# Peer-to-peer under the hood

**An in-depth look at p2p algorithms**

## Talker's reference

David Göthberg
Studiegången 7-208
416 81 Göteborg
Sweden

Tel: +46-31-459856
david@randpeer.com
www.randpeer.com

Stuff to bring along to the talk:
- This paper (talker's reference).
- A printout on paper of the slides.
- A CD with the slides in several formats (PNG, GIF, HTML). Also have the slides available on the Internet since I might lose / forget the CD.
- Lot's of business cards!

Remember:
- Preferably use a handheld or body mounted microphone.
- Talk slowly and clearly!

## Start of slides

Present the talk and myself:
- Hello everyone, I am David.
- I am going to talk about p2p-algorithms. Not about specific p2p-software.
- (This will only be a very simplified overview of p2p-algorithms.)
- I have researched p2p algorithms since 1997 and full time since the year 2000.
- Before that I used to work with Internet communication and computer security in embedded systems. (Internet in cars and other machinery.)
- I finished my p2p-research this summer.
- Now I am making a p2p-programming library. So that other programmers can build advanced p2p-applications easily, without having to spend years on research first.
- I come from Sweden but I also talk German so you can ask questions during this talk and after this talk in German if you like.

Ideal p2p systems:
- Fully distributed, totally serverless, fully scalable, globally searchable, bandwidth efficient, robust, encrypted, stealthy, preferably anonymous.
- Zero cost to run!

Target applications:
- We can not do network file storage (push) yet. We need very good trust systems for that, and we do not have that yet.

The Randpeer layer model:
- "This is what I am going to talk about today".
- Many current p2p algorithms are monolithic. I prefer to find algorithms that each solve one or more problems well and then layer them to solve the whole thing.
- Similar to how the IP-stack has layers that each solves one or more of the Internet communications problem.

Bootstrapping:
- Centralised approach:
  - Start servers / reflectors, reachable at fixed IPs or fixed DNS names.
  - (New IPs and new DNS names can be broadcasted onto the network with a signed message.)
- Distributed serverless approach:
  - Start list: 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 a friend via email or diskette.
  - Start lists as html files. For easy publication onto the web by users. Contains IP, port, crypto key and software version for each node in the list.

Internet problems:
- Node churn. (Home user computers usually do not run 24/7.)
- Any given node can only reach say 98% of all other nodes. (For instance, I can reach nodes in southern but not northern Lebanon!) This is due to erroneous ISP routers and/or network congestion.
- Firewalls and NAT routers.

Basic network layout: Two-dimensional grid
- Easy to understand
- Tricky to code
- Inefficient (Long jump distances)

Basic network layout: Hypercube (Multidimensional grid)
- Easy to understand
- Tricky to code
- Efficient (Short jump distances)
- Many DHTs like Chord and Kademlia are sort of multidimensional grids.

Basic network layout: Random network (Small world network)
- Slightly tricky to understand
- Easy to code
- Efficient (short jump distances)
- Many p2p networks are random networks, for instance Gnutella.

Netsplit prevention: We need random (far away) node addresses:
- Node address announcements (pings)
- Random walkers
- Chaos nets (David's own solution)

Flooding / overloading:
- Round based algorithms
- Throttling
- Randomised algorithms can spread load evenly

Throttled balanced broadcasts:
- It is both round based, throttled and randomised!
- Normally runs on top of random networks or small world networks.
- Often nodes don't need to send to all nodes, instead nodes need to receive enough messages.
- If TTL set to 6 and we have 5 connections = 4000 nodes reached.
- What if all nodes are broadcasting? Flooding...
- Or only one node is broadcasting? Most nodes will not receive any message...
- Jump counter instead of TTL (Time To Live counter).
- Can be set to forward several messages, say the 4 youngest messages each round.

Network statistics:
- Number of nodes online:
  - Use a tree.
  - Use the average distance in DHTs.
  - Use David's statistical method.
- We can also measure number of unique nodes per month!
- Network averages, medians, max and min. Tree or fuzzy tree! (Multirooted tree.)
- Network totals. Use a tree or total = average * number of nodes.
- Network time

Application subnets / joint bootstrapping:
- (Also called "rendezvous service".)
- New nets can bootstrap of old ones.
- Several small nets can help each other with bootstrapping.
- Old shrinking nets can bootstrap of newer big nets.
- "Public subnets" for co-operating applications.

Service subnets / rendezvous service:
- Also called "rendezvous service".
- Chat rooms and torrents as subnets/services.
- User IDs as subnets. (For instant messaging, chat and Internet telephoning etc.)
- "Servers" as services / subnets.
- P2p algorithms/protocols as subnets: We can plug in new kinds of search databases later on!

Subnetting overview:
- How they work, how they can be implemented:
  - Subnet address = hash( subnet name + subnet key )
  - About coexistence of huge subnets with small subnets.
  - How to find/announce subnets: Broadcast search, DHTs or subnets under subnets.
  - More about subnets under subnets.

Broadcast search:
- Simple old Gnutella style.
- Normally runs on top of a random network or a small world network.
- TTL or jump counter. Jump counter better = throttled balanced broadcast.
- Search horizon.
- Uses very much bandwidth.
- Width first or depth first search.
- Very good at handling complex searches such as "CCC and video and not Berlin"
- Supernodes.
- Search filters from neighbours (bloom filters).
- Mini groups. (Perhaps to messy to talk about?)

Distributed Hash Tables (DHTs) 1:
- Nodes organised in address order. As a list or a ring or a hypercube.
- Two reasons for rings:
  - First and last node in the "list" can backup each other.
  - Easier addressing in some cases.
- Item key = hash( item name )

Distributed Hash Tables (DHTs) 2:
- Shortcuts = Routing overlay.
- Some DHTs has no ring and only the routing overlay, often hypercubes.
- Search among millions of nodes = If one node has what you want, you find it.
- Not good for 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:
- This was of course just a very brief overview.
- I will be available outside afterwards for more questions and for discussion.
- So, any questions?
- (Those of you that want to really discuss p2p algorithms are of course welcome to join me for dinner later tonight.)

## Extras, left out of the talk due to time constraint

**Randomised algorithms:**
- P2p networking is chaotic. We can create order in chaos by using randomness. Because near perfect randomness means near perfect order.
- Most of my randomised p2p algorithms are of the type that others call "emergent networks".
- Simple example: Randomised work assignment.

**Perhaps more about chaos nets:**
- Throttled balanced broadcasts works on chaos nets too.
- Mention that some stuff can be run directly on top of the chaos net without needing a random network at all.

**File transfers:**
- Multi source downloading.
- Swarming. (Upload while downloading, Bittorrent style.)

**Media streaming (Radio & TV):**
- Nodes that download can also upload.

**File hashing:**
- Protects against fake/damaged files
- Simple hashes.
- Hash lists.
- Hash trees. (Hash lists are easier to code and usually good enough.)

**Network encryption and key handling:**
- Oh dear, this will be messy to explain, perhaps avoid the subject?
- Perhaps my talk next year can be about this?

**Stealth:**
- Random ports.
- Caller talks first.
- Server unique hello message.
- Stealthy UDP. (And port knocking.)
- Look alike or look random.
- Encrypt from first byte.
- Traffic analysis.

**Anonymity:**
- Onion routing / Encrypted telescopic tunnels.
- Darknets.
- CA or not CA, that is the question.
- Web of trust can be useful.
- Layered: Jump over the darknet, then web of trust nodes or CA nodes, then random nodes.

**Trust systems:**