Distributed Algorithms
Peer-to-Peer Systems

Alberto Montresor

Università di Trento

2018/10/18

Acknowledgments: J. Chase, K. Ross, D. Rubenstein, P. Maymounkov, D. Mazieres, D. Carra, B. Cohen, A. Legout, V. Samprati, K. Tamilmani, N. Liogkas, I. Mohomed, D. Epema

Table of contents

Introduction

Definition

A peer-to-peer system is a collection of peer nodes, that act both as servers and as clients

- Provide resources to other peers
- Consume resources from other peers

Characteristics

- Put together resources at the edge of the Internet
- Share resources by direct exchange between nodes
- Perform critical functions in a decentralized manner

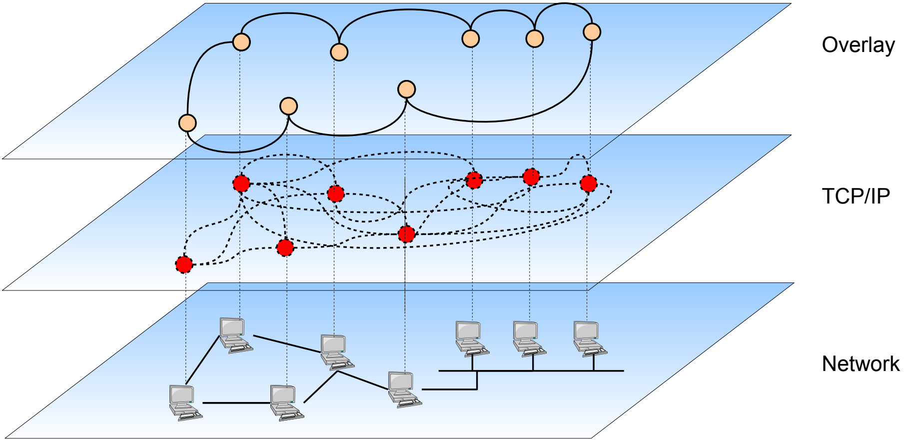
Motivation for P2P

- Cost-effective
  - Exploit the “dark matter” of the Internet constituted by “edge” resources
- No central point of failure
  - Control and resources are decentralized
- Scalability
  - Since every peer is alike, it is possible to add more peers to the system and scale to larger networks

It's a broad area. . .

- P2P file sharing
  - Gnutella
  - eMule
  - BitTorrent
- P2P communication
  - Instant messaging
  - Voice-over-IP: Skype
- P2P computation
  - Seti@home
- DHTs & their apps
  - Chord, CAN, Kademlia, . . .
- P2P wireless
  - Ad-hoc networking

Overlay networks



Overlay networks

Virtual edge

- TCP connection
- or simply a pointer to an IP address

Overlay maintenance

- Periodically ping to make sure neighbor is still alive
- Or verify liveness while messaging
- If neighbor goes down, may want to establish new edge
- New node needs to bootstrap

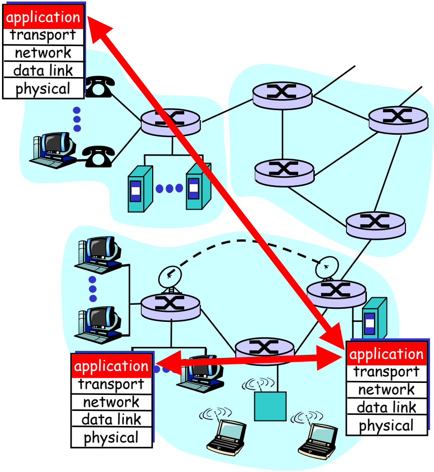
Overlay networks

Tremendous design flexibility

- Topology
- Message types
- Protocols
- Messaging over TCP or UDP

Underlying physical net is transparent to developer

- But some overlays exploit proximity



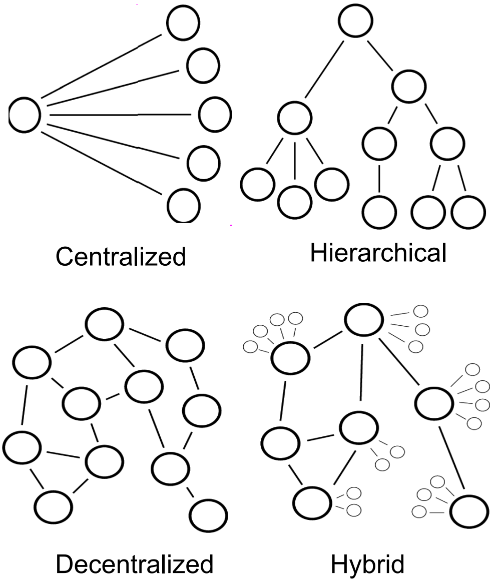
Overlay Topology

Unstructured:

- No explicit topology
- Observed rather than engineered
- Example: Gnutella, BitTorrent

Structured:

- An explicit "shape" is maintained
- Examples: Rings, Trees, DHTs
- Random topologies are "structured" as well



