Network Science: Peer-to-Peer Systems

Ozalp Babaoglu
Dipartimento di Informatica — Scienza e Ingegneria
Università di Bologna
www.cs.unibo.it/babaoglu/

Introduction

- Peer-to-peer (P2P) systems are extremely popular and account for much of the current Internet traffic
- Distributed systems where all nodes are *peers* without distinction between servers and clients
- Each node can be both a server and a client:
  - May provide services to other peers
  - May consume services from other peers
- Very different from the client-server model

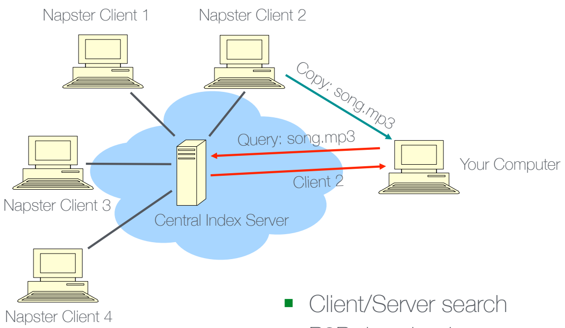
P2P History: 1969 — 1990

- The original Arpanet was P2P
- Each node was capable of:
  - Performing routing (locate machines)
  - Accepting ftp connections (file sharing)
  - Accepting telnet connections (distributed computation)

P2P History: 1999 — today

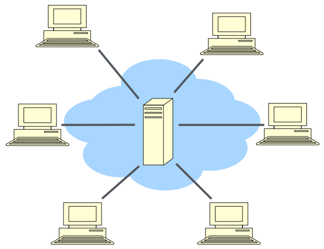
- The advent of Napster:
  - Jan 1999: the first version of Napster was released by Shawn Fanning, student at Northeastern University
  - July 1999: Napster Inc. founded
  - Feb 2001: Napster closed down

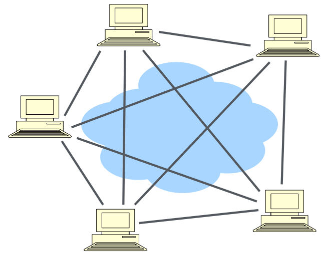
Napster



- Client/Server search
- P2P download
- Napster was not “pure P2P”

Client/Server vs. Peer-to-Peer

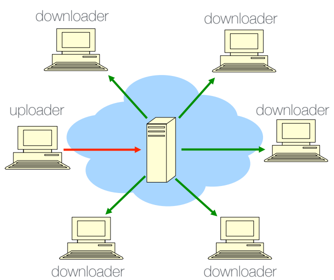




- Servers well connected to the “core” of the Internet
- Servers carry out critical tasks
- Clients only talk to servers

- Nodes located at the “periphery of the Internet”
- Tasks distributed across all nodes
- Clients talk to other clients

Example — Video sharing

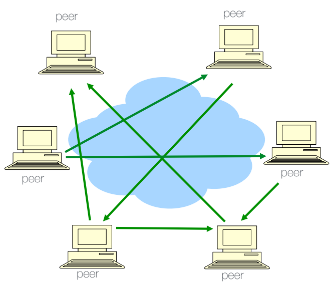


Client-Server: YouTube

Advantages
- Client can disconnect after upload
- Uploader needs little bandwidth
- Other users can find the file easily (just use search on server webpage)

Disadvantages
- Server may not accept file or remove it later (according to content policy)
- Whole system depends on the server (can be shut down)
- Server storage and bandwidth can be expensive

Example — Video sharing



Peer-to-peer: BitTorrent

Advantages
- Does not depend on a central server
- Bandwidth shared across nodes
- High scalability, low cost

Disadvantages
- Uploader must remain on-line to guarantee file availability
- Content is more difficult to find (no central directory)
- Freeloaders may cheat by only downloading without ever uploading

