Design and Analysis of Peer-to-Peer Systems

Aaron Harwood

aharwood@csse.unimelb.edu.au
http://p2p.csse.unimelb.edu.au

NICTA Victoria Laboratories
Peer-to-Peer Networks and Applications Research Group
Department of Computer Science and Software Engineering

December 3, 2007
We will use a design focused approach.

1. Introduction
2. Design principles
3. Churn and Connectivity
4. Advanced Queries
5. Software models
Challenges for Distributed Computing
Centered around Internet applications

e-Science ⇒ life sciences

Challenges for Distributed Computing
Centered around Internet applications

- e-Science ⇒ life sciences

- Content Distribution
  - Large data files.
  - VoIP, video streaming, IPTV.
  - Bandwidth allocation.
Challenges for Distributed Computing
Centered around Internet applications

- **e-Science ⇒ life sciences**

- **Content Distribution**
  - Large data files.
  - VoIP, video streaming, IPTV.
  - Bandwidth allocation.

- **Sensor Networks**
  - Connectivity – inter/intra, robust algs.
  - Local and global optimizations.
### Challenges for Distributed Computing

Centered around Internet applications

<table>
<thead>
<tr>
<th>e-Science ⇒ life sciences</th>
<th>Online Gaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Capacity – difficult to predict and costly to provide.</td>
<td></td>
</tr>
<tr>
<td>- Latency – directly relates to quality of service.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Content Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Large data files.</td>
</tr>
<tr>
<td>- VoIP, video streaming, IPTV.</td>
</tr>
<tr>
<td>- Bandwidth allocation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sensor Networks</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Connectivity – inter/intra, robust algs.</td>
</tr>
<tr>
<td>- Local and global optimizations.</td>
</tr>
</tbody>
</table>
## Challenges for Distributed Computing

Centered around Internet applications

### e-Science ⇒ life sciences

### Content Distribution
- Large data files.
- VoIP, video streaming, IPTV.
- Bandwidth allocation.

### Online Gaming
- Capacity – difficult to predict and costly to provide.
- Latency – directly relates to quality of service.

### Security
- Collaborative intrusion detection.
- Open systems – spam.

### Sensor Networks
- Connectivity – inter/intra, robust alg.
- Local and global optimizations.
Challenges for Distributed Computing
Centered around Internet applications

<table>
<thead>
<tr>
<th>e-Science ⇒ life sciences</th>
<th>Online Gaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Capacity – difficult to predict and costly to provide.</td>
<td></td>
</tr>
<tr>
<td>- Latency – directly relates to quality of service.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Content Distribution</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Large data files.</td>
<td></td>
</tr>
<tr>
<td>- VoIP, video streaming, IPTV.</td>
<td></td>
</tr>
<tr>
<td>- Bandwidth allocation.</td>
<td></td>
</tr>
<tr>
<td>- Collaborative intrusion detection.</td>
<td></td>
</tr>
<tr>
<td>- Open systems – spam.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sensor Networks</th>
<th>Pervasive Computing</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Connectivity – inter/intra, robust algs.</td>
<td></td>
</tr>
<tr>
<td>- Local and global optimizations.</td>
<td></td>
</tr>
<tr>
<td>- Mixed/augmented reality.</td>
<td></td>
</tr>
<tr>
<td>- Mobility, ad-hoc networking.</td>
<td></td>
</tr>
</tbody>
</table>
Importance of Peer-to-Peer
Emergent solution that overcomes many current challenges

The Peer-to-Peer model of distributed computing applies to a wide variety of application domains.
Some lessons learnt

- No one approach to building peer-to-peer systems is the best way. Different approaches suit different applications and sometimes a mix of structured and unstructured design is applicable.
Some lessons learnt

- No one approach to building peer-to-peer systems is the best way. Different approaches suit different applications and sometimes a mix of structured and unstructured design is applicable.

- Practically all useful protocols have changed significantly since their first publication. Implementations vary a lot.
Some lessons learnt

- No one approach to building peer-to-peer systems is the best way. Different approaches suit different applications and sometimes a mix of structured and unstructured design is applicable.

- Practically all useful protocols have changed significantly since their first publication. Implementations vary a lot.

- Software reuse, of existing protocols and services, is particularly useful for prototype applications and demonstrators. Using a modular or component based approach allows transparent replacement of lower layer functionality; it can be particularly hard to get it right though.
Characteristics of Peer-to-Peer applications
Decentralized, large scale, autonomous

Decentralized
Particularly applicable when the primary utility is provided by users to other users. E.g.:

- Resource sharing – data files and processing.
- Buying and selling – e.g. Amazon and eBay.
- Gaming – e.g. World of Warcraft.
- Collaboration – VoIP & video conferencing, social activities.
Characteristics of Peer-to-Peer applications
Decentralized, large scale, autonomous

**Decentralized**
Particularly applicable when the primary utility is provided by users to other users. E.g.:
- Resource sharing – data files and processing.
- Buying and selling – e.g. Amazon and eBay.
- Gaming – e.g. World of Warcraft.
- Collaboration – VoIP & video conferencing, social activities.

**Large scale**
- Available capacity of users’ computers become part of the whole – low cost for deployment.
- Mutual benefit – Metcalf’s Law.
- New functionality – anonymity.
Characteristics of Peer-to-Peer applications
Decentralized, large scale, autonomous

**Decentralized**
Particularly applicable when the primary utility is provided by users to other users. E.g.:
- Resource sharing – data files and processing.
- Buying and selling – e.g. *Amazon* and *eBay*.
- Gaming – e.g. *World of Warcraft*.
- Collaboration – VoIP & video conferencing, social activities.

**Large scale**
- Available capacity of users’ computers become part of the whole – low cost for deployment.
- Mutual benefit – Metcalf’s Law.
- New functionality – anonymity.

**Autonomous**
- Dynamic – users come and go obliviously to each other.
- No single control – independent administrative domains.
Outline

1. Introduction
2. Design principles
3. Churn and Connectivity
4. Advanced Queries
5. Software models
Common Protocol Operations
Join, leave, lookup/query, put, get, delete.

Network oriented operations
Common Protocol Operations

Join, leave, lookup/query, put, get, delete.

Network oriented operations

**Join** – The peer becomes *online* making itself available to other online peers.
Common Protocol Operations
Join, leave, lookup/query, put, get, delete.

Network oriented operations

**Join** – The peer becomes *online* making itself available to other online peers.

**Leave** – The peer changes to *offline*, and is not contactable while offline.
## Network oriented operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Join</strong></td>
<td>The peer becomes <em>online</em> making itself available to other online peers.</td>
</tr>
<tr>
<td><strong>Leave</strong></td>
<td>The peer changes to <em>offline</em>, and is not contactable while offline.</td>
</tr>
<tr>
<td><strong>Lookup/Query</strong></td>
<td>A message used to find peers (lookup) or data (query).</td>
</tr>
</tbody>
</table>
**Common Protocol Operations**

Join, leave, lookup/query, put, get, delete.

<table>
<thead>
<tr>
<th>Network oriented operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Join</strong> – The peer becomes <em>online</em> making itself available to other online peers.</td>
</tr>
<tr>
<td><strong>Leave</strong> – The peer changes to <em>offline</em>, and is not contactable while offline.</td>
</tr>
<tr>
<td><strong>Lookup/Query</strong> – A message used to find peers (lookup) or data (query).</td>
</tr>
</tbody>
</table>

| Data oriented operations |
Common Protocol Operations
Join, leave, lookup/query, put, get, delete.

Network oriented operations

Join – The peer becomes online making itself available to other online peers.

Leave – The peer changes to offline, and is not contactable while offline.

Lookup/Query – A message used to find peers (lookup) or data (query).

Data oriented operations

Put – Some data is made available.
## Common Protocol Operations

Join, leave, lookup/query, put, get, delete.

<table>
<thead>
<tr>
<th>Network oriented operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Join</strong> – The peer becomes <em>online</em> making itself available</td>
</tr>
<tr>
<td>to other online peers.</td>
</tr>
<tr>
<td><strong>Leave</strong> – The peer changes to <em>offline</em>, and is not</td>
</tr>
<tr>
<td>contactable while offline.</td>
</tr>
<tr>
<td><strong>Lookup/Query</strong> – A message used to find peers (lookup) or</td>
</tr>
<tr>
<td>data (query).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data oriented operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Put</strong> – Some data is made available.</td>
</tr>
<tr>
<td><strong>Get</strong> – Retrieve data.</td>
</tr>
</tbody>
</table>
Common Protocol Operations
Join, leave, lookup/query, put, get, delete.

Network oriented operations

Join – The peer becomes online making itself available to other online peers.

Leave – The peer changes to offline, and is not contactable while offline.

Lookup/Query – A message used to find peers (lookup) or data (query).

Data oriented operations

Put – Some data is made available.

Get – Retrieve data.

Delete – Previously inserted data is made unavailable.
The Neighbor Table
Fundamentally just a list of known IP addresses.

**Table:** Example Neighbor Table Entries

<table>
<thead>
<tr>
<th>IP address</th>
<th>Last ping time (ms)</th>
<th>Time since last contact (s)</th>
<th>Gateway IP</th>
<th>Protocol specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>210.28.10.5</td>
<td>35</td>
<td>600</td>
<td>-</td>
<td>...</td>
</tr>
</tbody>
</table>
The Neighbor Table
Fundamentally just a list of known IP addresses.