Criteria for topology selection

- Does it simplify location of data?
- Does it
  - balance the load, if nodes are equal?
  - exploit heterogeneity, otherwise?
- Is it robust?
  - Can it work if part of it is suddenly removed?
  - Can it be maintained in spite of churn?
- Has some correspondence with the underlying network topology?
  - Proximity (latency-based)
  - e.g., Pastry, Kazaa, Skype

Table of contents

Distributed Hash Table (DHT)

A peer-to-peer algorithm that offers an associative Map interface:

- *put*(KEY k, VALUE v): associate a value v to the key k
- VALUE *get*(KEY k): returns the value associated to key k

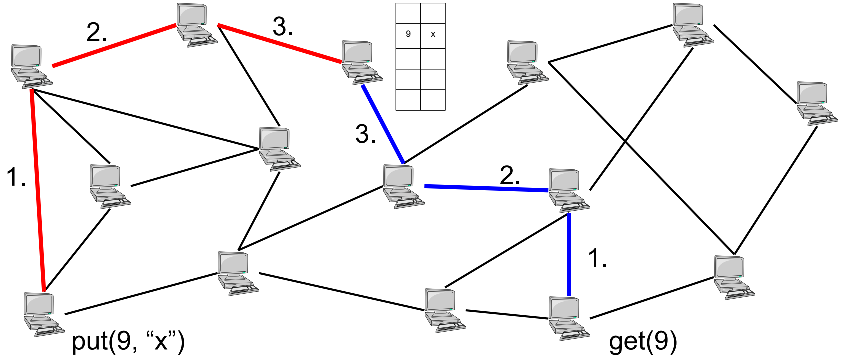
(Distributed) Hash Tables:

- Hash tables map keys to memory locations
- Distributed hash tables map keys to nodes

Organization:

- Each node is responsible for a portion of the key space
- Messages are routed between nodes to reach responsible nodes
- Replication used to tolerate failures

Routing in DHTs



DHT Implementations

- The founders (2001):
  - Chord
  - CAN
  - Pastry
  - Tapestry
- The ones which are actually used:
  - Kademlia and its derivatives (up to 4M nodes!)
    - BitTorrent
    - Kad (eMule)
    - The Storm Botnet
  - Cassandra DHT
    - Part of Apache Cassandra
    - Initially developed at Facebook
- The ones which are actually used, but we don't know much about:
  - Microsoft DHT based on Pastry
  - Amazon's Dynamo key-value store

Step 1: From Keys and Nodes to IDs

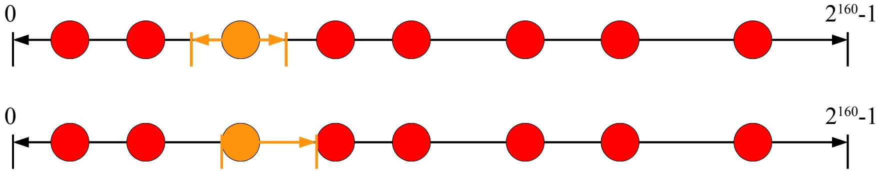
- Keys and nodes are represented by identifiers taken from an ID space
  - Key identifiers: computed through an hash function (e.g., SHA-1)
    - e.g., $ID(k) = SHA1(k)$
  - Node identifiers: randomly assigned or computed through an hash function
    - e.g., $ID(n) = SHA1(\text{IP address of } n)$

Why?

- Very low probability that two nodes have exactly the same ID
- Nodes and keys are mapped in the same space

Step 2: Partition the ID space

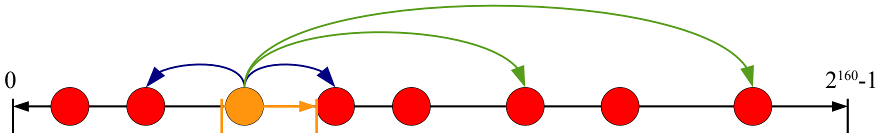
- Each node in the DHT stores some k, v pairs
- Partition the ID space in zones, depending on the node IDs:
- A pair (k, v) is stored at the node n such that (examples):
  - its identifier $ID(n)$ is the closest to $ID(k)$;
  - its identifier $ID(n)$ is the largest node id smaller than $ID(k)$



Step 2: Build overlay network

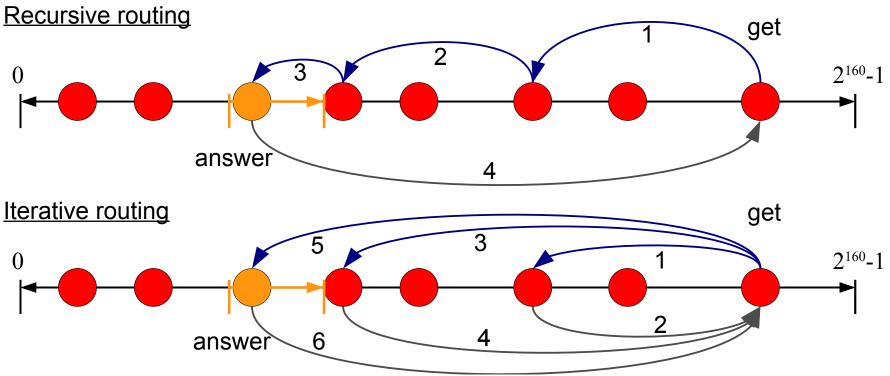
Each node has two sets of neighbors:

- Immediate neighbors in the key space (leafs)
  - Guarantee correctness, avoid partitions
  - If we had only them, linear routing time
- Long-range neighbors
  - Allow sub-linear routing
  - If we had only them, connectivity problems



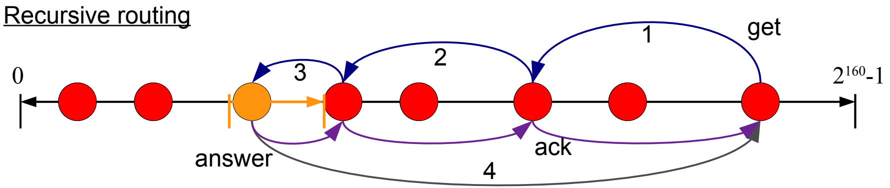
Step 3: Route puts/gets through the overlay

- Recursive routing: the initiator starts the process, contacted nodes forward the message
- Iterative routing: the initiator personally contact the nodes at each routing step



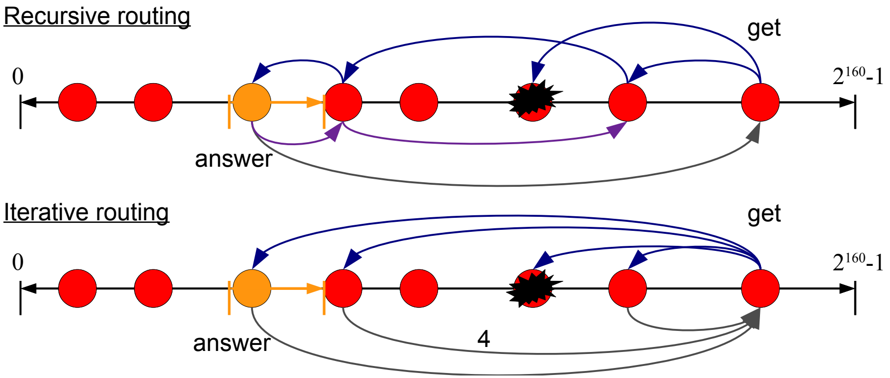
Routing around failures (1)

- Under churn, neighbors may have failed
- To detect failures, acknowledge each hop (recursive routing)



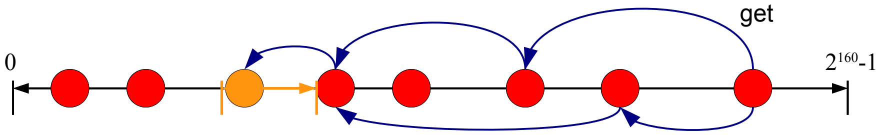
Routing around failures (2)

- If we don't receive ack or response, resend through a different neighbor



Routing around failures (3)

- Must compute timeouts carefully
  - If too long, increase put/get latency
  - If too short, get message explosion
- Parallel sending could be a design decision – see Kademlia

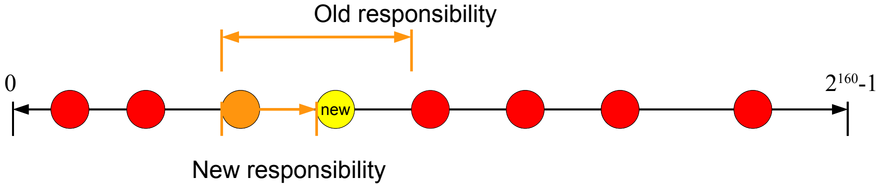


Computing good timeouts

- Use TCP-style timers
  - Keep past history of latencies
  - Use this to compute timeouts for new requests
- Works fine for recursive lookups
  - Only talk to neighbors, so history small, current
- In iterative lookups, source leads the entire lookup process
  - Must potentially have good timeout for any node

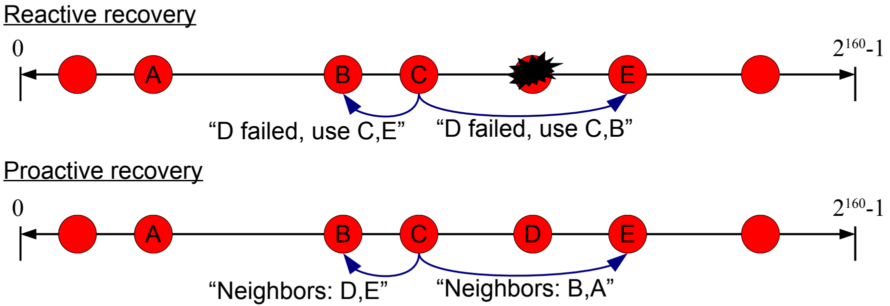
Recovering from failures

- Can't route around failures forever
  - Will eventually run out of neighbors
- Must also find new nodes as they join
  - Especially important if they're our immediate predecessors or successors