P2P vs. client-server

Client-server

Asymmetric: client and servers carry out different tasks

Global knowledge: servers have a global view of the network

Centralization: communications and management are centralized

Single point of failure: a server failure brings down the system

Limited scalability: servers easily overloaded

Expensive: server storage and bandwidth capacity is not cheap

Peer-to-peer

Symmetric: No distinction between nodes; they are peers

Local knowledge: nodes only know a small set of other nodes

Decentralization: no global knowledge, only local interactions

Robustness: several nodes may fail with little or no impact

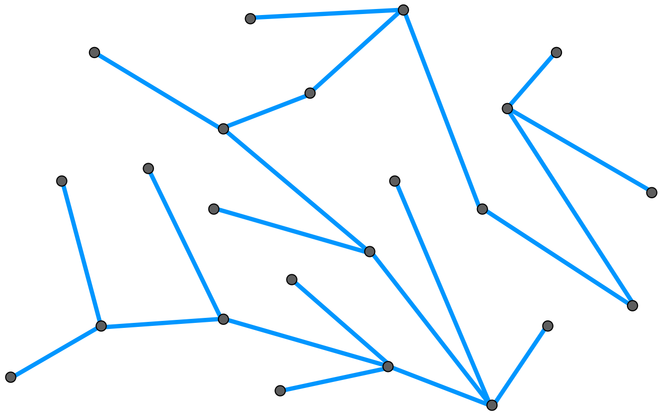
High scalability: high aggregate capacity, load distribution

Low-cost: storage and bandwidth are contributed by users

P2P and Overlay Networks

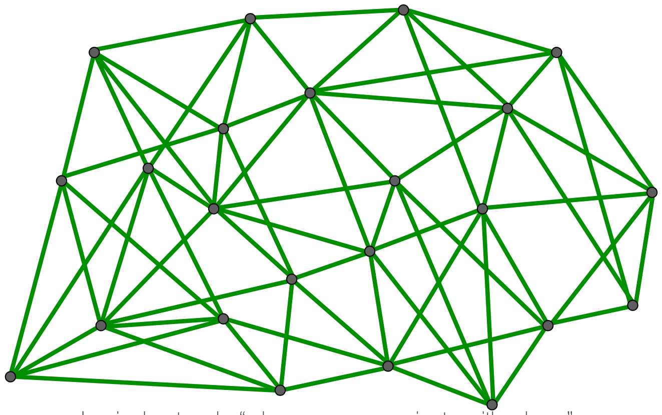
- Peer-to-Peer systems are usually structured as “overlays”
- Logical structures built on top of a physical routed communication infrastructure (IP) that creates the allusion of a completely-connected graph
- Links based on logical “knows” relationships rather than physical connectivity

Overlay networks



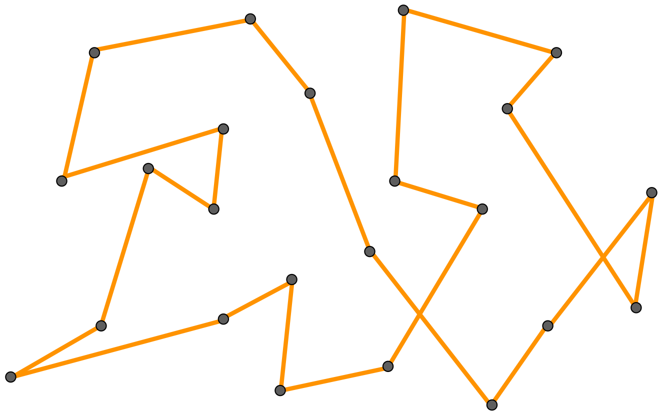
Physical network: “who has a communication link to whom”

Overlay networks



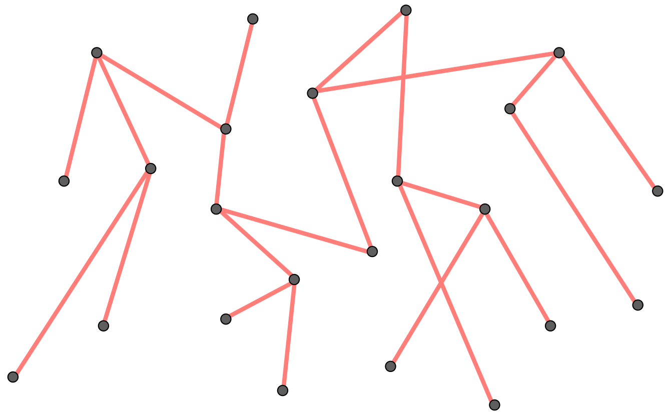
Logical network: “who can communicate with whom”
Typically fully-connected

