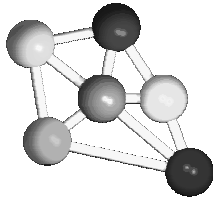


Peer-to-peer under the hood



David Göthberg
Studiegången 7-208
416 81 Göteborg
Sweden

Tel: +46-31-45 98 56
david@randpeer.com
www.randpeer.com

Peer-to-peer under the hood



- This talk will be about p2p algorithms. Not about specific p2p software.
- David has researched p2p algorithms since 1997 and full time since autumn 2000.
- David is making a p2p-programming library. So that other programmers can build advanced p2p applications easily, without having to spend years on research first.

Note that slides with a red X are not shown during talks but are added in printed and online versions to make the slides more comprehensible.

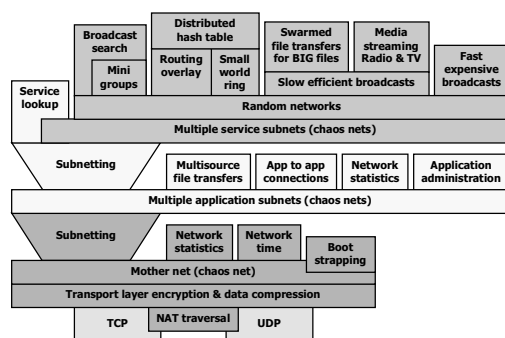
Ideal p2p systems

- Fully distributed
- Totally serverless
- Fully scalable
- Globally searchable
- Bandwidth efficient
- Robust
- Encrypted
- Stealthy
- Preferably anonymous

Target applications

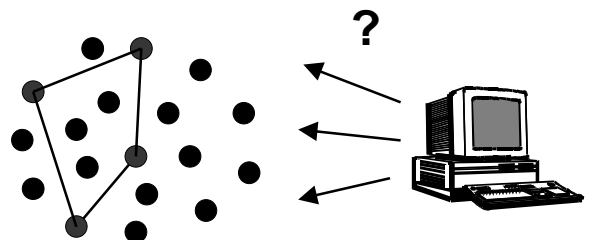
- File sharing
- Chat systems
- Instant messaging
- Internet telephoning
- Internet games
- Radio & TV (sent from home users computers)
- Distributed calculation systems
- And many more applications

The Randpeer layer model



Bootstrapping

The Internet



How to find the other peers on the Internet?

Bootstrapping 2



When a node joins a p2p network it needs some way to find the network.

Centralised approach:

- Start servers / reflectors. Reachable at fixed IPs or fixed DNS names.

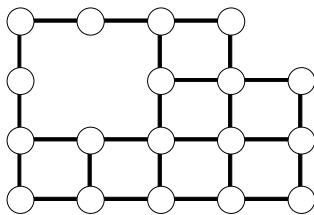
Distributed serverless approach:

- Start lists: Nodes remember nodes that are online often and much with the same IP.
- Old nodes use their start list to find the network.
- New nodes get a start list in the install package.
- If the start list is too old, get a fresh one from the web or IRC or from a friend via email or diskette.

Internet problems

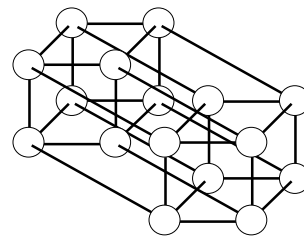
- High node churn = Home users often turn their computers off
- Erroneous ISP routers = All nodes can not reach each other
- Firewalls and NAT routers = All nodes can not reach each other

Two dimensional grid



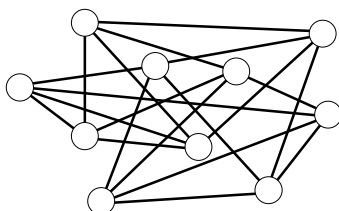
- Easy to understand
- Tricky to code and maintain
- Inefficient (Long jump distances)

Hypercube



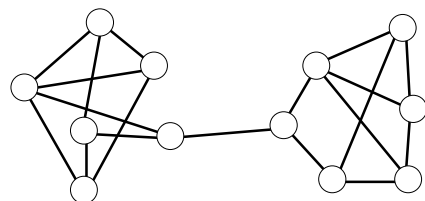
- Easy to understand
- Tricky to code and maintain
- Efficient (Short jump distances)

Random network (Small world network)



- Slightly tricky to understand
- Easy to code and maintain
- Efficient (Short jump distances)

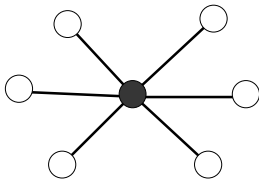
Netsplit prevention



We need random (far away) node addresses:

- Node address announcements
- Random walkers
- Chaos nets (David's own solution)

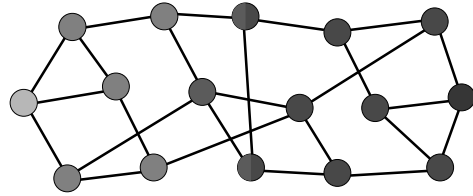
Flooding / Overloading



Methods to prevent overloading of nodes:

- Round based algorithms
- Throttling
- Randomised algorithms (to spread load evenly)