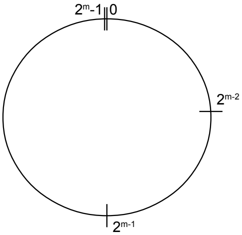
Recovery from failures

- Reactive recovery
  - When a node stops sending acknowledgments, notify other neighbors of potential replacements
- Proactive recovery
  - Periodically, each node sends its neighbor list to each of its neighbors



Chord

- ID space: uni-dimensional ring in $[0, 2^m - 1]$ $(m = 160)$
- Routing table size: $O(\log n)$
- Routing time: $O(\log n)$



Bibliography

I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications.
In *Proc. of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 149–160, San Diego, CA, 2001. ACM Press.

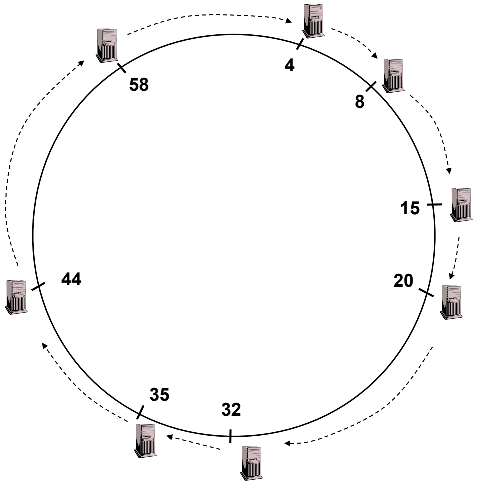
http://www.disi.unitn.it/~montreso/ds/papers/chord.pdf

Identifier mapping

Example:

- Node 8 maps $[5, 8]$
- Node 15 maps $[9, 15]$
- Node 20 maps $[16, 20]$
- ...
- Node 4 maps $[59, 4]$

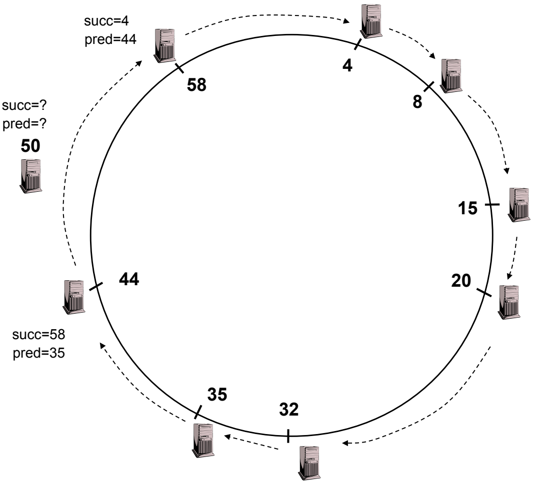
- Random ID assignment
- Each node maintains a pointer to its successor



Join procedure (1)

Example:

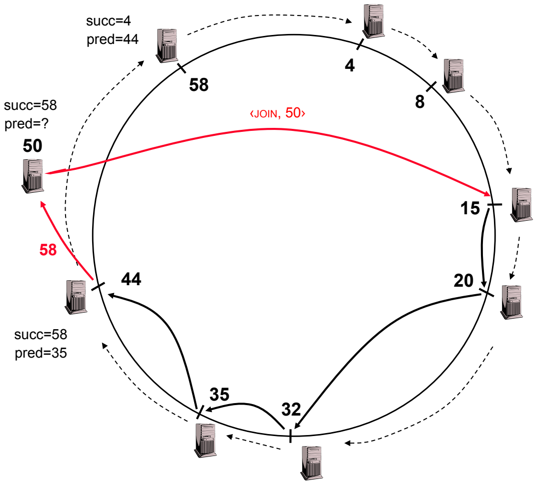
- Node with $id = 50$ joins the ring
- Node 50 needs to know at least one node already in the system
- Assume known node is 15



Join procedure (2)

Example:

- Node 50: send ⟨JOIN, 50⟩ to node 15
- Message is routed to node 44
- Node 44: returns node 58
- Node 50: updates its successor to 58



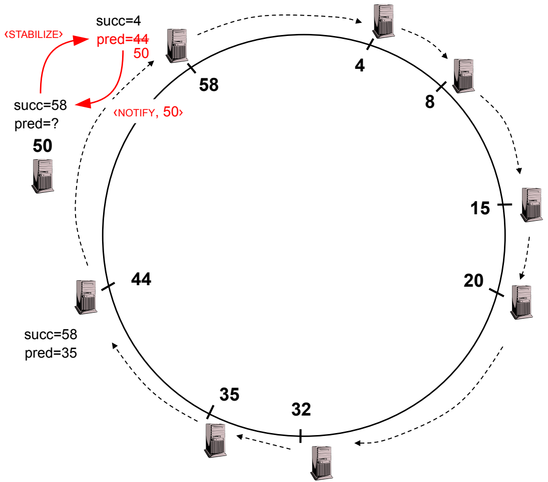
Stabilization

- Periodically, each node A:
  - sends a $\langle$STABILIZE$\rangle$ message to its successor B
- Upon receiving $\langle$STABILIZE$\rangle$ message from A, node B:
  - returns its predecessor $B' = \mathit{pred}(B)$ to A by sending a $\langle$NOTIFY$, B'\rangle$ message
  - updates its predecessor to A, if A is between B' and B
- Upon receiving $\langle$NOTIFY$, B'\rangle$ message from B, node A:
  - updates its successor to B', if B' is between A and B