Overlay networks



Overlay network (ring): "who knows whom"

Overlay networks



Overlay network (binary tree): "who knows whom"

P2P Environment

- Completely decentralized control with limited local states
- High latency, low bandwidth communication
- Churn
  - Nodes may disconnect temporarily
  - New nodes are continuously joining the system, while others leave permanently
- Security
  - P2P clients runs on machines under the total control of their owners
  - Malicious users may try to bring down the system
- Selfishness
  - Users may run hacked clients in order to avoid contributing resources

Why P2P?

- Decentralization enables deployment of applications that are:
  - Highly available
  - Fault-tolerant
  - Self organizing
  - Scalable
  - Difficult or impossible to shut down
- This results in a "grassroots" approach and "democratization" of the Internet

P2P Problems

- Overlay construction and maintenance
  - e.g., random, two-level, ring, etc.
- Data location
  - locate a given data object among a large number of nodes
- Data dissemination
  - propagate data in an efficient and robust manner
- Global reasoning with local information
  - maintain local views with small per node state
- Tolerance to churn
  - maintain system invariants (e.g., topology, data location, data availability) despite node arrivals and departures

P2P Applications

- Sharing of content:
  - File sharing, content delivery networks
  - Gnutella, eMule, Akamai
- Sharing of storage:
  - Distributed file systems
- Sharing of CPU time:
  - Parallel computing, Grid
  - Seti@home, Folding@home, FightAids@home (typically not pure P2P)

P2P Topologies

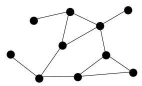
- Unstructured
- Structured
  - Centralized
  - Hierarchical
- Hybrid

Evaluating topologies

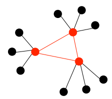
- Manageability
  - How hard is it to keep working?
- Information coherence
  - How authoritative is the info?
- Extensibility
  - How easy is it to grow?
- Fault tolerance
  - How well can it handle failures?
- Censorship
  - How hard is it to shut down?

Some Common Topologies

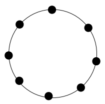
- **Flat unstructured**: a node can connect to any other node
  - only constraint: maximum degree *dmax*
  - fast join procedure
  - good for data dissemination, bad for location



- **Two-level unstructured**: nodes connect to a superpeer
  - superpeers form a small overlay
  - used for indexing and forwarding
  - high load on superpeer



- **Flat structured**: constraints based on node ids
  - allows for efficient data location
  - constraints require long join and leave procedures

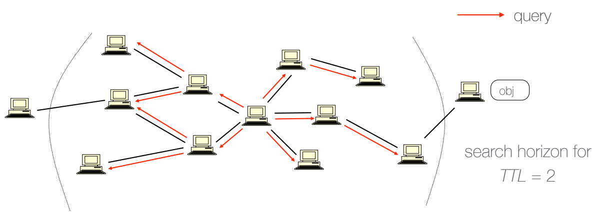


Data location in unstructured networks:

- *Problem*: find the set of nodes that store a copy of a given object
- *Flooding*: forward the search message to all neighbors
  - A search message contains either keywords or an object id
- Advantages:
  - simplicity
  - no topology constraints
- Disadvantages:
  - high network overhead (huge traffic generated by each search request)
  - flooding stopped by *Time-To-Live* (*TTL*) which produces search horizon
  - only applicable to small number of nodes

Data location in unstructured networks: Flooding

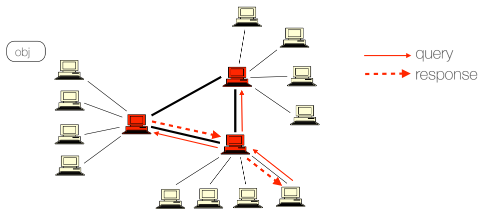
- Flooding in a flat unstructured network:



Objects that lie outside of the horizon are not found

Data location in unstructured networks: Superpeers

- Two-Level Overlay



Data location in structured networks: Key-Based Routing

- Structured networks: use a routing algorithm that implements Key-Based Routing (KBR) [Chord, Pastry, Overnet, Kad, eMule]
- KBR (also known as Distributed Hash Tables) works as follows:
  - nodes are (randomly) assigned unique node identifiers (Id)
  - given a key k, the node with the smallest Id greater than or equal to k among all nodes in the network is known as the *root* of key k
  - given a key k, a KBR algorithm can route a message to the root of k in a small number of hops, usually $O(\log n)$
  - the location of object $objectId$ is tracked by the root of key $objectId$
  - thus, one can find the location of an object by routing a message to the root of its $objectId$ and querying the root for the location of the object

Structured overlay network: Chord

Basics

- Each peer is assigned a unique ***m*-bit identifier** *id*.
- Every peer is assumed to store data contained in a file.
- Each file has a unique ***m*-bit key** *k*.
- Peer with smallest identifier $id \geq k$ is responsible for storing file with key k.
- $succ(k)$: The peer (i.e., node) with the smallest identifier $p \geq k$.

Note

All arithmetic is done modulo $M = 2^m$. In other words, if $x = k \cdot M + y$, then $x \bmod M = y$.

Structured overlay network: Chord

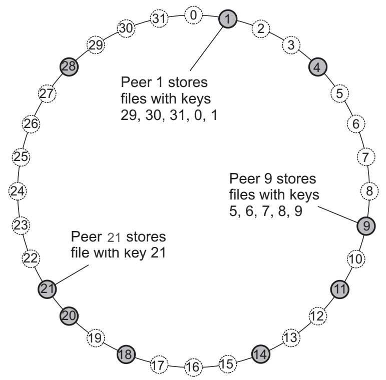
Basics

- Each peer is assigned a unique ***m*-bit identifier** *id*.
- Every peer is assumed to store data contained in a file.
- Each file has a unique ***m*-bit key** *k*.
- Peer with smallest identifier $id \geq k$ is responsible for storing file with key k.
- $succ(k)$: The peer (i.e., node) with the smallest identifier $p \geq k$.

Note

All arithmetic is done modulo $M = 2^m$. In other words, if $x = k \cdot M + y$, then $x \bmod M = y$.

Example



Efficient lookups

Partial view = finger table

- Each node p maintains a **finger table** $FT_p[]$ with at most m entries:

$$FT_p[i] = succ(p + 2^{i-1})$$

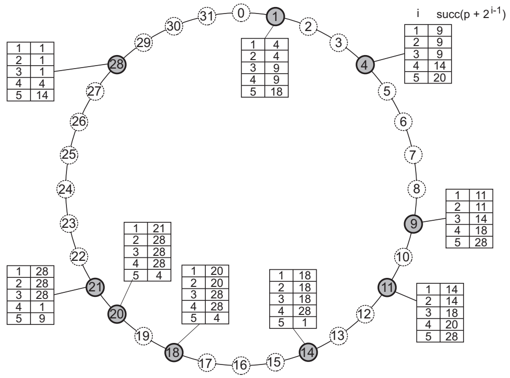
Note: $FT_p[i]$ points to the first node succeeding p by at least 2^{i-1}.

- To look up a key k, node p forwards the request to node with index j satisfying