**Table: Example Neighbor Table Entries**

<table>
<thead>
<tr>
<th>IP address</th>
<th>Last ping time (ms)</th>
<th>Time since last contact (s)</th>
<th>Gateway IP</th>
<th>Protocol specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>210.28.10.5</td>
<td>35</td>
<td>600</td>
<td>-</td>
<td>⋮</td>
</tr>
<tr>
<td>192.168.1.5</td>
<td>250</td>
<td>75320</td>
<td>210.28.10.37</td>
<td>⋮</td>
</tr>
</tbody>
</table>
The Neighbor Table
Fundamentally just a list of known IP addresses.

Table: Example Neighbor Table Entries

<table>
<thead>
<tr>
<th>IP address</th>
<th>Last ping time (ms)</th>
<th>Time since last contact (s)</th>
<th>Gateway IP</th>
<th>Protocol specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>210.28.10.5</td>
<td>35</td>
<td>600</td>
<td>-</td>
<td>⋮</td>
</tr>
<tr>
<td>192.168.1.5</td>
<td>250</td>
<td>75320</td>
<td>210.28.10.37</td>
<td>⋮</td>
</tr>
<tr>
<td>210.28.10.37</td>
<td>75</td>
<td>10</td>
<td>-</td>
<td>⋮</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

An example Neighbor Table. Host 210.28.10.37 is a gateway for host 192.168.1.5 on a private network (the combination of gateway/private IP is unique). The gateway is itself another peer in the network.
Bootstrapping a P2P Network
Methods for finding an existing peer to connect to.

Reliable peers – Some number of “24/7” peers that are well known. Not needed once the total number of online peers at any one time is large enough for caching to work.
Bootstrapping a P2P Network

Methods for finding an existing peer to connect to.

**Reliable peers** – Some number of “24/7” peers that are well known. Not needed once the total number of online peers at any one time is large enough for caching to work.

**Cached list** – A list of previously contacted peer IP addresses.
Bootstrapping a P2P Network
Methods for finding an existing peer to connect to.

Reliable peers – Some number of “24/7” peers that are well known. Not needed once the total number of online peers at any one time is large enough for caching to work.

Cached list – A list of previously contacted peer IP addresses.

IP broadcast – A network level method of searching for peers. Maybe limited to local networks or inefficient random sampling of IP space.
Bootstrapping a P2P Network
Methods for finding an existing peer to connect to.

**Reliable peers** – Some number of “24/7” peers that are well known. Not needed once the total number of online peers at any one time is large enough for caching to work.

**Cached list** – A list of previously contacted peer IP addresses.

**IP broadcast** – A network level method of searching for peers. Maybe limited to local networks or inefficient random sampling of IP space.

**Host cache service** – An independent, well known, “24/7” service. Useful for testing P2P applications. Sometimes called a name server.
Bootstrapping a P2P Network
Methods for finding an existing peer to connect to.

Reliable peers – Some number of “24/7” peers that are well known. Not needed once the total number of online peers at any one time is large enough for caching to work.

Cached list – A list of previously contacted peer IP addresses.

IP broadcast – A network level method of searching for peers. Maybe limited to local networks or inefficient random sampling of IP space.

Host cache service – An independent, well known, “24/7” service. Useful for testing P2P applications. Sometimes called a name server.

Manual – User supplied existing IP address. Should always be a possible option.
Simple Unstructured Example

A Gnutella like approach: ping, pong, query, hit, get, push.

**Ping and pong** – used to advertise how much data is available at the peers.
Simple Unstructured Example
A *Gnutella* like approach: ping, pong, query, hit, get, push.

**Ping and pong** – used to advertise how much data is available at the peers.

**Query and hit** – used to search for data and respond to searches.
Simple Unstructured Example

A Gnutella like approach: ping, pong, query, hit, get, push.

Ping and pong – used to advertise how much data is available at the peers.

Query and hit – used to search for data and respond to searches.

Get and push – used to obtain data.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

Node 2 sends a connect message to node 1.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

Node 1 sends an ok message back to node 2.
Simple Unstructured Example
A *Gnutella* like approach: ping, pong, query, hit, get, push.

Node 1 and 2 are now connected.
Simple Unstructured Example
A *Gnutella* like approach: ping, pong, query, hit, get, push.

Similarly node 3 connects to node 1.
Simple Unstructured Example
A Gnutella like approach: ping, pong, query, hit, get, push.

Similarly node 4 connects to node 3.
Simple Unstructured Example
A *Gnutella* like approach: ping, pong, query, hit, get, push.

Similarly node 5 connects to node 3.
Simple Unstructured Example
A *Gnutella* like approach: ping, pong, query, hit, get, push.

Similarly node 6 connects to node 4.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

Similarly node 7 connects to node 4.
Simple Unstructured Example
A *Gnutella* like approach: ping, pong, query, hit, get, push.

A ping floods through the network.
Simple Unstructured Example

A Gnutella like approach: ping, pong, query, hit, get, push.

A ping floods through the network.
Simple Unstructured Example
A *Gnutella* like approach: ping, pong, query, hit, get, push.

A ping floods through the network.
Simple Unstructured Example
A *Gnutella* like approach: ping, pong, query, hit, get, push.

A ping floods through the network.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

A ping floods through the network.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

A ping floods through the network.
Simple Unstructured Example

A Gnutella like approach: ping, pong, query, hit, get, push.

A ping floods through the network.
Simple Unstructured Example

A Gnutella like approach: ping, pong, query, hit, get, push.

Pongs follow the same path back.
Simple Unstructured Example
A *Gnutella* like approach: ping, pong, query, hit, get, push.

Pongs follow the same path back.
Simple Unstructured Example
A *Gnutella* like approach: ping, pong, query, hit, get, push.

Pongs follow the same path back.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

Node 2 can now make more connections.
Simple Unstructured Example
A *Gnutella* like approach: ping, pong, query, hit, get, push.

Consider nodes 4, 6, 7 behind a firewall.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

Query from node 2 floods to all nodes.
Simple Unstructured Example
A *Gnutella* like approach: ping, pong, query, hit, get, push.

Recall: 4 had already connected to 3.

Query from node 2 floods to all nodes.
Simple Unstructured Example
A Gnutella like approach: ping, pong, query, hit, get, push.

Hits come back from some nodes.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

Hits come back from some nodes.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

Hits come back from some nodes.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

A get to a public IP address works.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

A get to a public IP address works.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

A get to a private IP address fails.
Simple Unstructured Example
A *Gnutella* like approach: ping, pong, query, hit, get, push.

Push is used to request the file.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

Push is used to request the file.
Simple Unstructured Example

A *Gnutella* like approach: ping, pong, query, hit, get, push.

Push is used to request the file.
Superpeers
Do indexing and searching on behalf of others.
Superpeers
Do indexing and searching on behalf of others.

- Superpeers
- Do indexing and searching on behalf of others.

Design and Analysis of Peer-to-Peer Systems
Aaron Harwood

Introduction
Design principles
Churn and Connectivity
Advanced Queries
Software models

Diagram:
- Superpeers
- Do indexing and searching on behalf of others.
- Can have more than one parent
- Leaf
- Superpeer
- Cluster
Superpeers
Do indexing and searching on behalf of others.

join and send index of local resources to superpeer
Superpeers
Do indexing and searching on behalf of others.
Superpeers
Do indexing and searching on behalf of others.

remove index from superpeer before leaving
Superpeers
Do indexing and searching on behalf of others.
Superpeers
Do indexing and searching on behalf of others.
Superpeers
Do indexing and searching on behalf of others.
Superpeers
Do indexing and searching on behalf of others.
Superpeers
Do indexing and searching on behalf of others.
Superpeers
Do indexing and searching on behalf of others.
Superpeers
Do indexing and searching on behalf of others.
Superpeers
Do indexing and searching on behalf of others.
Superpeers
Do indexing and searching on behalf of others.
Superpeers
Do indexing and searching on behalf of others.
From Wang, Yun, et al., 2007:

- Peers with degree $x$ can occur with probability $f(x) = C x^\alpha$ where $C > 0$, $\alpha < -1$; called the power law distribution.

- For leaf to superpeer degree, $C = 0.8140$ and $\alpha = 2.5815$ is a good fit. A leaf may connect to several superpeers.

- For superpeer to leaf degree, follows a Gaussian distribution with $\sigma = 0.933$ and $x_0 = 29.4109$.

- For superpeer to superpeer degree, a double peak Gaussian fits, $\sigma_1 = 1.9019$, $x_1 = 30.4794$, $\sigma_2 = 1.3413$, $x_2 = 25.0275$. Different implementations of the software cause different peaks.
Designing a superpeer network
Superpeer redundancy, cluster sizes.

Designing a superpeer network
Superpeer redundancy, cluster sizes.
Designing a superpeer network
Superpeer redundancy, cluster sizes.

From Yang and Garcia-Molina, 2003:

- In a $k$-redundant cluster, leaves send identical information to $k$ superpeers. In this case, the failure of a superpeer does not disconnect the leaves from the network and the superpeers can share the network load, clients send queries in round-robin fashion and superpeers in the cluster coordinate incoming queries from other clusters.
Designing a superpeer network
Superpeer redundancy, cluster sizes.