Join procedure (4)

Example:

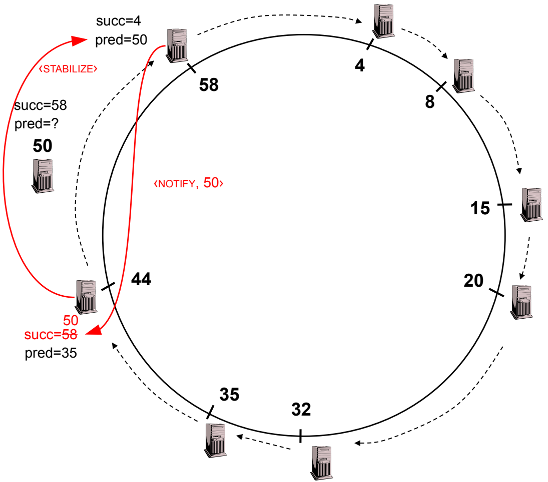
- Node 50: send ⟨STABILIZE⟩ to node 58
- Node 58: update predecessor to 50
- Node 58: send ⟨NOTIFY, 50⟩ back



Join procedure (5)

Example:

- Node 44: send ⟨STABILIZE⟩ to its successor node 58
- Node 58: replies with ⟨NOTIFY, 50⟩
- Node 44: updates it successor to 50

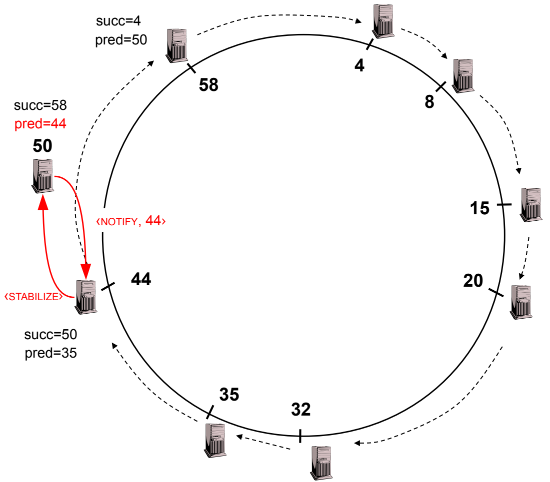


Join procedure (6)

Example:

- Node 44: send ⟨STABILIZE⟩ to its new successor, node 50
- Node 50: updates it predecessor to 44

This completes the joining operation!

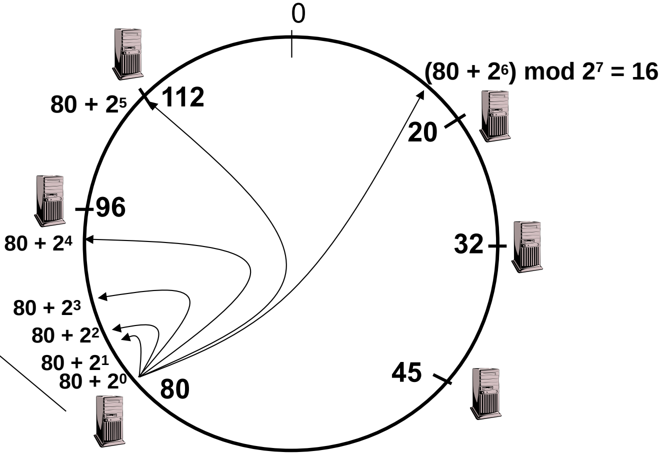


Achieving efficiency

- Chord requires each node to keep a finger table containing up to m entries
- The i-th entry $(0 \leq i \leq m-1)$ of node n will contain the address of the successor of $(n + 2^i) \bmod 2^m$
- Fingers are used in routing to reduce the number of hops to $O(\log N)$

Achieving efficiency

i	*ft[i]*
0	96
1	96
2	96
3	96
4	96
5	112
6	20



Achieving robustness

- To improve robustness, each node maintains $k > 1$ immediate successors instead of only one
- In the ⟨NOTIFY⟩ message, node A can send its $k - 1$ successors to its predecessor B
- Upon receiving the ⟨NOTIFY⟩ message, B can update its successor list by concatenating the successor list received from A with A itself

Optimizations

- Reduce latency
  - Choose finger that reduces expected time to reach destination
  - Choose the closest node from range $[n + 2^{i-1}, n + 2^i)$ as successor
- Accommodate heterogeneous systems
  - Multiple virtual nodes per physical node