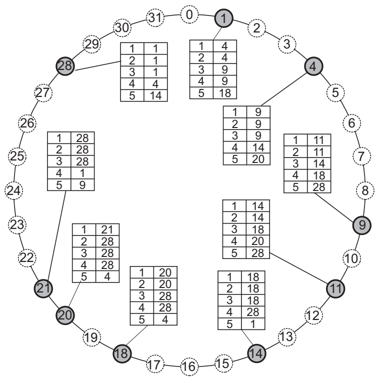
$$FT_p[j] \leq k < FT_p[j+1]$$

- If $p < k < FT_p[1]$, the request is also forwarded to $FT_p[1]$

Example finger tables

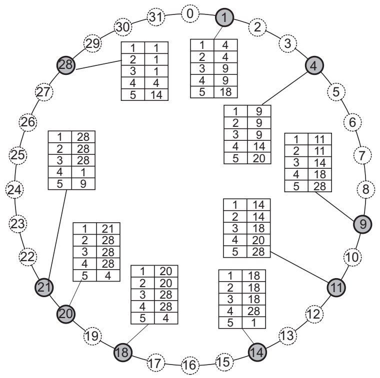


Example lookup: 15@4



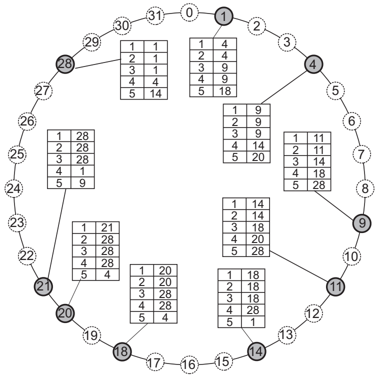
1. $FT_4[4] \leq 15 < FT_4[5]$ $\Rightarrow 4 \rightarrow 14$
2. $p = 14 < 15 < FT_p[1]$ $\Rightarrow 14 \rightarrow 18$

Example lookup: 15@4



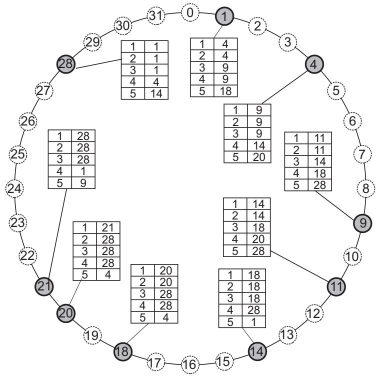
1. $FT_4[4] \leq 15 < FT_4[5]$ $\Rightarrow 4 \rightarrow 14$
2. $p = 14 < 15 < FT_p[1]$ $\Rightarrow 14 \rightarrow 18$

Example lookup: 15@4



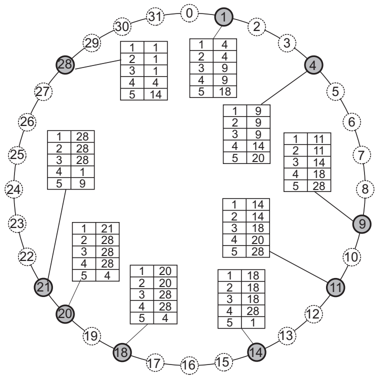
1. $FT_4[4] \leq 15 < FT_4[5]$
   $\Rightarrow 4 \rightarrow 14$
2. $p = 14 < 15 < FT_p[1]$
   $\Rightarrow 14 \rightarrow 18$

Example lookup: 22@4



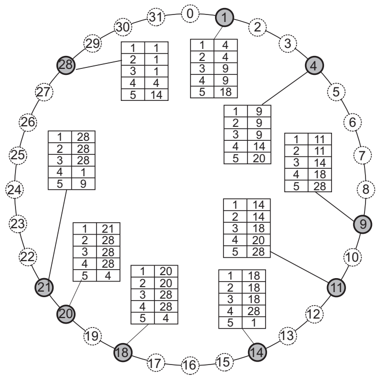
1. $FT_4[5] \leq 22$
   $\Rightarrow 4 \rightarrow 20$
2. $FT_{20}[1] \leq 22 < FT_{20}[2]$
   $\Rightarrow 20 \rightarrow 21$
3. $p = 21 < 22 < FT_{21}[1]$
   $\Rightarrow 21 \rightarrow 28$