From Yang and Garcia-Molina, 2003:

- In a $k$-redundant cluster, leaves send identical information to $k$ superpeers. In this case, the failure of a superpeer does not disconnect the leaves from the network and the superpeers can share the network load, clients send queries in round-robin fashion and superpeers in the cluster coordinate incoming queries from other clusters.

- $k$ times as much overhead for leaf joins/leaves. If every cluster is $k$ redundant then $k^2$ increase in open connections. Usually $k = 2$ is reasonable.
From Yang and Garcia-Molina, 2003:

- In a $k$-redundant cluster, leaves send identical information to $k$ superpeers. In this case, the failure of a superpeer does not disconnect the leaves from the network and the superpeers can share the network load, clients send queries in round-robin fashion and superpeers in the cluster coordinate incoming queries from other clusters.

- $k$ times as much overhead for leaf joins/leaves. If every cluster is $k$ redundant then $k^2$ increase in open connections. Usually $k = 2$ is reasonable.

- Increasing cluster size decreases aggregate load, but increases individual load. Superpeer redundancy is good, to an extent, as it greatly reduces individual load.
Designing a superpeer network
Superpeer redundancy, cluster sizes.

If a cluster consists of a fraction $f$ of all nodes in the network, i.e. $f = \frac{\text{cluster size}}{\text{number of users}}$, then the superpeer for that cluster will be submitting roughly $f$ of all queries on behalf of its clients.
Designing a superpeer network
Superpeer redundancy, cluster sizes.

If a cluster consists of a fraction $f$ of all nodes in the network, i.e. $f = \frac{\text{cluster size}}{\text{number users}}$, then the superpeer for that cluster will be submitting roughly $f$ of all queries on behalf of its clients. Files in its own index constitute roughly $f$ of all responses to queries and so $1 - f$ of all results will be sent from other clusters to this superpeer.
If a cluster consists of a fraction $f$ of all nodes in the network, i.e. $f = \frac{\text{cluster size}}{\text{number users}}$, then the superpeer for that cluster will be submitting roughly $f$ of all queries on behalf of its clients. Files in its own index constitute roughly $f$ of all responses to queries and so $1 - f$ of all results will be sent from other clusters to this superpeer. Therefore the total expected incoming results is proportional to $f(1 - f)$ which has a maximum at $\frac{1}{2}$.
If a cluster consists of a fraction $f$ of all nodes in the network, i.e. $f = \frac{\text{cluster size}}{\text{number users}}$, then the superpeer for that cluster will be submitting roughly $f$ of all queries on behalf of its clients. Files in its own index constitute roughly $f$ of all responses to queries and so $1 - f$ of all results will be sent from other clusters to this superpeer. Therefore the total expected incoming results is proportional to $f(1 - f)$ which has a maximum at $\frac{1}{2}$. Designs should keep $f$ away from around a tenth to a half of the total number of peers in the network.
Other aspects
TTL, failures, malicious peers.

**TTL** – Time To Live limits the extent of flooding which reduces traffic, at the expense of search effectiveness. *Targeted query* enables the user to select the starting peer for the query broadcast, in case of small TTL values.
Other aspects
TTL, failures, malicious peers.

**TTL** – Time To Live limits the extent of flooding which reduces traffic, at the expense of search effectiveness. *Targeted query* enables the user to select the starting peer for the query broadcast, in case of small TTL values.

**Peer failures** – Nothing in the protocol is done (nor necessary needs to be) about failing peers. Peer failures can lead to network partitioning, depends on the actual connections at the time. Fairly robust against random peer failure due to power law distribution.
Other aspects
TTL, failures, malicious peers.

**TTL** – Time To Live limits the extent of flooding which reduces traffic, at the expense of search effectiveness. *Targeted query* enables the user to select the starting peer for the query broadcast, in case of small TTL values.

**Peer failures** – Nothing in the protocol is done (nor necessary needs to be) about failing peers. Peer failures can lead to network partitioning, depends on the actual connections at the time. Fairly robust against random peer failure due to power law distribution.

**Malicious peers** – IP addresses, associations, full path names are all visible. Induced denial of service via fake push and fake pong messages is possible. Fake content.
Reinforcement learning in unstructured networks
Not really structured yet – *Freenet*-like approach.

One approach:

1. For a given query value, initially a peer sends the query to a selection of neighbors with equal probability.
Reinforcement learning in unstructured networks
Not really structured yet – *Freenet*-like approach.

One approach:

1. For a given query value, initially a peer sends the query to a selection of neighbors with equal probability.
2. Hits that come back from the neighbors are ranked in terms of their quality and quantity.
Reinforcement learning in unstructured networks
Not really structured yet – *Freenet*-like approach.

One approach:

1. For a given query value, initially a peer sends the query to a selection of neighbors with equal probability.
2. Hits that come back from the neighbors are ranked in terms of their quality and quantity.
3. Neighbors that are ranked highly are then more likely to be selected in the future for that particular query value.
One approach:

1. For a given query value, initially a peer sends the query to a selection of neighbors with equal probability.
2. Hits that come back from the neighbors are ranked in terms of their quality and quantity.
3. Neighbors that are ranked highly are then more likely to be selected in the future for that particular query value.
Reinforcement learning in unstructured networks
Not really structured yet – *Freenet*-like approach.

One approach:

1. For a given query value, initially a peer sends the query to a selection of neighbors with equal probability.
2. Hits that come back from the neighbors are ranked in terms of their quality and quantity.
3. Neighbors that are ranked highly are then more likely to be selected in the future for that particular query value.

If the querying peer also adds highly ranked peers to its neighbor list, and removes low ranking peers, then peer communities will form.
Reinforcement learning in unstructured networks
Not really structured yet – *Freenet*-like approach.

One approach:

1. For a given query value, initially a peer sends the query to a selection of neighbors with equal probability.
2. Hits that come back from the neighbors are ranked in terms of their quality and quantity.
3. Neighbors that are ranked highly are then more likely to be selected in the future for that particular query value.

If the querying peer also adds highly ranked peers to its neighbor list, and removes low ranking peers, then peer communities will form.

This approach also provides a level of reputation for peers, to avoid malicious peers. (But hits have to be verified first.)
Overlay network routing
Peers are assigned a P2P network-wide unique ID (e.g. 128 bit number)
Overlay network routing
Peers are assigned a P2P network-wide unique ID (e.g. 128 bit number)
Overlay network routing
Peers are assigned a P2P network-wide unique ID (e.g. 128 bit number)
Overlay networks establish a level of networking above the Transport Layer, and so it is sometimes called *Application Layer networking.*
Overlay networks establish a level of networking above the Transport Layer, and so it is sometimes called *Application Layer networking*. Overlay networks can also use source routing, with appropriate discovery mechanisms.
Overlay networks establish a level of networking above the Transport Layer, and so it is sometimes called *Application Layer networking*. Overlay networks can also use source routing, with appropriate discovery mechanisms. The superpeer design can be used to distinguish between peers that relay or route for other peers and peers that exist only at the edges of the overlay network.
Overlay networks establish a level of networking above the Transport Layer, and so it is sometimes called *Application Layer networking*. Overlay networks can also use source routing, with appropriate discovery mechanisms. The superpeer design can be used to distinguish between peers that relay or route for other peers and peers that exist only at the edges of the overlay network. Functionality and services can be built on top of the overlay network in much the same way as it is built on top of existing network layer technology.
Overlay networks establish a level of networking above the Transport Layer, and so it is sometimes called *Application Layer networking*. Overlay networks can also use source routing, with appropriate discovery mechanisms. The superpeer design can be used to distinguish between peers that relay or route for other peers and peers that exist only at the edges of the overlay network. Functionality and services can be built on top of the overlay network in much the same way as it is built on top of existing network layer technology. Need methods for publishing services, finding services, forming logical groups, securely communicating, etc.
Do we have structure yet?
Not really.

Resource discovery still requires broadcast or a well known centralized service.
Do we have structure yet?
Not really.

Resource discovery still requires broadcast or a well known centralized service.
E.g., for a given unique ID, $x$, of some resource or peer, how long does it take to find where is $x$ in the P2P network?
Do we have structure yet?
Not really.

Resource discovery still requires broadcast or a well known centralized service. E.g., for a given unique ID, $x$, of some resource or peer, how long does it take to find where is $x$ in the P2P network? It depends on how much information each peer records locally, how accurate or up-to-date that information is, and how that information relates between neighboring peers. In this worst case, $O(n)$ messages are required.
Do we have structure yet?
Not really.

Resource discovery still requires broadcast or a well known centralized service.
E.g., for a given unique ID, \( x \), of some resource or peer, how long does it take to find where is \( x \) in the P2P network?
It depends on how much information each peer records locally, how accurate or up-to-date that information is, and how that information relates between neighboring peers. In this worst case, \( O(n) \) messages are required.
In our discussion about structured networks we will mostly illustrate peer lookups rather than data put/get (i.e. DHTs). The concept is the same though.
Asymptotic Minimum Cost Routing
Graph theoretic approach.

*Moores Bound* gives the maximum number of peers, \( n(k, d) \), that can be contained in a network of diameter \( k \) and degree \( d \):

\[
n(k, d) \leq \frac{d(d - 1)^k - 2}{d - 2} = O(d^k).
\]
Asymptotic Minimum Cost Routing
Graph theoretic approach.