CAN

- Associate to each node and item a unique ID in an d-dimensional Cartesian space on a d-torus
- Routing table size is constant: $O(d)$
- Guarantees that a key is found in at most $d \cdot n^{1/d}$ steps, where n is the total number of nodes

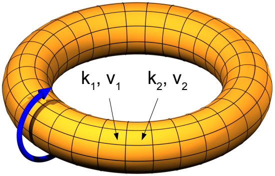


Figure: A 2-torus

Bibliography

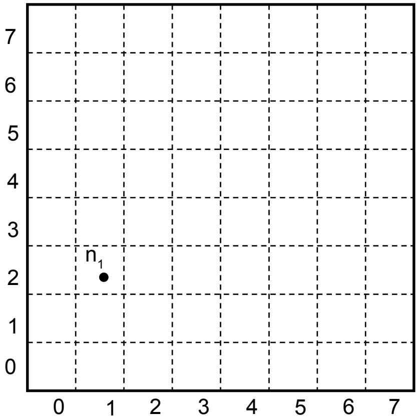
S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network.
In *Proc. of SIGCOMM'01*, pages 161–172, San Diego, California, USA, 2001. ACM.
http://www.disi.unitn.it/~montreso/ds/papers/CAN.pdf

Example: 2-dimensional space

- Space divided between nodes
- All nodes cover the entire space
- Each node covers either a square or a rectangular area of ratios 1 : 2 or 2 : 1

Example:

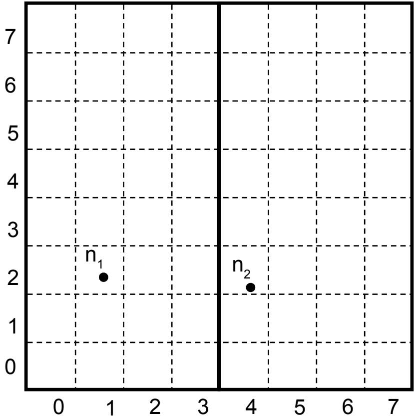
- Node $n_1 : (1, 2)$ – first node that joins – cover the entire space



Example: 2-dimensional space

Example:

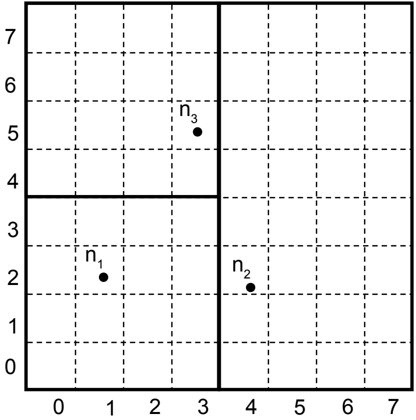
- Node $n_2 : (4, 2)$ joins: space is divided between n_1 and n_2



Example: 2-dimensional space

Example:

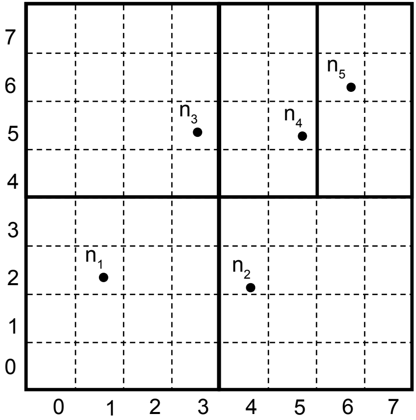
- Node $n_3 : (3, 5)$ joins



Example: 2-dimensional space

Example:

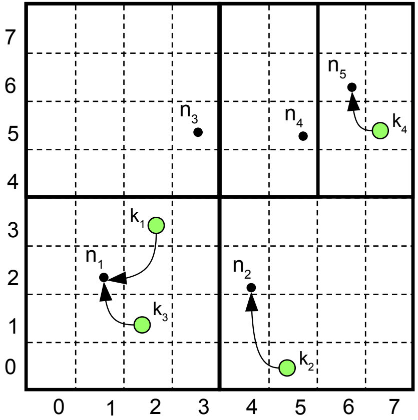
- Nodes $n_4 : (5, 5)$ and $n_5 : (6, 6)$ join



Example: 2-dimensional space

Example:

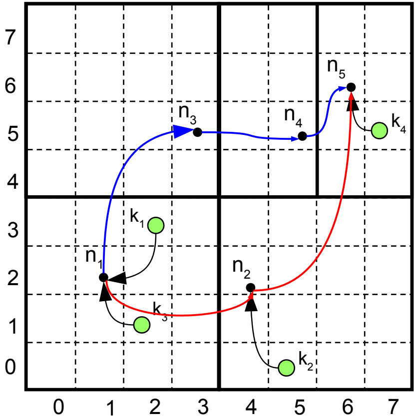
- Nodes: $n_1 : (1, 2)$, $n_2 : (4, 2)$, $n_3 : (3, 5)$, $n_4 : (5, 5)$, $n_5 : (6, 6)$
- Items: $k_1 : (2, 3)$, $k_2 : (5, 1)$, $k_3 : (2, 1)$, $k_4 : (7, 5)$
- Each item is stored by the node who owns its mapping in the space