Example lookup: 22@4



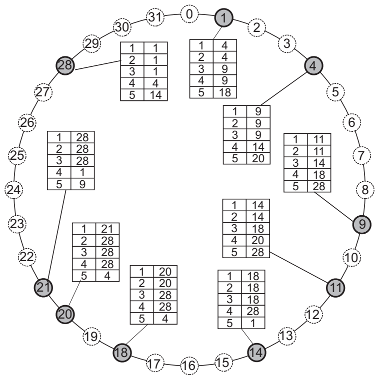
1. $FT_4[5] \leq 22$
   $\Rightarrow 4 \rightarrow 20$
2. $FT_{20}[1] \leq 22 < FT_{20}[2]$
   $\Rightarrow 20 \rightarrow 21$
3. $p = 21 < 22 < FT_{21}[1]$
   $\Rightarrow 21 \rightarrow 28$

Example lookup: 22@4



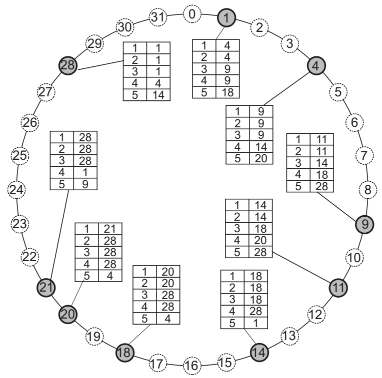
1. $FT_4[5] \leq 22$
   $\Rightarrow 4 \rightarrow 20$
2. $FT_{20}[1] \leq 22 < FT_{20}[2]$
   $\Rightarrow 20 \rightarrow 21$
3. $p = 21 < 22 < FT_{21}[1]$
   $\Rightarrow 21 \rightarrow 28$

Example lookup: 22@4



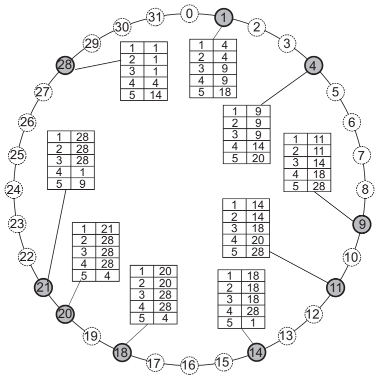
1. $FT_4[5] \le 22$
   $\Rightarrow 4 \rightarrow 20$
2. $FT_{20}[1] \le 22 < FT_{20}[2]$
   $\Rightarrow 20 \rightarrow 21$
3. $p = 21 < 22 < FT_{21}[1]$
   $\Rightarrow 21 \rightarrow 28$

Example lookup: 18@20



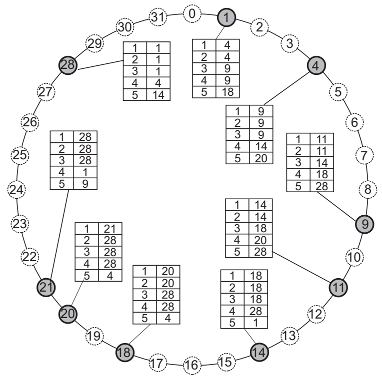
1. $p = 20 \not< 18 < FT_p[1]$
   $\not\Rightarrow 20 \rightarrow 21$
2. $FT_{20}[5] < 18$
   $\Rightarrow 20 \rightarrow 4$
3. $FT_4[4] \le 18 < FT_4[5]$
   $\Rightarrow 4 \rightarrow 14$
4. $p = 14 < 18 < FT_p[1]$
   $\Rightarrow 14 \rightarrow 18$

Example lookup: 18@20



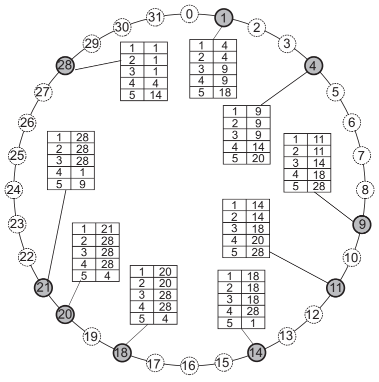
1. $p = 20 \not< 18 < FT_p[1]$
   $\not\Rightarrow 20 \rightarrow 21$
2. $FT_{20}[5] < 18$
   $\Rightarrow 20 \rightarrow 4$
3. $FT_4[4] \le 18 < FT_4[5]$
   $\Rightarrow 4 \rightarrow 14$
4. $p = 14 < 18 < FT_p[1]$
   $\Rightarrow 14 \rightarrow 18$

