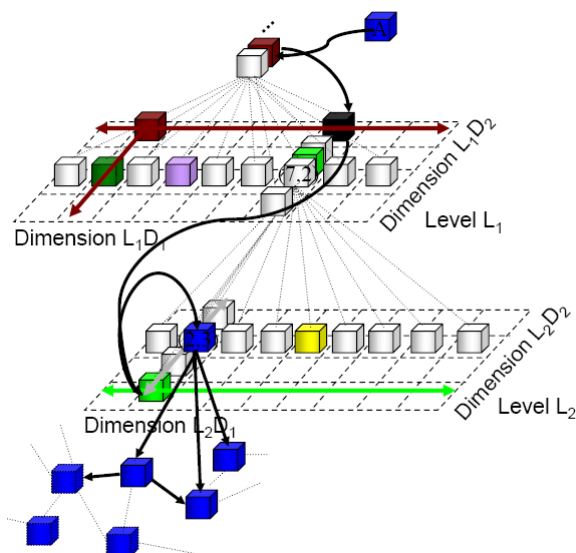
- *Extensibility*
 - *Ease of adding resources and growing a system*
- *Scalability*
 - *Strict: Efficiency converges to non-zero value at increasing scale*
 - *Pragmatic: Efficiency/Overhead resource consumption behavior compared to reference system at growing scale*
 - *Is not limited to resources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

Assessment Scheme

*/** nothing important ***

SHARK

- *Symmetric redundant hierarchy adaption for routing of keywords*
- *Implements a metadata search functionality*
- *Uses a multidimensional metadata structure*



Group Splitting:

- *Group splitting when too large (too many objects within a single group)*
- *Locally self-organizing*

Evaluations:

- *4 orders of magnitude less traffic than Gnutella*
- *Logarithmic growth in average node degree*
- *Logarithmic grows in average number of levels*

Summary and Conclusions

- *Definition of metrics and scales allows for efficient comparison and dedicated design of “competitive” search and lookup schemes*
- *New search scheme SHARK with advantages*
 - *Query traffic 4 orders of magnitude less than Gnutella*

- *Usability limitations due to structuring possible*
- *Large potential application domain, e.g. P2P trading, file sharing, grid resource discovery*

Web Services and Peer-to-Peer

Introduction

Web services:

- *Provide “some functionality” over the internet*
- *Accessed using well-defined interface*
- *Targeted at machine-to-machine communication*
- *Driven mainly by industry and not academia*
- *Platform to implement a service-oriented architecture (SOA)*
- *Is described in a machine-processable format (specially WSDL)*
- *➔ Great potential for a combination with P2P*

Architecture and Important Standards

Key characteristics:

- *Loose coupling*
- *Simple usage/integration*
- *Independent of programming language*
- *Independent of operating system*

Three participants of a web service scenario:

- *Service provider*
 - *Creates and publishes an interface of a service using WSDL*
 - *Contributes the actual implementation of the service*
- *UDDI service registry*
 - *Collects and categorize interface definitions*
 - *Offers interface definitions to users (directory listing, search engines)*
- *Service requestor*
 - *Is the client (machine or human)*

Three 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*

XML:

- *Extensible Markup Language*

- *Key to platform and programming language neutral data exchange*
- *XML namespace*
 - *XML allows arbitrary strings to be used as local element names (URI: Uniform Resource Locator)*
 - *Namespace provides means to avoid naming conflicts → namespace + local name = qualified name of an element*
- *XML scheme*
 - *Building block for creating modular XML documents*
 - *Makes syntactical restrictions for XML elements*
 - *Used to define structural patterns restricting range of values, define simple and complex data types, enumerations and choices*

WSDL:

- *Web Services Description Language*
- *XML based format for describing web service's interface*
- *Defines*
 - *Types (data types exchanged by the messages)*
 - *Messages (exchanged between client and service)*
 - *Every message consists of parts that are of a certain type*
 - *Port types*
 - *Contain a set of operations provided by the web service*
 - *An operation can have input, output and failure message*
 - *Bindings*
 - *Assigns a data encoding format and a transport protocol to the web service's operations*
 - *More than one protocol binding is possible*
 - *Usually SOAP over HTTP is used*
 - *Service*
 - *Defines for each binding a port as the actual end point*
 - *The location is the physical address of the web service*
 - *A service can have multiple ports*