Example: 2-dimensional space

Example:

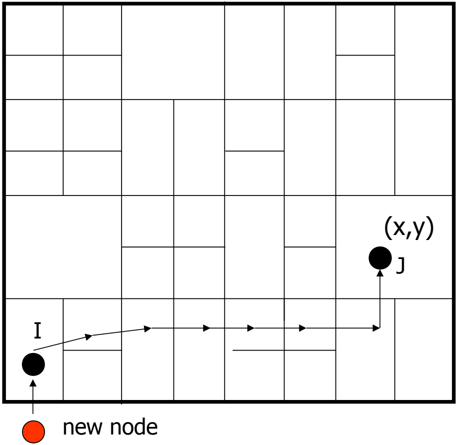
- Each node knows its neighbors in the d-space
- Forward query to the neighbor that is closest to the query id
- Example: assume n_1 queries k_4
- Can route around some failures



Example: 2-dimensional space

Node joining:

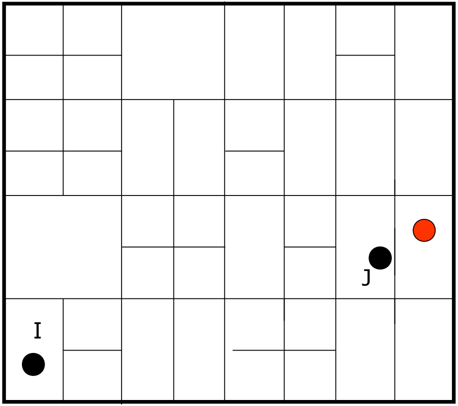
1. Discover some node I already in CAN
2. Pick random point (x, y) in space
3. I routes to (x, y), discovers node J



Example: 2-dimensional space

Node joining:

1. Split J zone in half
2. New node owns one half



Node departures

Take-over mechanism:

- Node explicitly hands over its zone and the associated (key,value) database to one of its neighbors
- A maximum of $2d$ nodes need to be contacted
- Problem: in case of network failure, no regeneration of data
- Solution: every node has a backup of its neighbors

Multi-verse?

Increasing availability:

- Each key is mapped into r different realities
- Each reality is associated with a different hash function
- A key is not available only when the r nodes hosting it in different realities are down at the same time

Kademlia

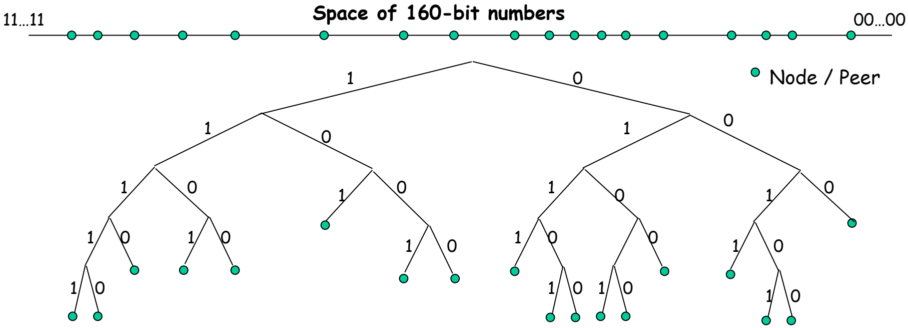
Key points

- Kademlia uses tree-based routing
- SHA-1 hash function in a 160-bit address space
- Every node maintains information about keys close to itself
  - Distance based on the XOR metric: $d(a,b) = a \oplus b$
- Uses parallel asynchronous queries to avoid timeout delays
- Routes are selected based on latency

Bibliography

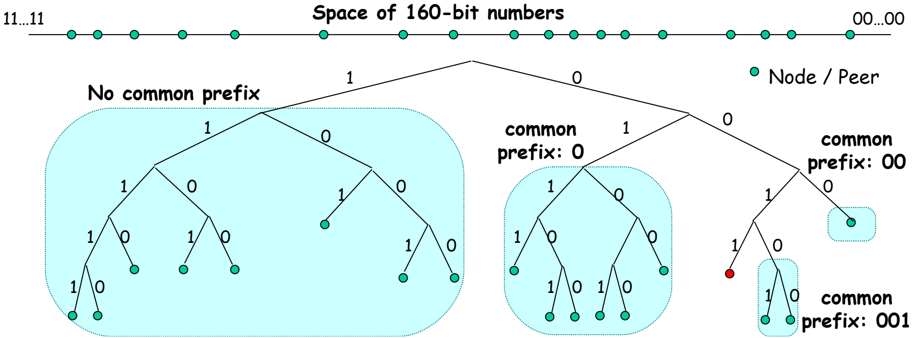
P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric.
In *Proc. of the* 1^{st} *International Workshop on Peer-to-Peer Systems (IPTPS'02)*, pages 258–263. Springer, 2002.
http://www.disi.unitn.it/~montreso/ds/papers/kademlia.pdf