Example lookup: 18@20



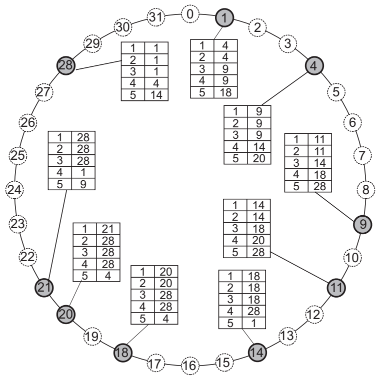
1. $p = 20 \not< 18 < FT_p[1]$
   $\not\Rightarrow 20 \rightarrow 21$
2. $FT_{20}[5] < 18$
   $\Rightarrow 20 \rightarrow 4$
3. $FT_4[4] \le 18 < FT_4[5]$
   $\Rightarrow 4 \rightarrow 14$
4. $p = 14 < 18 < FT_p[1]$
   $\Rightarrow 14 \rightarrow 18$

Example lookup: 18@20



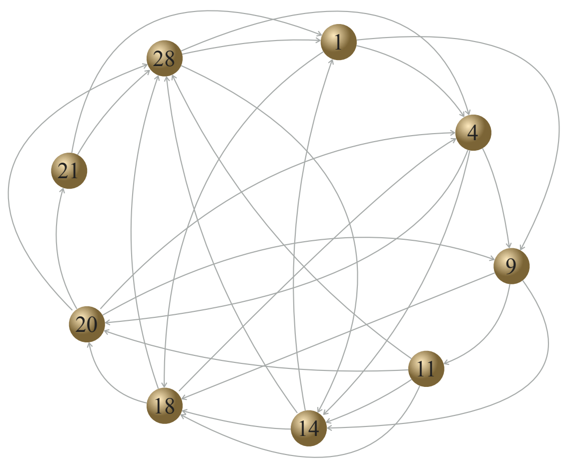
1. $p = 20 \not< 18 < FT_p[1]$ $\not\Rightarrow 20 \rightarrow 21$
2. $FT_{20}[5] < 18$ $\Rightarrow 20 \rightarrow 4$
3. $FT_4[4] \leq 18 < FT_4[5]$ $\Rightarrow 4 \rightarrow 14$
4. $p = 14 < 18 < FT_p[1]$ $\Rightarrow 14 \rightarrow 18$

Example lookup: 18@20

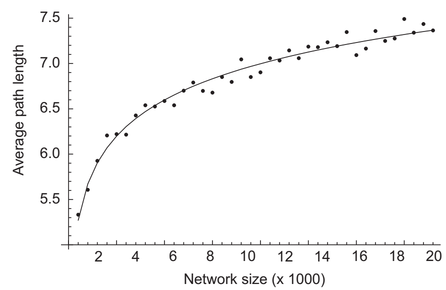


1. $p = 20 \not< 18 < FT_p[1]$ $\not\Rightarrow 20 \rightarrow 21$
2. $FT_{20}[5] < 18$ $\Rightarrow 20 \rightarrow 4$
3. $FT_4[4] \leq 18 < FT_4[5]$ $\Rightarrow 4 \rightarrow 14$
4. $p = 14 < 18 < FT_p[1]$ $\Rightarrow 14 \rightarrow 18$