*Moores Bound* gives the maximum number of peers, \( n(k, d) \), that can be contained in a network of diameter \( k \) and degree \( d \):

\[
n(k, d) \leq \frac{d(d - 1)^k - 2}{d - 2} = O(d^k).
\]

A network is said to be *asymptotically cost optimal* if its cost \( c = d k \) is a constant factor from the minimum cost. It follows that with \( d > 2 \) and \( k \) a function of \( d \):

\[
c > d \log_{d-1} \left(n - \frac{2}{d}(n - 1)\right) = \Omega(d \log_d n)
\]
Moores Bound gives the maximum number of peers, \( n(k, d) \), that can be contained in a network of diameter \( k \) and degree \( d \):

\[
n(k, d) \leq \frac{d(d - 1)^k - 2}{d - 2} = O(d^k).
\]

A network is said to be asymptotically cost optimal if its cost \( c = d k \) is a constant factor from the minimum cost. It follows that with \( d > 2 \) and \( k \) a function of \( d \):

\[
c > d \log_{d-1} \left( n - \frac{2}{d}(n - 1) \right) = \Omega(d \log_d n)
\]

If every peer in a P2P network has at most \( d \) neighbors in its neighbor table then it will take \( O(\log_d n) \) messages to locate a given peer.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.

Design and Analysis of Peer-to-Peer Systems
Aaron Harwood

Introduction
Design principles
Churn and Connectivity
Advanced Queries
Software models
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Inefficient Structured Example
A distributed doubly linked list.
Greedy Routing Protocol
Not always the best protocol to use.

When a peer receives a lookup for \( x \), the peer selects, from the union of the peers’ IDs in its neighbor table and its own ID, the peer with ID that is closest (using modulo arithmetic) to \( x \), called the next hop peer.
Greedy Routing Protocol
Not always the best protocol to use.

When a peer receives a lookup for \( x \), the peer selects, from the union of the peers’ IDs in its neighbor table and its own ID, the peer with ID that is closest (using modulo arithmetic) to \( x \), called the next hop peer.

Tie-breakers can be selected with a priority for the lowest ID.
Greedy Routing Protocol
Not always the best protocol to use.

When a peer receives a lookup for $x$, the peer selects, from the union of the peers’ IDs in its neighbor table and its own ID, the peer with ID that is closest (using modulo arithmetic) to $x$, called the next hop peer.

Tie-breakers can be selected with a priority for the lowest ID. The message is then forwarded to the next hop peer.
When a peer receives a lookup for \( x \), the peer selects, from the union of the peers’ IDs in its neighbor table and its own ID, the peer with ID that is closest (using modulo arithmetic) to \( x \), called the next hop peer.

Tie-breakers can be selected with a priority for the lowest ID. The message is then forwarded to the next hop peer. However, if that peer is itself the closest, then the peer responds to the joining peer.
Greedy Routing Protocol
Not always the best protocol to use.

When a peer receives a lookup for $x$, the peer selects, from the union of the peers’ IDs in its neighbor table and its own ID, the peer with ID that is closest (using modulo arithmetic) to $x$, called the next hop peer.

Tie-breakers can be selected with a priority for the lowest ID. The message is then forwarded to the next hop peer. However, if that peer is itself the closest, then the peer responds to the joining peer.

E.g., on the previous slide, if peer 4 joins via 14 then the lookup message would route via peer 1 since 1 is closer to 5 than 10.
Unique peer IDs can be effectively generated by selecting at random from a sufficiently large identifier space (e.g. 128 bit space). A clash can always be resolved by the joining peer selecting again (it's very unlikely to clash again!).
Unique Peer IDs

Unique peer IDs can be effectively generated by selecting at random from a sufficiently large identifier space (e.g. 128 bit space). A clash can always be resolved by the joining peer selecting again (its very unlikely to clash again!). A hash function on the peer’s IP address can also be used to generate a unique peer ID, assuming the peer’s IP address is a public one. Otherwise the hash function can be applied to the private/gateway IP address combination.
Cost of the Cycle
From a graph theoretic view.

The cost is $O(n)$ – lookups for an ID take $O(n)$ steps, but each peer has a neighbor table with $O(1)$ entries.
Cost of the Cycle
From a graph theoretic view.

The cost is $O(n)$ – lookups for an ID take $O(n)$ steps, but each peer has a neighbor table with $O(1)$ entries. So the cycle has a cost no better than an unstructured network.
Cost of the Cycle
From a graph theoretic view.

The cost is $O(n)$ – lookups for an ID take $O(n)$ steps, but each peer has a neighbor table with $O(1)$ entries. So the cycle has a cost no better than an unstructured network. (Actually, the cycle is a so called *Moore Graph* since for degree 2 and diameter $k = \left\lfloor \frac{n}{2} \right\rfloor$ it contains the most number of nodes possible.)
Achieving a Lower Cost
Using extra edges - a bi-directional *Chord*-like example.

Assuming a network of many peers, only the relevant peers are shown in this diagram.
Achieving a Lower Cost
Using extra edges - a bi-directional *Chord*-like example.

Every peer has a similarly defined set of connections.
Achieving a Lower Cost
Using extra edges - a bi-directional *Chord*-like example.

Every peer has a similarly defined set of connections.
Achieving a Lower Cost
Using extra edges - a bi-directional *Chord*-like example.

Every peer has a similarly defined set of connections.
Achieving a Lower Cost
Using extra edges - a bi-directional *Chord*-like example.

The actual Chord protocol does not point in both directions because of correctness proofs that require it to point in one direction only.
Achieving a Lower Cost
Using extra edges - a bi-directional *Chord*-like example.

The actual Chord protocol does not point in both directions because of correctness proofs that require it to point in one direction only.
How to find the correct neighbors to include in the neighbor table?
Achieving a Lower Cost
Using extra edges - a bi-directional Chord-like example.

The actual Chord protocol does not point in both directions because of correctness proofs that require it to point in one direction only.

How to find the correct neighbors to include in the neighbor table? Use the lookup protocol to lookup the “ideal peer”; the protocol returns the nearest peer. When there are few peers, some lookups will return the same peer and so the neighbor table can be compacted in this case.
Achieving a Lower Cost
Using extra edges - a bi-directional *Chord*-like example.

The cycle interval, between two neighbors in the table, that includes the destination is halving in length each lookup step. Therefore the number of lookups is $O(\log n)$ (with high probability) with the last hop on the longest path being a hop in the cycle.
Achieving a Lower Cost
Using extra edges - a bi-directional *Chord*-like example.

The cycle interval, between two neighbors in the table, that includes the destination is halving in length each lookup step. Therefore the number of lookups is $O(\log n)$ (with high probability) with the last hop on the longest path being a hop in the cycle.

Also, only $O(\log n)$ neighbors required in the table, so the cost is $O(\log^2 n)$; which is good but still not asymptotic minimum cost of $O\left(\frac{\log n}{\log \log n}\right)$. 
Asymptotic Minimum Cost P2P Network
Koorde – based on the deBruijn Graph.

In base 2, with \(2^b\) identifiers numbered 0, 1, \ldots, \(2^b - 1\), a node \(m\) connects to \(2m \mod 2^b\) and \((2m + 1) \mod 2^b\).

\[
m.\text{Lookup}(k, kshift, i)
\]

if \(k \in (m, \text{successor}]\) then
  return (successor)
else if \(i \in (m, \text{successor}]\) then
  return \{ d.\text{Lookup}(k, kshift \ll 1, i \circ \text{topBit}(kshift)) \}
else return (successor.\text{Lookup}(k, kshift, i))
Asymptotic Minimum Cost P2P Network

*Koorde* – based on the deBruijn Graph.

In the P2P network, node *m* maintains its immediate (cycle) successor and the predecessor to 2*m*, called *d*, in its neighbor table.

### m.Lookup(*k*, *kshift*, *i*)

if *k* ∈ (*m*, *successor*) then
    return (*successor*)

else if *i* ∈ (*m*, *successor*) then
    return { *d*.Lookup(*k*, *kshift* << 1, *i* ◦ topBit(*kshift*)) }

else return (*successor*.Lookup(*k*, *kshift*, *i*))
Asymptotic Minimum Cost P2P Network

Koorde – based on the deBruijn Graph.

Lookup for $k = 3$ starting at 5, would route via imaginary deBruijn nodes 10, 4, 9 to 3. The path is followed in the P2P network by following the real nodes that support those imaginary nodes.

$m$.Lookup($k$, $kshift$, $i$)

if $k \in (m, \text{successor}]$ then return (successor)
else if $i \in (m, \text{successor}]$ then return 
\{ $d$.Lookup($k$, $kshift \ll 1$, $i \circ \text{topBit}(kshift)$) \}
else return (successor$.Lookup(k$, $kshift$, $i$))
Asymptotic Minimum Cost P2P Network

*Koorde* – based on the deBruijn Graph.

Lookup for \( k = 3 \) starting at 5, would route via imaginary de-Bruijn nodes 10, 4, 9 to 3. The path is followed in the P2P network by following the real nodes that support those imaginary nodes.

\[
m.\text{Lookup}(k, kshift, i)\\
\text{if } k \in (m, \text{successor}] \text{ then return (successor)}\\
\text{else if } i \in (m, \text{successor}] \text{ then return } \{ d.\text{Lookup}(k, kshift << 1, i \circ \text{topBit}(kshift)) \}\\
\text{else return (successor.\text{Lookup}(k, kshift, i))}
\]
The path terminates back at 5, since it is the successor of 3 in this example. It terminates early because both 4 and 3 are succeeded by 5.
Asymptotic Minimum Cost P2P Network
*Koorde* – based on the deBruijn Graph.