Throttled balanced broadcasts



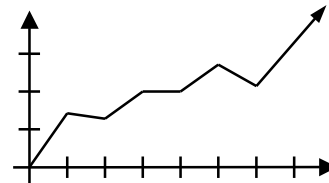
Often a node doesn't need to see *all* broadcasted messages, only *enough* messages

Throttled balanced broadcasts 2 ^X

Often nodes don't need to broadcast to all nodes, instead nodes need to receive enough messages:

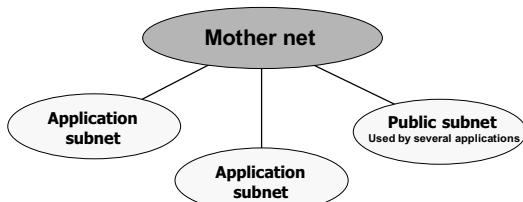
- Jump counter instead of TTL (Time To Live) (Count up instead of down)
- Youngest message is forwarded
- Each originating node reaches about the same amount of nodes
- Forward some messages per link per round
- Different message types can / should be handled separately = "slotted broadcasting"

Network statistics



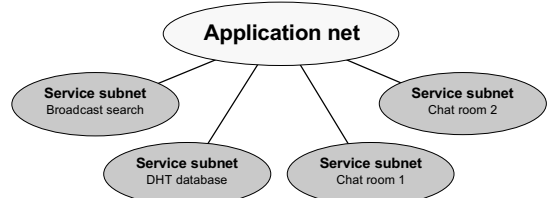
- Number of nodes online
- Number of unique nodes per month
- Network averages, medians, max and min
- Network totals (sums)
- Network time

Application subnets



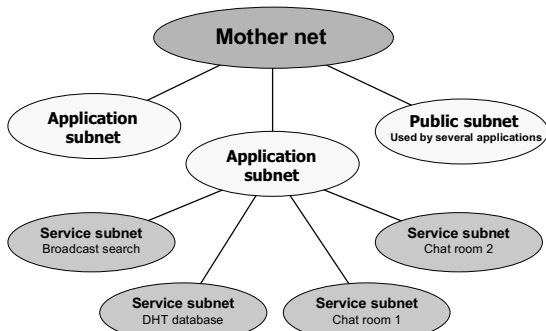
- New nets can bootstrap off old nets
- Small nets help each other with bootstrapping
- Old shrinking nets can bootstrap of new big nets
- "Public subnets" for co-operating applications

Service subnets (Rendezvous service)



- Chat rooms and file transfer swarms as subnets
- User IDs and servers as subnets
- Algorithms / protocols as subnets: Plug in new kinds of search databases later on!

Subnetting overview



Subnetting

If you don't know the name and the key you should not be able to "see" the subnet:

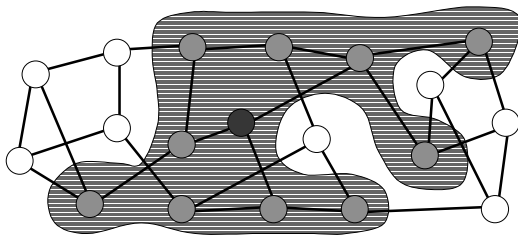
subnet address = hash(subnet name + subnet key)

Huge subnets must be able to coexist with small subnets.

Ways to implement subnet lookup (rendezvous):

- Broadcast search
- Distributed Hash Table (DHT)
- Subnets under subnets

Broadcast search



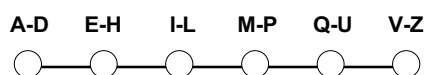
Preferably uses throttled balanced broadcasts

Broadcast search 2

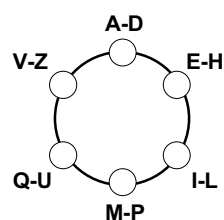
Gnutella, Kazaa and many more:

- Uses a random network
- Search limited by a TTL or jump counter
- Search horizon = Only reaches the closest nodes
- Uses very much bandwidth
- Very good at handling complex searches such as "ccc AND video AND NOT Berlin"
- Width first or depth first search
- Some optimisations can increase the horizon: supernodes, search filters, mini groups

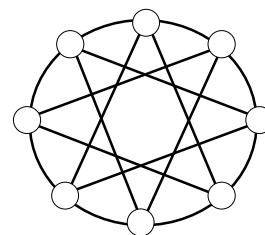
Distributed Hash Tables 1



Nodes organised in address order in a list or a ring or a hypercube



Distributed Hash Tables 2



Routing overlay = Shortcuts across the ring make lookups much faster.

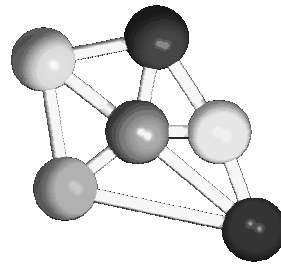
Distributed Hash Tables 3



Chord, Pastry, Kademlia and many more DHTs:

- Nodes often connected in a ring
- Shortcuts = Routing overlay
- Some DHTs are hypercubes instead of rings
- Item key = hash(item name)
- Search among millions of nodes = If one node has what you want, you find it
- Can not do complex searches, only key lookup
- Complex to implement and vulnerable to attacks
- Problems with unevenly distributed data (Hashing only partly fixes the problem)

The end



www.randpeer.com