SOAP:

- *Responsible for data encoding and data exchange*
- *Is independent of an underlying transport protocol*
- *Message container with*
 - *Envelop*
 - *Mandatory XML root element which contains SOAP header and body*
 - *Header*
 - *Optional element which carries additional information for recipients and intermediaries*
 - *Body*
 - *Carries application specific information for the final recipient*

HTTP:

- *Hypertext Transfer Protocol*
- *Stateless application-level protocol for data exchange*
- *Primarily used by web browser to access web servers, wider applicability through extensions*

UDDI:

- *Universal Description, Discovery and Integration*
- *Directory service providing registration and search capabilities*
- *Supports metadata annotations for service categorization*

WS-:*

- *Additional emerging standards to retrofit web services for commercial and secure usage*

Service Orchestration

- *Large business processes need combination of different services → there are several languages emerging for that:*
 - *XDPL, BPML, WSCI, ebXML, BPEL4WS*

Comparison on Peer-to-Peer and Web Services

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*

Resulting Architecture

- *P2P in a closed and secure system*
- *Web services access for the outside world*

Peer-to-peer Market Management

- *Focus on the economic aspect of P2P networks*
- *Support of real world commercial applications*
- *Decentralized marketplace with generic service support*
- *Observations*
 - *Selfish behavior of peers, no cooperation*
 - *Need for incentives*

Requirements

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 files without contributing any resources or performing any tasks itself*
 - *Two types of Freeriders*
 - *Peers not providing resources like files or hardware resources*
 - *Peers not providing base functionality like forwarding search requests*
- *Existing solutions have weaknesses*
 - *Mainly file-sharing oriented*
 - *Sometimes weak security measures*
 - *Not applicable for commercial purposes*
- *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*

Functional Requirements:

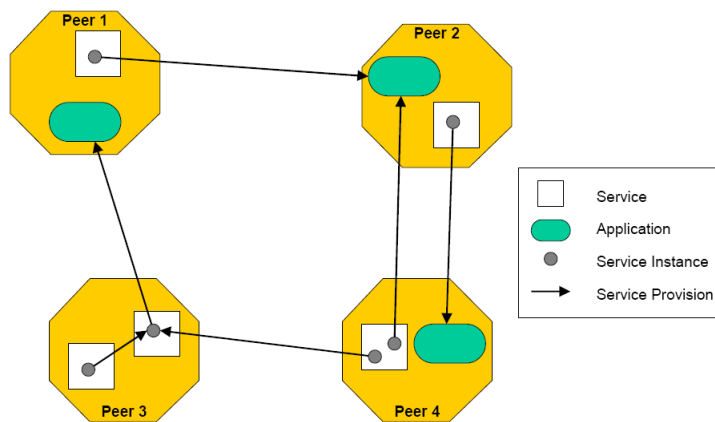
- *Service Support*
 - *Support of completely different services (resource-based services and higher-level services)*
- *Market-based Management*
 - *Creation of a market place for trading different services*
 - *Managed by market mechanisms providing appropriate incentives*
- *Decentralization*

Non-functional Requirements:

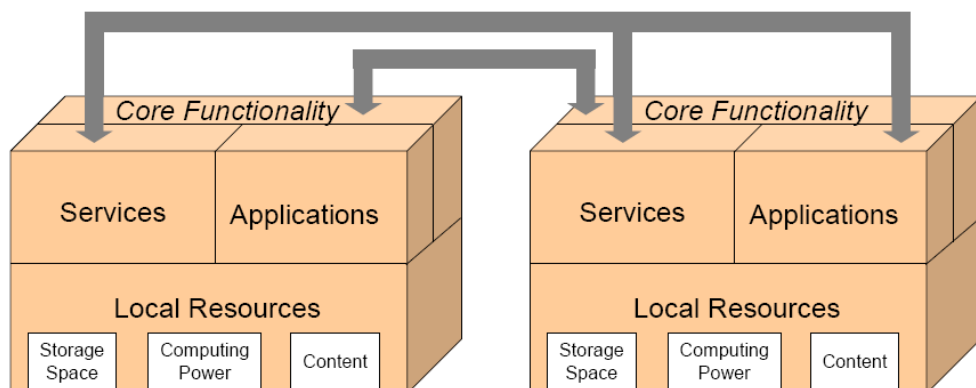
- *Efficiency*
 - *Economically efficient allocation and use of services*
 - *Maximization of overall social benefit of all participants*
 - *Efficient use of technical resources*
- *Scalability*
- *Reliability (Continuous availability, ability to perform correctly and securely)*
- *Accountability*