From Kaashoek and Karger:

- The algorithm described will contact $O(b)$ nodes. This can be reduced to $O(\log n)$ with high probability by selecting an appropriate starting imaginary node from the nodes between $m$ and its successor.
Asymptotic Minimum Cost P2P Network

*Koorde* — based on the deBruijn Graph.

From Kaashoek and Karger:

- The algorithm described will contact $O(b)$ nodes. This can be reduced to $O(\log n)$ with high probability by selecting an appropriate starting imaginary node from the nodes between $m$ and its successor.
- Koorde operates much like Chord in all other respects.
From Kaashoek and Karger:

- The algorithm described will contact $O(b)$ nodes. This can be reduced to $O(\log n)$ with high probability by selecting an appropriate starting imaginary node from the nodes between $m$ and its successor.
- Koorde operates much like Chord in all other respects.
- Degree $\delta$ deBruijn graphs can be used. In this case node $m$ connects to $\delta m, \delta m + 1, \ldots, \delta m + (\delta - 1)$ all modulo $\delta b$. In this case the diameter is $O(\log_\delta n)$. 
Connection Caching
Traffic based approach that quickly reaches cost optimality.

The rate of \((v, v')\)-requests determines whether a new connection is formed or not. Similarly the rate of requests across a connection determines if the connection is pruned or not. This is basically connection caching. A new connection enters the cache and is evicted if it is not used.
Connection Caching
Traffic based approach that quickly reaches cost optimality.
Connection Caching
Traffic based approach that quickly reaches cost optimality.
Connection Caching
Traffic based approach that quickly reaches cost optimality.
Connection Caching
Traffic based approach that quickly reaches cost optimality.
Connection Caching
Traffic based approach that quickly reaches cost optimality.
Connection Caching
Traffic based approach that quickly reaches cost optimality.
Connection Caching
Traffic based approach that quickly reaches cost optimality.
Connection Caching
Traffic based approach that quickly reaches cost optimality.
Connection Caching
Traffic based approach that quickly reaches cost optimality.
Connection Caching
Traffic based approach that quickly reaches cost optimality.
Connection Caching
Traffic based approach that quickly reaches cost optimality.
Connection Caching
Traffic based approach that quickly reaches cost optimality.
The steady state out degree for various $\tau_{in}$ and 1000 peers.
Mean hop count and average out degree on the same plot versus number of peers.
Connection Caching
Simulation - optimality

Mean hop count and average out degree on the same plot versus $\tau_{in}$.

This line converges to the asymptotic minimum.
Why limit Neighbor Table sizes to $O(\log n)$ or $O(1)$? Most PCs on the Internet can cache at least 1000’s of IP addresses with insignificant overhead. However in some cases this may not be possible.
Why limit Neighbor Table sizes to $O(\log n)$ or $O(1)$? Most PCs on the Internet can cache at least 1000’s of IP addresses with insignificant overhead. However in some cases this may not be possible.

An alternative approach is to limit bandwidth usage (and to a lessor extent processing capacity). Allow the peer to maintain a Neighbor Table as large as it can, within specified bandwidth limitations.
Why limit Neighbor Table sizes to $O(\log n)$ or $O(1)$? Most PCs on the Internet can cache at least 1000’s of IP addresses with insignificant overhead. However in some cases this may not be possible.

An alternative approach is to limit bandwidth usage (and to a lesser extent processing capacity). Allow the peer to maintain a Neighbor Table as large as it can, within specified bandwidth limitations.

Still need to prune entries to ensure they are not stale – stale entries cause timeouts during communication.
Accordion Protocol

Salient design points

- Bandwidth budget can be specified by the user in terms of average desired rate of traffic and the maximum burst size. Accordion uses available bandwidth with both proactive and reactive techniques to improve its neighbor table.
Accordion Protocol

Salient design points

- Bandwidth budget can be specified by the user in terms of average desired rate of traffic and the maximum burst size. Accordion uses available bandwidth with both proactive and reactive techniques to improve its neighbor table.
- If $n_1$ forwards a lookup for key $k$ to $n_2$, $n_2$ returns a set of neighbors in the ID range between $n_2$ and $k$. This allows $n_1$ to learn about neighbors that, by definition, it does not yet have in its table.
Accordion Protocol
Salient design points

- Bandwidth budget can be specified by the user in terms of average desired rate of traffic and the maximum burst size. Accordion uses available bandwidth with both proactive and reactive techniques to improve its neighbor table.
- If $n_1$ forwards a lookup for key $k$ to $n_2$, $n_2$ returns a set of neighbors in the ID range between $n_2$ and $k$. This allows $n_1$ to learn about neighbors that, by definition, it does not yet have in its table.
- Explorative lookups from a node $a$ are sent to neighbors selected by whether the gap in identifier space to the next neighbor is relatively too large. An explorative lookup asks the neighbor to send back (a small constant number) $m$ entries in that range. Entries are chosen using Vivaldi network coordinates so as to try to reduce expected delay to $a$. 
Accordion Protocol
Salient design points

- Bandwidth budget can be specified by the user in terms of average desired rate of traffic and the maximum burst size. Accordion uses available bandwidth with both proactive and reactive techniques to improve its neighbor table.
- If $n_1$ forwards a lookup for key $k$ to $n_2$, $n_2$ returns a set of neighbors in the ID range between $n_2$ and $k$. This allows $n_1$ to learn about neighbors that, by definition, it does not yet have in its table.
- Explorative lookups from a node $a$ are sent to neighbors selected by whether the gap in identifier space to the next neighbor is relatively too large. An explorative lookup asks the neighbor to send back (a small constant number) $m$ entries in that range. Entries are chosen using Vivaldi network coordinates so as to try an reduce expected delay to $a$.
- When learning about a new neighbor, nodes also learn about their bandwidth budgets so that they can avoid pushing the neighbor over its bandwidth budget.
Accordion Protocol
Salient design points

- Bandwidth budget can be specified by the user in terms of average desired rate of traffic and the maximum burst size. Accordion uses available bandwidth with both proactive and reactive techniques to improve its neighbor table.
- If $n_1$ forwards a lookup for key $k$ to $n_2$, $n_2$ returns a set of neighbors in the ID range between $n_2$ and $k$. This allows $n_1$ to learn about neighbors that, by definition, it does not yet have in its table.
- Explorative lookups from a node $a$ are sent to neighbors selected by whether the gap in identifier space to the next neighbor is relatively too large. An explorative lookup asks the neighbor to send back (a small constant number) $m$ entries in that range. Entries are chosen using *Vivaldi network coordinates* so as to try an reduce expected delay to $a$.
- When learning about a new neighbor, nodes also learn about their bandwidth budgets so that they can avoid pushing the neighbor over its bandwidth budget.
- Neighbors are pruned based on an estimation of whether they are still alive or not.
Fault Tolerance
Transient failures (in terms of a cycle for simplicity).

For an unstructured design, faulty peers or connections do not usually frustrate the protocol. Data may become unavailable if peers are disconnected or not reached by a query, but the data becomes available again when the fault is repaired.
Fault Tolerance
Transient failures (in terms of a cycle for simplicity).

For an unstructured design, faulty peers or connections do not usually frustrate the protocol. Data may become unavailable if peers are disconnected or not reached by a query, but the data becomes available again when the fault is repaired. Unlike unstructured designs, structured designs can be problematic in the presence of faults.
Fault Tolerance
Transient failures (in terms of a cycle for simplicity).

For an unstructured design, faulty peers or connections do not usually frustrate the protocol. Data may become unavailable if peers are disconnected or not reached by a query, but the data becomes available again when the fault is repaired. Unlike unstructured designs, structured designs can be problematic in the presence of faults. Structured designs benefit from proactive maintenance to ensure that faults are found and repaired before they become a problem. This prevents logical errors and also reduces the number of timeouts that messages experience when being routed.
The Greedy Routing Protocol will stop at peer 7. Depending on the design, peer 7 may erroneously conclude that it is the closest peer to 11, since its only live neighbor is 6.
Fault Tolerance
Transient failures.

In this simple example, two faulty nodes will partition the network.
Fault Tolerance
Transient failures.

Including $\alpha$-successors can be used to address transient failures.

- $\alpha = 1$ has no redundancy.
- $\alpha = 2$ has twice as many connections.
- $\alpha = O(\log n)$ is sufficient for surviving with high probability up to half the nodes failing simultaneously (see work concerning the Chord protocol).

Extra connections also help the Greedy Routing Protocol to route quicker, but still $O(n)$ routing for constant $\alpha$. 
Both nodes 7 and 14 might (erroneously) conclude that node 10 has failed. A lookup from 5, for 11, would therefore terminate at 14.
However a lookup from 3 will terminate at 10. Since the lookup operation is used to insert a new peer into the network, converging to a consistently defined structure can be problematic.
Fault Tolerance
What guarantees will the protocol provide?

- Will message delivery always reach the destination, if connected by at least one path?
Fault Tolerance
What guarantees will the protocol provide?