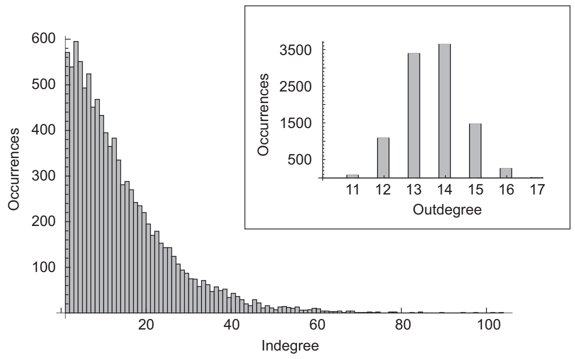
The Chord Graph



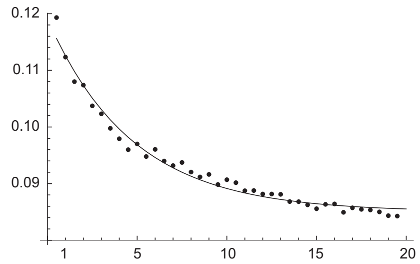
Chord: path lengths



Chord: degree distribution



Chord: clustering coefficient

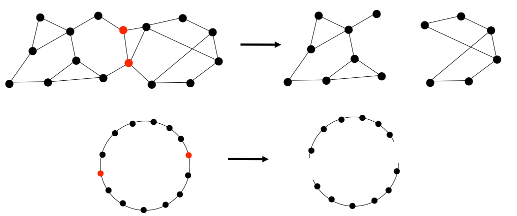


Note

CC is computed over undirected Chord graph; *x*-axis shows number of 1000 nodes.

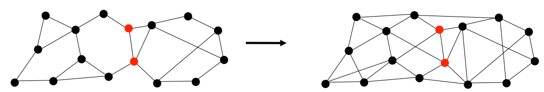
Effects of Churn

- Churn can have several effects on a P2P system:
  - data objects may be become unavailable if all replicas disconnect
  - routing tables may become inconsistent
    (e.g., entries may point to disconnected nodes)
  - the overlay may become partitioned if many nodes suddenly disconnect:

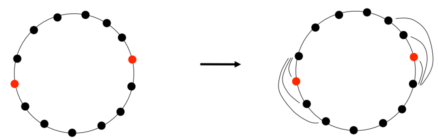


Churn — Preventing Partitions

- A naïve approach to preventing partitions is to increase the average node degree



- Ring partitions can be avoided by keep a list of successor nodes



Churn Tolerance

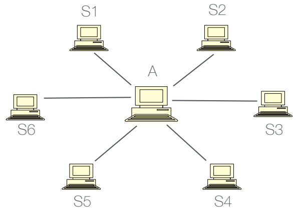
- Node arrivals and departures must not disrupt the normal behavior of the system
  - system invariants must be maintained
    - connected overlay (i.e., no partitions), low average path length
    - data objects accessible from anywhere in the network
- Two types of churn tolerance:
  - dynamic recovery: ability to react to changes in the overlay to maintain system invariants (e.g., heal partitions)
  - static resilience: ability to continue operating correctly before adaptation occurs (e.g., route messages through alternate paths)

Security

- Security in P2P systems is hard to enforce:
  - Users have full control of their computers
  - Modified clients may not follow the standard protocol
  - Data may be corrupted
  - Private data stored on remote computers may disclosed

Security — Weak identities

- The user may leave the system and rejoin it with a new identity (different user id)
- If an attack is detected, the attacker can re-enter the system with a new id
- An attacker may create a large number of false identities (Sybil attack)



Example of Sybil attack:

- Nodes S1 to S6 are actually 6 instances of the P2P client running on the same machine
- The attacker can intercept all traffic coming from or going to node A

Security — Strong identities

- The user cannot change its identity
- Solution: use a centralized, trusted Certification Authority (CA)
  - Each new user must obtain an identity certificate
  - The certificate is digitally signed by the CA, whose public key is known by all users
  - A certificate cannot be forged (require the CA's private key)
  - To prove his identity, a user signs a message with his private key, and attaches the corresponding certificate signed by the CA
- Strong identities prevent Sybil Attacks
- If an attacker is caught, it cannot easily rejoin the system

Security — Weak vs. Strong identities

- Strong identities require a centralized CA
  - New nodes must contact the CA before joining the network:
    - The CA response may be slow
    - If the CA is unavailable, new nodes cannot join
  - The security of the system depends on CA:
    - The CA must correctly verify the identity of the requester
    - The CA's private key must be secret
- Many P2P systems use weak identities
  - IP addresses already gives some identity information
  - Some systems ensure anonymity