Architecture

- Consist of three models. Each of them describes one aspect of the architecture:
 - Market model
 - Peers play the role of services providers and services customers (can do both at the same time)
 - Market is not bound to a certain location, but is completely distributed
 - Offers low barriers of entry
 - Additional mechanisms are required:
 - Dynamic Pricing
 - Reputation
 - Binding service contracts
 - Contract enforcement
 - Use model



- Peer model
 - Describes the different parts of a peer
 - Every peer brings its own resources into the system
 - Applications on a peer use the peer's local resources
 - Services on a peer use the peer's local resources and provide remote access to these resources
 - Every peer requires core functionality
 - To manage its services
 - To manage the access to services
 - Mechanisms for resource management, service discovery, reputation management and accounting, charging and pricing



Case Studies

Peer-to-peer middleware:

- The implementations of core functionality form a middleware

PeerMart:

- Basic Concept
 - Each service is traded in a double auction
 - Each auction is mapped onto a cluster of broker peers
- Pricing
 - Maps a service onto a value (price)
 - Communicates price to other peers
- Accounting & Charging
 - Aggregate the service value
 - Maps the aggregated value onto a monetary charge
- Broker set:
 - Each peer has public/private key pair, certified offline and bound to node ID
 - Service have a unique service ID
 - 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 delays between peers
 - Brokers forward candidates for a match, matches determined by majority decisions
- Trading scenarios:
 - APs
 - Access points (APs) provide mobile terminals access to the internet
 - IPSs
 - Internet service providers (ISPs) are connected to one or more Internet Exchange Points (IXPs)

Open questions related to bandwidth trading:

- Would such an approach be technically and economically feasible?
- Would such an approach be accepted by stakeholders?
- What if providers exploit their powerful position?
- Who is to blame if there is a problem?

Decentralized Accounting

Incentive Patterns

- Trust based 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 patterns
 - Barter trade pattern
 - Barter trade is defined as the exchange of services. Hence, the consumer remunerates the provider simultaneously providing a service in return.
 - 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.

Comparison of incentive patterns:

	Collective/community pattern	Barter trade pattern	Bond based pattern
Examples	EigenTrust, Nice	BitTorrent, eMule	Kazaa
Scalability	+/-	+	+/-
Persistence	+	-	+
Anonymity	-	+	+/-
No Trust	-	+	+/-
Flexibility	+/-	-	+
Acceptance	+	+	-

Barter-Trade versus Bond-based Patterns:

- Barter-Trade
 - Only immediate and bilateral trading
 - Exchange goods must be of equal value
 - Low transaction costs
- Bond-based
 - Flexibility: deferred and multilateral trading
 - Forgery
 - Double-spending
 - 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
- Overuse has to be compensated
 - By providing own services
 - By paying money to a peer who has reached T (threshold) in opposite way

Economic Mechanisms Required:

- Pricing
- Accounting
- Charging

P2P Accounting

Goals of accounting:

- Ensure accountability
- Provide incentives → Enforce fair sharing of resources by peers
- Enable commercial use → Server as a basis for additional charging and payment mechanisms

Accounting Design Space:

- *Local accounting*
 - *Accounting information is stored locally*
 - *Suitable for trusted environments*
- *Token-based accounting*
 - *Accounting information is stored in tokens which can be aggregated locally*
 - *Tokens are issued by a trusted token issuer, e.g. a bank or a quorum of peers*
- *Remote accounting*
 - *Accounting information is stored remotely → Accounting information can be distributed and replicated over several peers*
 - *Provider/Consumer send balance updates to remote peers*
 - *Increases reliability and availability of accounting data*
 - *A higher credibility or trustworthiness can be achieved*
 - *Three different types of remote accounting:*
 - *Central accounting*
 - *Central server stores accounting information*
 - *Hybrid accounting*
 - *Several (super)peers which are highly trusted store accounting information*
 - *Distributed accounting*
 - *Accounting information is stored in a fully distributed manner*
 - *Two types of distributed accounting*
 - *Distributed non-redundant accounting*
 - *Every account is held by only one peer → Easy to maintain but vulnerable against failures and malicious peers*
 - *Distributed redundant accounting*
 - *Accounts are replicated over several peers -> Difficult to keep consistent and higher accounting costs but higher availability and more reliable against maliciousness*
 - *Example: KARMA*
 - *Problems:*
 - *Collusion among peers → update only provider account*
 - *Either peer may cheat*
 - *Provider doesn't deliver service*
 - *Consumer sends incorrect balance update*
 - *Consistency of accounts*
 - *An mediator can be used to collect all updates first and analyze/verify them*

Token-based accounting

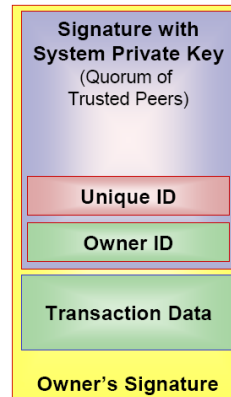
Building blocks:

- *Token Structure*
 - *Transaction receipts*
 - *General exchange medium*
 - *Prevent forgery and robbery*
- *Transaction protocols*
 - *Peers exchange services*
 - *Peers cannot respend tokens*
- *Token aggregation protocol*
 - *Issue new tokens*

- Exchange of foreign tokens against new own tokens
- Protection against double spending
 - Detect and trace double spending

Token Structure:

- Signature with system private key
 - Sign for validation
 - Prevent forgery
- Unique ID
 - Prevent double spending
- Owner ID
 - Trace to double spender
 - No anonymity required
- Transaction data
 - Receipt
- Owner's signature
 - Avoid falsification



Token Exchange:

Token werden hergestellt, indem sie von jedem Mitglied einer SuperPeer Gruppe signiert werden. Danach können die Token als Zahlungsmittel verwendet werden. Da nur mit eigenen Token bezahlt werden kann, müssen gesammelte Token beim SuperPeer gegen neue, eigene Token getauscht werden.

Es gibt zwei Mögliche Transferprotokolle: Der Secure Approach deckt zweifaches Ausgeben desselben Token schon vor der Transaktion auf. Dafür muss aber eine Liste der gültigen Token eines Peer auf einem anderen Peer abgespeichert werden. Dieser muss dann benutzte Token von seiner Liste streichen. Beim Scalable Approach kann zweifaches Ausgeben zwar nicht verhindert werden, es wird aber bemerkt. SuperPeers veröffentlichen dazu Listen mit Token, welche sie gegen neue Token eingetauscht haben. Jetzt kann entdeckt werden, wenn derselbe Token zum zweiten mal eingetauscht wird.

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.

PeerMint Account Model:

- Account: Repository to keep accounting data
- Session account: keeps accounting data within a session

- Mapped onto m session mediation peers
 - Collect balance updates from provider and consumer
 - Verify if peers agree and update account accordingly
- Peer account: aggregates a peer's balance over several sessions
 - Mapped onto p peer account holders
 - Collect balance updates from session mediation peers
 - Account is update according to majority decision

PeerMint Tariff Model:

- Define how service usage is accounted for
- Specify when and by how much accounts are updated
- Enables a variety of settlement schemes

Potential attacks:

- Corrupt leaf set: The probability that the attacker acquires the majority of a leafset is nevertheless very small (even 10% of all the nodes in the network are compromised)

PeerMint Conclusions:

- Designed mechanisms are efficient and scale well
- High reliability even in the presence of malicious peers
- Generic service support through use of tariffs
- Future Work
 - Consider and prevent overlay splitting
 - Use reputation to punish malicious peers
 - Study other forms of malicious attacks (e.g. DDoS)

Hybrid Peer-to-Peer Systems

Hybrid Peer-to-Peer Systems

Definition:

- Initially, Systems combining P2P and C/S characteristics were called hybrid
- Compared to pure P2P systems, there is another dynamic hierarchical layer

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
 - Via bootstrap-server (host list from a web server)
 - Via peer cache (from previous sessions)
 - Via well-known host
 - 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)*
- *Signaling connections*
 - *Stable as long as neighbors do not change*
- *Content transfer connections*
 - *Temporary*
 - *Out of band transmission*

Hybrid P2P Example: Gnutella 0.6

- *Higher signaling efficiency than pure P2P*
- *Same reliability (no single point of failure)*

Network Organization:

- *Upon connection to the network via a super peer, each node is a leaf node*
- *It announces its shared content to the super peer it connected to*
- *Super peer updates its routing table*
- *An election mechanism decides which node becomes a super peer or a leaf node (depending on capabilities, network connection, uptime, etc.)*

Routing:

- *Content request*
 - *Leaf node sends request to super peer*
 - *Super peer looks up in its routing tables whether content is offered by one of its leaf nodes. In this case the request is forwarded to this node*
 - *Additionally the super peer increases the hop counter and forwards this request to the super peer it is connected to*
 - *To enable backward routing, the peers has to store the GUID of the message connected to the information from which peer it received the request in the previous hop*
- *Content response*
 - *If a leaf node receives a request, it double-checks whether it shares the file*
 - *In case of success, the leaf node sends a content reply back to the requesting peer, by sending it back to that node it received the message from*
- *Content exchange*
 - *Directly between the leaf nodes via HTTP connections*

Hybrid Architectures

JXTA:

- *Peergroup*
 - *Ad hoc set of peers with common interests*
 - *Common policies among participants*
 - *Hierarchical relationship among different peergroups*
- *Rendezvous peers*
 - *Maintain an index of advertisements via the Shared Resource Distributed Index (SRDI) service*

- *Edge peers*
 - *Use SRDI to push advertisements to their rendezvous and discover remote advertisements*

Brocade:

- *A hybrid approach aiming at improving the end-to-end routing distance and reducing the network bandwidth usage*
- *Exploits knowledge of the underlying network characteristics*
- *Involves two overlay networks → the default overlay network and a secondary network among super peers where super peers are placed in critical locations*
- *Routing:*
 - *Edge peers forward messages to nearby super peers*
 - *Super peers operate as “shortcuts” and tunnel efficiently message to their final destinations*
 - *After arriving at the closest super node to the destination, the message is delivered again via the default overlay network to the final destination*

Shark:

- *Queries are routed initially in the structured part satisfying the involved keywords*
- *The targeted unstructured network is located*
- *Broadcast mechanisms are used in the set of related peers*

Omicron:

- *Two-tier architecture → peers are grouped in clusters*
 - *Redundancy and fault-tolerance*
 - *Locality aware*
 - *Finer load balancing*
 - *Handling hot spots*
- *In a cluster there are different roles*
 - *Maintainer → maintains the structure (topology)*
 - *Indexer → indexing advertised items*
 - *Cacher → caching popular items*
 - *Router → routing queries*

Hybrid Routing

OceanStore (P2P Storage System):

- *An additional probabilistic routing mechanism based on attenuated bloom filters is used to take advantage of the non-uniform query distribution*

Bloom Filters:

- *Filter für kontinuierlichen Datenstrom mit dem schnell gemessen werden kann, ob die Daten schon einmal aufgetreten sind*
- *Uses hash functions*
- *Provides tradeoff between required time and space*
- *Errors may happen: positive replies may be false, negative replies are always correct*

Hybrid PIER:

- *Distributed query engine built on top of CAN*
- *Exploits the advantage of looking for popular items → ultrapeers are used to locate popular items*
- *Query routing:*