- Will message delivery always reach the destination, if connected by at least one path?
- Will messages be delivered in order?
Fault Tolerance
What guarantees will the protocol provide?

- Will message delivery always reach the destination, if connected by at least one path?
- Will messages be delivered in order?
- What about data? Will it be preserved?
Fault Tolerance
What guarantees will the protocol provide?

- Will message delivery always reach the destination, if connected by at least one path?
- Will messages be delivered in order?
- What about data? Will it be preserved?
Fault Tolerance
What guarantees will the protocol provide?

- Will message delivery always reach the destination, if connected by at least one path?
- Will messages be delivered in order?
- What about data? Will it be preserved?

Designing for fault tolerance is not only possible but is a inevitable requirement for large scale systems like P2P networks. *However*, the implementation, testing and debugging requires a lot of effort.
In unstructured networks, peers do not really store the data “on the network”, but rather just make local data available. References to the data are sent as hits and possibly as the results of other search operations.
In unstructured networks, peers do not really store the data “on the network”, but rather just make local data available. References to the data are sent as hits and possibly as the results of other search operations.

In structured networks, and especially distributed hash tables, data is typically stored by value, i.e. on the network.
In unstructured networks, peers do not really store the data “on the network”, but rather just make local data available. References to the data are sent as hits and possibly as the results of other search operations.

In structured networks, and especially distributed hash tables, data is typically stored by value, i.e. on the network. If the same data is available on multiple peers then it can be quicker to access and also more reliable. However it can be harder to update, i.e. we have a data consistency problem.
Data operations
Store by value or reference?

In unstructured networks, peers do not really store the data “on the network”, but rather just make local data available. References to the data are sent as hits and possibly as the results of other search operations.

In structured networks, and especially distributed hash tables, data is typically stored by value, i.e. on the network. If the same data is available on multiple peers then it can be quicker to access and also more reliable. However it can be harder to update, i.e. we have a data consistency problem. If a peer wants to remove data when not online then store by reference may be more appropriate. On the other hand if the data should be available even if the peer is offline then store by value is required.
If objects should be persistent in the network then we can apply replication and/or migrated. Also, as peers join/leave, a structured network will need to ensure that objects can still be efficiently located. Unstructured networks do not have this overhead.
Replication, Migration, Object Caching

If objects should be persistent in the network then we can apply replication and/or migrated. Also, as peers join/leave, a structured network will need to ensure that objects can still be efficiently located. Unstructured networks do not have this overhead.

For a structured network, usually there is one peer that is the “owner” and other peers that cache copies. Keeping track of the owner and ensuring any kinds of consistent updates can be a lot of effort (both logically and implementation wise) to ensure it works in all circumstances.
If objects should be persistent in the network then we can apply replication and/or migrated. Also, as peers join/leave, a structured network will need to ensure that objects can still be efficiently located. Unstructured networks do not have this overhead.

For a structured network, usually there is one peer that is the “owner” and other peers that cache copies. Keeping track of the owner and ensuring any kinds of consistent updates can be a lot of effort (both logically and implementation wise) to ensure it works in all circumstances.

Replication is typically easier than migration; migration is a kind of replication that deletes the original after it has been copied to a new peer. Migration can be problematic in the face of churn, i.e. when the target peer leaves during the migration, or other peers join, etc.
Replication, Migration, Object Caching

If objects should be persistent in the network then we can apply replication and/or migrated. Also, as peers join/leave, a structured network will need to ensure that objects can still be efficiently located. Unstructured networks do not have this overhead.

For a structured network, usually there is one peer that is the “owner” and other peers that cache copies. Keeping track of the owner and ensuring any kinds of consistent updates can be a lot of effort (both logically and implementation wise) to ensure it works in all circumstances.

Replication is typically easier than migration; migration is a kind of replication that deletes the original after it has been copied to a new peer. Migration can be problematic in the face of churn, i.e. when the target peer leaves during the migration, or other peers join, etc.

Objects can be cached on the return journey along the lookup paths, to make subsequent lookups much faster.
Replication strategies include:

- Replicate on $k$-successors and $k$-predecessors in the identifier space.

- Rename replica objects by appending a “salt” value and store the replica object as a first class object.

The first method is the most popular. The second method may be easier to implement but it does not guarantee that the replicas are on different peers.
Other aspects

Parallel Lookups – Can reduce lookup time by avoiding timeouts on some paths. Can also be used to increase robustness and frustrate malicious peers.
Other aspects

Parallel Lookups – Can reduce lookup time by avoiding timeouts on some paths. Can also be used to increase robustness and frustrate malicious peers.

Proximity Preservation – If peers get a choice of which neighbors will go into the table, then all other things being equal they should choose the peer with the smallest ping time. This leads to a lower total latency for a given logical path.
Other aspects

Parallel Lookups – Can reduce lookup time by avoiding timeouts on some paths. Can also be used to increase robustness and frustrate malicious peers.

Proximity Preservation – If peers get a choice of which neighbors will go into the table, then all other things being equal they should choose the peer with the smallest ping time. This leads to a lower total latency for a given logical path.

Load Balancing with Virtual Nodes – Peers can put themselves into the network more than once, with different identifiers. This can help to even out load imbalances. (It is also a form of attack by malicious peers!)
Perhaps the single biggest security problem for P2P networks is that the IP addresses of the peers is usually quite visible to other peers. We can use anonymizing networks to obscure peer IP addresses but these protocols add long latencies.
Security, Reputation/trust and Malicious Peers

Perhaps the single biggest security problem for P2P networks is that the IP addresses of the peers is usually quite visible to other peers. We can use anonymizing networks to obscure peer IP addresses but these protocols add long latencies. Trusted Computing may actually help to build very secure P2P networks in a very easy way!
Perhaps the single biggest security problem for P2P networks is that the IP addresses of the peers is usually quite visible to other peers. We can use anonymizing networks to obscure peer IP addresses but these protocols add long latencies. Trusted Computing may actually help to build very secure P2P networks in a very easy way!

There are many reputation and trust models proposed in the literature. Reinforcement learning described earlier is itself a fundamental approach to eliminate (or ostracize) “unperforming” or malicious peers. Some game theoretic approaches have been proposed for bandwidth allocation among a competing set of peers.
Outline

1. Introduction
2. Design principles
3. Churn and Connectivity
4. Advanced Queries
5. Software models
Peers join and leave the P2P network autonomously. This process gives rise to *churn*.
Peers join and leave the P2P network autonomously. This process gives rise to *churn*.

From a global perspective, churn increases as the number of peers increases, since the churn of each individual peer is additive.
Peers join and leave the P2P network autonomously. This process gives rise to *churn*.

From a global perspective, churn increases as the number of peers increases, since the churn of each individual peer is additive.

Measuring and modeling churn is difficult because of the widely heterogeneous behavior of peers.
Churn and Connectivity

- Peers join and leave the P2P network autonomously. This process gives rise to *churn*.
- From a global perspective, churn increases as the number of peers increases, since the churn of each individual peer is additive.
- Measuring and modeling churn is difficult because of the widely heterogeneous behavior of peers.
- Understanding how a P2P protocol is affected by churn is vital for the success of the protocol.
A widely used method for modeling churn is to consider that each peer spends time a mean time $1/\lambda$ offline and a mean time $1/\mu$ online, where the times are exponentially distributed with parameters $\lambda$ and $\mu$ resp.
A widely used method for modeling churn is to consider that each peer spends time a mean time $1/\lambda$ offline and a mean time $1/\mu$ online, where the times are exponentially distributed with parameters $\lambda$ and $\mu$ resp. Therefore, if the peer is offline at time 0, the probability that it is online by time $t$,

$$P[t] = 1 - e^{-\lambda t}.$$ 

Similarly for changing from online to offline.
Random Model of Churn
The Engset model.

Peers are either online or offline; each peer independently arrives and departs with rates $\lambda$ and $\mu$ respectively.

$$\frac{\lambda}{\lambda+\mu} N$$  $$\frac{\mu}{\lambda+\mu} N$$
A Path in the Network
And the effectiveness of bootstrapping from a cached table.

Consider a path along $k$ peers. Assume that if any of the peers leave the network then path is lost.
Consider a path along $k$ peers. Assume that if any of the peers leave the network then path is lost.
The probability that none of the peers along a path of length $k$ have left the network, i.e. that the path still exists, at time $t$ given that it existed at time 0 is:

$$\prod_{i=1}^{k} e^{-\mu t} = e^{-k \mu t}.$$
Consider a path along $k$ peers. Assume that if any of the peers leave the network then path is lost.

The probability that none of the peers along a path of length $k$ have left the network, i.e. that the path still exists, at time $t$ given that it existed at time 0 is:

$$\prod_{i=1}^{k} e^{-\mu t} = e^{-k\mu t}.$$ 

Of course, a peer may leave the network and then come back to rejoin the path. In this case we might ask for the fraction of time that the $k$ peers are all in the network (i.e. the probability at some time instant that they are all online):

$$\left(\frac{\lambda}{\mu + \lambda}\right)^k.$$
A Path in the Network
And the effectiveness of bootstrapping from a cached table.

For a cached neighbor table of $k$ entries then the probability that at least one of them is online is similarly:

$$1 - \left( \frac{\mu}{\mu + \lambda} \right)^k.$$
Other distributions

Yao, Leonard, et al. consider the *shifted Pareto* distribution:

\[ P[t] = 1 - (1 + t/\beta)^{-\alpha}, \quad t > 0, \alpha > 1 \]

with mean \( \beta/(\alpha - 1) \).
Other distributions

Yao, Leonard, et al. consider the *shifted Pareto* distribution:

\[ P[t] = 1 - (1 + t/\beta)^{-\alpha}, \quad t > 0, \alpha > 1 \]

with mean \( \beta/(\alpha - 1) \).

Some recent work has shown that if peers are separated into “short lived class” and “long lived class” then the classes each behave much closer to a random distribution.
Half-life
Connecting churn and connectivity

Consider the *half-life*, $\tau$, intuitively as meaning that after time $t + \tau$, at most half of the system can be extrapolated from its state at time $t$. Nodes arrive at a rate $\lambda$ (here $\lambda$ is an aggregate rate) and each node independently departs at rate $\mu$. 
Consider the *half-life*, $\tau$, intuitively as meaning that after time $t + \tau$, at most half of the system can be extrapolated from its state at time $t$. Nodes arrive at a rate $\lambda$ (here $\lambda$ is an aggregate rate) and each node independently departs at rate $\mu$. If there are $N$ nodes in the system then the doubling time is $N/\lambda$. 
Consider the *half-life*, $\tau$, intuitively as meaning that after time $t + \tau$, at most half of the system can be extrapolated from its state at time $t$. Nodes arrive at a rate $\lambda$ (here $\lambda$ is an aggregate rate) and each node independently departs at rate $\mu$. If there are $N$ nodes in the system then the doubling time is $N/\lambda$.

The probability $p$ that a node fails in time $\tau$ is $1 - e^{-\mu \tau}$. Setting $\tau = (1/\mu) \ln 2$ makes $p = 0.5$, so the halving time is $(1/\mu) \ln 2$. 
Consider the *half-life*, $\tau$, intuitively as meaning that after time $t + \tau$, at most half of the system can be extrapolated from its state at time $t$. Nodes arrive at a rate $\lambda$ (here $\lambda$ is an aggregate rate) and each node independently departs at rate $\mu$. If there are $N$ nodes in the system then the doubling time is $N/\lambda$.

The probability $p$ that a node fails in time $\tau$ is $1 - e^{-\mu\tau}$. Setting $\tau = (1/\mu)\ln 2$ makes $p = 0.5$, so the halving time is $(1/\mu)\ln 2$.

The half-life is $\min\{N/\lambda, (1/\mu)\ln 2\}$. 
A node can initially know at most $N$ other nodes. According to half-life all of these $N$ known neighbors will be gone within $O(\log N)$ half-lives with high probability.
A node can initially know at most $N$ other nodes. According to half-life all of these $N$ known neighbors will be gone within $O(\log N)$ half-lives with high probability.

**Theorem [Liben-Nowell et al.]:** There exists a sequence of joins and leaves such that any node that, at any time, has received an average of fewer than $k$ notifications per half-life will be disconnected from the network with probability at least $(1 - \frac{1}{e-1})^k \approx 0.418^k$. 
Half-life
Connecting churn and connectivity

Let \( \{n_1, n_2, \ldots \} \) be the nodes that \( u \) knows about by time \( t \). Let notification of \( n_1 \) at time \( t_1 \) to be the most recent.
Half-life
Connecting churn and connectivity

Let \( \{n_1, n_2, \ldots \} \) be the nodes that \( u \) knows about by time \( t \). Let notification of \( n_1 \) at time \( t_1 \) to be the most recent. Assuming fewer than \( k \) notifications per half-life \( (\mu = \lambda = 1) \) consider the smallest \( t \) such that fewer than \( t k \) nodes have been heard of.
Let \( \{n_1, n_2, \ldots \} \) be the nodes that \( u \) knows about by time \( t \). Let notification of \( n_1 \) at time \( t_1 \) to be the most recent. Assuming fewer than \( k \) notifications per half-life (\( \mu = \lambda = 1 \)) consider the smallest \( t \) such that fewer than \( t \cdot k \) nodes have been heard of.

The probability that node \( n_i \) is still alive is \( e^{t_i - t} \) since the probability that it departs after time \( \tau \) is \( 1 - e^{-\mu \tau} \).
Let \( \{n_1, n_2, \ldots \} \) be the nodes that \( u \) knows about by time \( t \). Let notification of \( n_1 \) at time \( t_1 \) to be the most recent. Assuming fewer than \( k \) notifications per half-life (\( \mu = \lambda = 1 \)) consider the smallest \( t \) such that fewer than \( tk \) nodes have been heard of.

The probability that node \( n_i \) is still alive is \( e^{t_i - t} \) since the probability that it departs after time \( \tau \) is \( 1 - e^{-\mu \tau} \).

The probability that all nodes have failed, disconnecting \( u \) from the network, is \( P = \prod_i \left( 1 - e^{t_i - t} \right) \).
Let \( \{n_1, n_2, \ldots \} \) be the nodes that \( u \) knows about by time \( t \). Let notification of \( n_1 \) at time \( t_1 \) to be the most recent. Assuming fewer than \( k \) notifications per half-life (\( \mu = \lambda = 1 \)) consider the smallest \( t \) such that fewer than \( t k \) nodes have been heard of.

The probability that node \( n_i \) is still alive is \( e^{t_i - t} \) since the probability that it departs after time \( \tau \) is \( 1 - e^{-\mu \tau} \).

The probability that all nodes have failed, disconnecting \( u \) from the network, is \( P = \prod_i \left(1 - e^{t_i - t} \right) \).

In the time interval \( (t - \delta, t) \) fewer than \( \delta k \) notifications occurred and so no notifications occurred after time \( t - 1/k \) or \( t_1 \leq t - 1/k \). This generalizes to \( t_i \leq t - i/k \).
Half-life
Connecting churn and connectivity

Now:

\[ P = \prod_{i}(1 - e^{t_{i} - t}) \geq \prod_{i}(1 - e^{-i/k}) \]

\[ \geq \left( \prod_{j \geq 1}(1 - e^{-j}) \right)^{k}. \]

The last inequality results from lower bounding each \( e^{-i/k} \) by \( e^{-\lceil i/k \rceil} \).
Half-life
Connecting churn and connectivity

Now:

\[ P = \prod_{i} (1 - e^{t_i - t}) \geq \prod_{i} (1 - e^{-i/k}) \]
\[ \geq \left( \prod_{j \geq 1} (1 - e^{-j}) \right)^k. \]

The last inequality results from lower bounding each \( e^{-i/k} \) by \( e^{-\lceil i/k \rceil} \).

Finally, \( \prod (1 - e^{-j}) \geq 1 - \sum e^{-j} = 1 - \frac{1}{e-1} \), completing the proof.
Requirements for a 2-hop Network
From a random graph point of view.

From random graph theory we know that a random graph $G(n, p)$, which has $n$ nodes and which includes each possible edge independently with probability $p$, almost surely (i.e. with probability approaching 1 as $n$ approaches $\infty$) has a connected component of size $\Theta(n)$ when $p = c/n$ and $c > 1$. 
From random graph theory we know that a random graph \( G(n, p) \), which has \( n \) nodes and which includes each possible edge independently with probability \( p \), almost surely (i.e. with probability approaching 1 as \( n \) approaches \( \infty \)) has a connected component of size \( \Theta(n) \) when \( p = c/n \) and \( c > 1 \). More specifically the size of the largest connected component when \( c > 1 \) is given by \( \epsilon_c n + o(n) \) where \( \epsilon_c \) is the unique solution to \( \epsilon + e^{-c\epsilon} = 1 \).
Consider two peers, $i$ and $j$, where the probability of an edge connecting them is $p$ and otherwise the probability that they are connected by at least one path that is passing through at most one other peer is $1 - (1 - p^2)^{n-2}$. 
Requirements for a 2-hop Network
From a random graph point of view.

Consider two peers, \( i \) and \( j \), where the probability of an edge connecting them is \( p \) and otherwise the probability that they are connected by at least one path that is passing through at most one other peer is \( 1 - (1 - p^2)^{n-2} \).

The probability that \( i \) is connected to \( j \) in at most 2 hops is:

\[
p_{2\text{-hop}} = p + (1 - p) \left( 1 - (1 - p^2)^{n-2} \right)
= 1 - (1 - p)^{n-1} (1 + p)^{n-2}.
\]
Requirements for a 2-hop Network
From a random graph point of view.

Consider two peers, \( i \) and \( j \), where the probability of an edge connecting them is \( p \) and otherwise the probability that they are connected by at least one path that is passing through at most one other peer is \( 1 - (1 - p^2)^{n-2} \). The probability that \( i \) is connected to \( j \) in at most 2 hops is:

\[
p_{2-hop} = p + (1 - p) \left(1 - (1 - p^2)^{n-2}\right)
\]

\[
= 1 - (1 - p)^{n-1} (1 + p)^{n-2}.
\]

Furthermore, by substituting \( p \rightarrow c \frac{\sqrt{n}}{n} \), for constant \( c > 0 \), we have

\[
\lim_{n \rightarrow \infty} p_{2-hop} = 1 - e^{-c^2},
\]

which rapidly approaches 1 as \( c \) increases.
Consider two peers, $i$ and $j$, where the probability of an edge connecting them is $p$ and otherwise the probability that they are connected by at least one path that is passing through at most one other peer is $1 - (1 - p^2)^{n-2}$.