- Flooding is used for popular items
- DHT lookup is used for rare items
- Efficient routing is provided when item popularity is non-uniform

System Comparison and Discussion

Hybrid P2P System	Main Objectives	Key Mechanisms
Omicron	Efficient and stable large scale, heterogeneous, dynamic P2P systems	Clustering, Dynamic role assignment
SHARK	Scalable range queries	Hybrid structural Overlay
JXTA	Low cost overlay management	Role separation
Brocade	Efficient mapping of overlay to underlay network	Location-aware subnetwork
OceanStore	Low cost search for popular queries	Bloom Filters based caching
Hybrid PIER	Low cost search for popular items	Hybrid structural overlay

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 physical network completely because of hub structure
- Asymmetric load (super peers have to bear significantly higher load)

Advantages of Hybrid P2P:

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

Further P2P systems based on hybrid P2P:

- Emule/Edonkey/Kazaa/FastTrack

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

PlanetLab Central (PLC):

- Trusted intermediary between node owners and node users
- Control of operating systems on node
- Access permissions and resource allocation to services

- Operated by PlanetLab administrators at Princeton University

PlanetLab and P2P:

- PlanetLab is a hybrid P2P system
 - Nodes are relatively autonomous
 - Local control through admin slice
- PlanetLab enables deployment of P2P applications at planetary scale and evaluation of P2P applications in a realistic setting

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

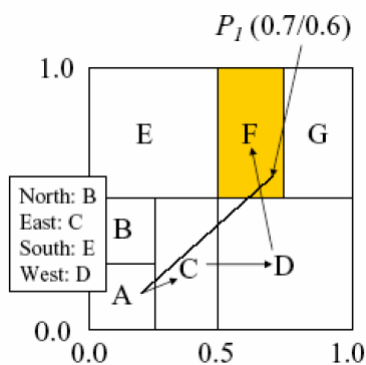
- Counting Bloom Filter
 - Each entry in the filter need not be a single bit but rather a small counter
 - Delete operation possible → decrementing counter
- Spectral Bloom Filter
 - Extend the data structure to support estimates of frequencies
- Compressed Bloom Filter
 - When the filter is intended 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

- A scalable Content Addressable Network
- Uses d -dimensional Cartesian coordinate space (d -torus) → each node owns a zone on the torus
- Storing Key/Value pair ($K1, V1$)
 - $K1$ mapped to Point $P1$ using uniform hash function
 - ($K1, V1$) stored at the node N that owns the zone containing $P1$
- Each node stores IP address and coordinate zone of adjoining zones → node's routing table

CAN 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 JOIN message to node whose zone contains P (node N)
- 3) Node N splits its zone and allocates half to new node, transfer of (key,value) 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 Markets

- *P2P applications often lack revenue generation*

Revenue model of P2P:

- *Currently only indirect (e.g. ads)*
- *Viable direct business models are sought*

Markets:

- *Markets comprise out of 4 main phases*
 - *Knowledge: participants gather knowledge about products and potential trading partners*
 - *Intention: participants formulate their offers to trade*
 - *Agreement: result – the allocation – is determined based on the offers*
 - *Settlement: the deal resulting from the agreement phase are executed*

Market Requirements:

- *Extensible with respect to mechanisms and functions*
- *Extensible with respect to market participants*
- *Robust and secure*
- *No monopolistic market provider, instead distributed infrastructure*

Basics

P2P Application Style:

- *Packaged solutions*
 - *Es ist klar definiert wer teilnehmen kann, wer registriert ist und wer mit wem kommunizieren darf*
- *Set of common definitions*
 - *Grundsätzlich kann jeder daran teilnehmen*

Business Models versus Revenue Models

- *Business Model: Totality of processes and arrangements that define a company's approach to commercial markets in order to sell services and/or goods and generate profit*
- *Revenue Model: Includes all arrangements that permit the participants in business interactions to charge fees which are covered by one or several other participants in order to cover costs and add a margin to create profit*
 - *Indirect Revenue Model: product is free of charge → gain received from third party*

- *Direct Revenue Model: receipts come directly from customer*

Requirements of a Revenue Model:

- *Differentiated Charging*
- *Allocation Effectiveness*

Sample Revenue Models

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

Discussion

- Revenue models for P2P application style are not different from those for traditional application style

Increase Revenue:

- *Instant Messaging*
 - Bundling with interactive agents
 - Providing location based services
 - Multiple service levels (Skype / Skype Out)
- *Digital Content Sharing*
 - Try to own communities
 - Bundling digital content with other goods (e.g. concert tickets)
- *Grid Computing:*
 - Bundling is no solution and micropayment may not be feasible
 - Barter-like structures → provide information goods as reimbursement
- *Collaboration*
 - Bundling similar to IM
 - Multiple service levels

Mobile and Collaborative P2P Systems

Background Technology

Ad-hoc Networks:

- Self configuring
- Infrastructure free
- Wireless
- Unpredictable terminal mobility
- Limited radio transmission range
- MANET = Mobile Ad-hoc Networks

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

- Expensive connections

Paradigms and Application Areas

Challenges:

- Adapt to physical network (no zigzag routes) with cross layer communication
- Decrease signaling load significantly
- Avoid long transfer distances
- Cope with route breaks

Application Support and Areas:

- Provide context-aware routing in MANETs
- Context-aware: Wissen über die anderen Nodes und Vektoren – wohin sie sich bewegen, wie schnell, wie hoch die Reichweite ihres Wireless ist, usw.
- Facilitate location-based services without any additional support of central entities
- Search in your local proximity for available services and content

Problems of Mobile P2P Systems

- High background traffic / background noise
- P2P overlay non-optimal routing
 - Random establishment of connections in the virtual overlay resulting in zigzag routes
 - Virtual overlay network does not match to the physical network → too much control overhead for connection maintenance
- Mobility and DHT
 - DHT is hardly adjustable to physical layer
 - Zigzag routes can't be avoided in physical multi-hop network
 - → not efficient in mobile scenarios

Employing a MANET just as another TCP network on which a P2P network can be established is not reasonable

→ Mobile Peer-to-Peer Protocol (MPP) is necessary

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

Mobility Support

Similarities between unstructured P2P and MANET:

Similarity	P2P	MANET
Basic routing principle	Virtual broadcast, flooding	Physical broadcast, flooding
Network topology	Flat and frequently changing (frequent logon and logoffs)	Flat and frequently changing (terminal mobility)
Network logon	Via a portal (bootstrap node)	Via a portal (specific well known broadcast radio channel)
Network management	QoS and A4C difficult to implement (no central management)	QoS and A4C difficult to implement (no central management)

Mobile P2P Protocol (MPP):

- *MPP is based on Dynamic Source Routing (DSR)*
- *DSR: bei jeder Weitergabe des Pakets wird die eigene Adresse angehängt – dynamic: wird so lange gemacht, bis das Netzwerk stabil ist*
- *Employs cross layer communication to match virtual network on physical DSR network → avoids zigzag routes completely*
- *Employs HTTP for content transmission → possibility to download from multiple sources and to continue an interrupted download from other sources*

Advantages of the joint approach of MPP:

- *MANET controls the network organization*
 - *Changes in the topology are automatically taken into account by the P2P network*
 - *Network layer is responsible for routing*
 - *Application controls data exchange*
- *Integration of both networks avoids redundant information requests*
- *Inter-layer communication optimizes the network performance, as the overlay can be optimally adjusted to the physical network*
- *The application layer protocol MPP simplifies the implementation of new services*

Overall MANET Advantages:

- *Service can be located without any central element*
- *Fast and cheap*
- *Location awareness by using MANET*
- *Other services can easily be added, because of the P2P basis*

Collaboration

- *Two types of collaboration*
 - *Active collaboration → physical user interaction*
 - *Passive collaboration → multi-hop information dissemination*

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*

Summary

Mobile P2P Challenges:

- *Reduce P2P lookup search traffic overhead to overcome low transmission data rates of mobile devices*
- *Address high churn rates due to frequent join and leave of nodes*
- *Consider limited resources of mobile devices*
- *Minimize number of hops*