The probability that $i$ is connected to $j$ in at most 2 hops is:

$$p_{2-hop} = p + (1 - p) \left(1 - (1 - p^2)^{n-2}\right)$$

$$= 1 - (1 - p)^{n-1} (1 + p)^{n-2}.$$

Furthermore, by substituting $p \rightarrow c \frac{\sqrt{n}}{n}$, for constant $c > 0$, we have

$$\lim_{n \to \infty} p_{2-hop} = 1 - e^{-c^2},$$

which rapidly approaches 1 as $c$ increases.

This provides an additional perspective for maintaining an $O(\sqrt{n})$ routing table.
Outline

1. Introduction
2. Design principles
3. Churn and Connectivity
4. Advanced Queries
5. Software models
Unstructured networks can have straightforward implementations for some kinds of spatial/range queries, e.g. finding all files of type image and with file size more than 1kB and less than 1MB is a kind of range query.
Unstructured networks can have straightforward implementations for some kinds of spatial/range queries, e.g. finding all files of type image and with file size more than 1kB and less than 1MB is a kind of range query. Structured networks need alternative methods of handling these kinds of queries in order to maintain their competitive lookup and neighbor table complexities.
Advanced Queries
Spatial/range queries.

Unstructured networks can have straightforward implementations for some kinds of spatial/range queries, e.g. finding all files of type image and with file size more than 1kB and less than 1MB is a kind of range query.

Structured networks need alternative methods of handling these kinds of queries in order to maintain their competitive lookup and neighbor table complexities.

Spatial/range queries are important because they form the basis for other kinds of complex operations including publish/subscribe and coordinated resource scheduling.
Spatial Queries
A 3D virtual world example.

The image shows two alternative views of a 3D virtual world. Each view is from a separate peer in the P2P network. The peers collectively maintain the virtual world information.
Spatial Queries
A tree based index is used for efficiency.

Unlike documents which are accessed by name (i.e., where the query is simply a file name), spatial queries usually require complex computations.
Spatial Queries
A tree based index is used for efficiency.

1. Unlike documents which are accessed by name (i.e., where the query is simply a file name), spatial queries usually require complex computations.

2. In 2-d space, a query can be executed efficiently by recursively subdividing the underlying space into four regions, i.e., a quadtree (in 3-d it is called an octree):
Spatial Queries
A tree based index is used for efficiency.

1. Unlike documents which are accessed by name (i.e., where the query is simply a file name), spatial queries usually require complex computations.

2. In 2-d space, a query can be executed efficiently by recursively subdividing the underlying space into four regions, i.e., a quadtree (in 3-d it is called an octree):
   - Objects are indexed into the quadtree by some predefined rule.
Spatial Queries

A tree based index is used for efficiency.

1. Unlike documents which are accessed by name (i.e., where the query is simply a file name), spatial queries usually require complex computations.

2. In 2-d space, a query can be executed efficiently by recursively subdividing the underlying space into four regions, i.e., a quadtree (in 3-d it is called an octree):
   - Objects are indexed into the quadtree by some predefined rule.
   - Effect is to exponentially reduce the number of intersection calculations (e.g., by a factor of 4 for each descent in the tree).
Spatial Queries
A tree based index is used for efficiency.
Assign responsibility for regions of space to peers in the system.
Distributed Spatial Queries

1. Assign responsibility for regions of space to peers in the system.

2. If a peer is responsible for a region, then it is responsible for query computations that intersect that region and storing objects that fall into that region.
Distributed Spatial Queries

1. Assign responsibility for regions of space to peers in the system.

2. If a peer is responsible for a region, then it is responsible for query computations that intersect that region and storing objects that fall into that region.

3. Although we could subdivide space into a regular grid, it does not maintain scalability of a quadtree:
Distributed Spatial Queries

1. Assign responsibility for regions of space to peers in the system.

2. If a peer is responsible for a region, then it is responsible for query computations that intersect that region and storing objects that fall into that region.

3. Although we could subdivide space into a regular grid, it does not maintain scalability of a quadtree:
   - Many of the grid cells will be empty and will have to be needlessly processed.
Distributed Spatial Queries

1. Assign responsibility for regions of space to peers in the system.

2. If a peer is responsible for a region, then it is responsible for query computations that intersect that region and storing objects that fall into that region.

3. Although we could subdivide space into a regular grid, it does not maintain scalability of a quadtree:
   - Many of the grid cells will be empty and will have to be needlessly processed.
   - Quadtree only partitions space if it contains objects while the grid results in the partition of a region regardless of whether or not it contains any objects.
Assign responsibility for regions of space to peers in the system.

If a peer is responsible for a region, then it is responsible for query computations that intersect that region and storing objects that fall into that region.

Although we could subdivide space into a regular grid, it does not maintain scalability of a quadtree:

- Many of the grid cells will be empty and will have to be needlessly processed.
- Quadtree only partitions space if it contains objects while the grid results in the partition of a region regardless of whether or not it contains any objects.
- Hence, with a grid, where responsibilities of cells are assigned to peers, we will have load-balancing problems.
Instead of distributing a single tree, which would have a single root and thus a bottleneck, it is possible to break the tree into forests by forcing all objects/queries to start from a minimum level in the tree.
Up to a point, the client/server system provides query results in less time than the P2P system. However, with increased number of queries, the client/server system soon becomes significantly worse.
A publish/subscribe system, can be implemented as a spatial index where the objects are subscriptions and the queries are publications.
A publish/subscribe system, can be implemented as a spatial index where the objects are subscriptions and the queries are publications. A user inserts a subscription object into the system and waits for hits to return.
A publish/subscribe system, can be implemented as a spatial index where the objects are subscriptions and the queries are publications.

A user inserts a subscription object into the system and waits for hits to return.

Data is published by querying the regions that represent the data. Hits on the inserted subscription objects are returned to the subscribers.
Outline

1. Introduction
2. Design principles
3. Churn and Connectivity
4. Advanced Queries
5. Software models
Most of the examples in this tutorial were essentially key-based routing examples. The notion of data storage is separated and pushed up to a higher layer.
Most of the examples in this tutorial were essentially key-based routing examples. The notion of data storage is separated and pushed up to a higher layer. A DHT is then a storage system that uses key-based routing to locate the appropriate peer(s) to store the data.
Interfaces
Specific implementation details.

**routing** – The interface allows general access to all peers along the path to the destination.
Interfaces
Specific implementation details.

**routing** – The interface allows general access to all peers along the path to the destination.

**lookup** – The interface allows general access only to the destination peer.
Interfaces
Specific implementation details.

- **routing** – The interface allows general access to all peers along the path to the destination.
- **lookup** – The interface allows general access only to the destination peer.
- **storage** – The interface allows only get/put type operations.
OpenDHT (University of California at Berkeley) is a publicly accessible distributed hash table (DHT) storage service.
OpenDHT (University of California at Berkeley) is a publicly accessible distributed hash table (DHT) storage service. OpenDHT supports a narrow put-get-rm interface. The put RPC writes a key-value pair into the DHT, the get RPC retrieves one or more values previously put under a key, and the rm RPC removes values from the DHT. The current OpenDHT deployment limits a single value to 1 kilobyte in size.
Peer Review
For reliably detecting malicious peers.

1. The logic to be secured is put in the form of a finite state machine. All peers have this logic.
2. All peers regularly record the state of their finite state machine, including messages, etc, in a secure log. The log is signed and sent to a set of known “witness peers”.
3. All peers sign the messages that they exchange with other peers.
4. A peer can be audited at the request of some other peer. The requesting peer sends the set of signed messages to the witness peers. The witness peers rerun the finite state machine to check if the messages that were produced are the correct ones.
5. If the audit fails, then there exists a signed log plus signed messages that indicate the peer has done something different to the logic. Thus the peer can be ostracized on this basis.
Some lessons learnt

- No one approach to building peer-to-peer systems is the best way. Different approaches suit different applications and sometimes a mix of structured and unstructured design is applicable.
Some lessons learnt

- No one approach to building peer-to-peer systems is the best way. Different approaches suit different applications and sometimes a mix of structured and unstructured design is applicable.

- Practically all useful protocols have changed significantly since their first publication. Implementations vary a lot.
Some lessons learnt

- No one approach to building peer-to-peer systems is the best way. Different approaches suit different applications and sometimes a mix of structured and unstructured design is applicable.

- Practically all useful protocols have changed significantly since their first publication. Implementations vary a lot.

- Software reuse, of existing protocols and services, is particularly useful for prototype applications and demonstrators. Using a modular or component based approach allows transparent replacement of lower layer functionality; it can be particularly hard to get it right though.
Thank you for attending this tutorial.
Other Academics in our Research Group

Chris Leckie

Egemen Tanin
Thank you for attending this tutorial.
Other Academics in our Research Group

Chris Leckie
Egemen Tanin
Lars Kulik
Thank you for attending this tutorial.
Other Academics in our Research Group

Chris Leckie
Egemen Tanin
Lars Kulik
Santosh Kulkarni
Thank you for attending this tutorial.
Other Academics in our Research Group

Chris Leckie  Egemen Tanin  Lars Kulik  Santosh Kulkarni  Shanika Karunasekera