Kademlia Tree



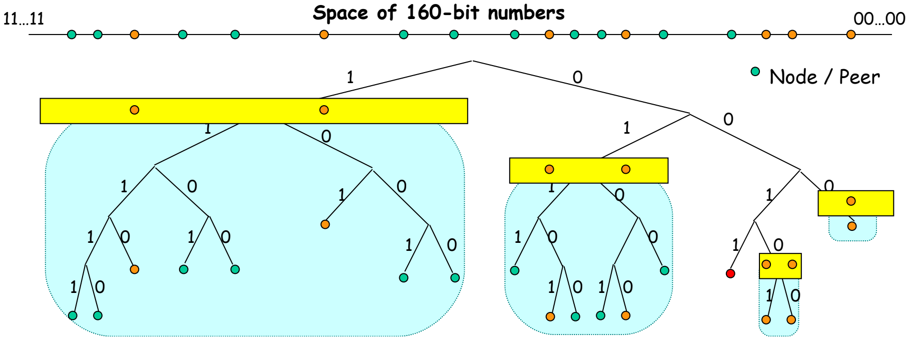
- Nodes are treated as leafs in binary tree
- Node's position in the tree is determined by the shortest unique prefix of its ID
- A node is responsible for all "closest" IDs (those having same prefix as itself)

Kademlia Tree



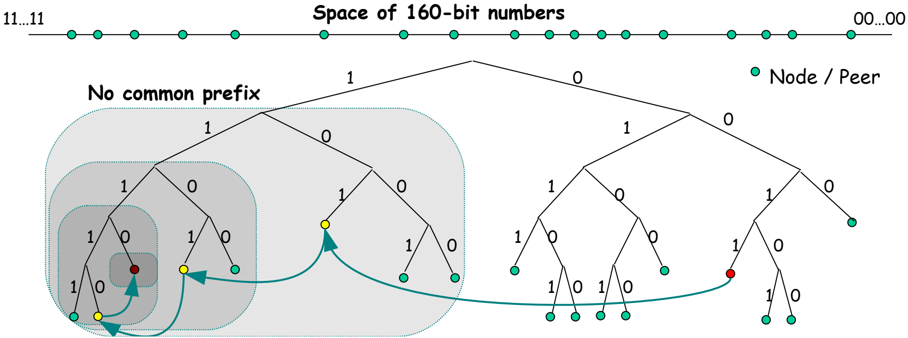
- From the point of view of each node, the tree is divided into a series of maximal subtrees that do not contain the node
- Example: the red node with prefix `0011`
- A node must know at least one node in each of these subtrees

Routing table



- Consider routing table for a node with prefix `0011`
- The routing table is composed of a series of *k*-buckets corresponding to each of the subtrees
- Consider a 2-bucket example, each bucket will have at least 2 contacts for each subtree

Kademlia Tree



- Consider a query for ID 111010… initiated by node 0011100…

Messages

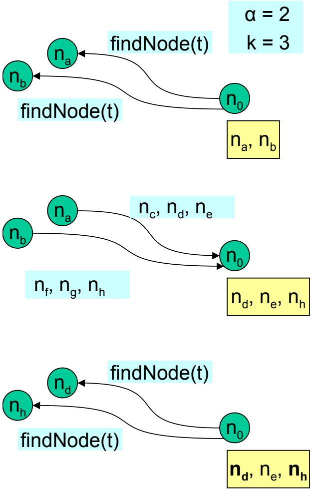
Kademlia protocol consists of 4 RPCs:

- $ping_{n \to m}()$
  - Probe node m to see if it is online
- $store_{n \to m}(k, v)$
  - Instruct node m to store a $\langle k, m \rangle$ pair
- $findNode_{n \to m}(t)$
  - Returns the k contacts "closest" to t
- $findValue_{n \to m}(k)$
  - Returns the value associated to k, if present, or
  - Returns k contacts closest to k

Routing

Goal: find k nodes closest to ID t – Protocol executed by n_0

- Initial phase :
  - insert in a set S all the nodes in the routing table
- Iteration
  - select a subset $T \subseteq S$ of the α nodes closest to t
  - invoke *findNode*(t) on nodes in T, in parallel
  - collect the replies in a new set S
  - repeat until no new node is discovered
- Final phase
  - invoke *findNode*(t) to all of k closest nodes not already queried
  - return when have results from all the k-closest nodes



Kademlia summary

Strengths:

- Low control message overhead
- Tolerance to node failure and leave
- Capable of selecting low-latency path for query routing
- Unlike Chord, Kademlia is symmetric: $a \oplus b = b \oplus a$
  - Peers receive lookup queries from precisely the same set of neighbors contained in their routing tables

Weaknesses:

- Balancing of storage load is not truly solved
- No experimental results provided

Cassandra

Few information available:

- $O(1)$ routing hops
- $O(N)$ routing state
  - Thanks to a routing protocol that guarantees that eventually every node knows every other node

Bibliography

D. Featherston. Cassandra: Principles and application.
http://www.disi.unitn.it/~montreso/ds/papers/Cassandra.pdf

Security aspects of DHTs

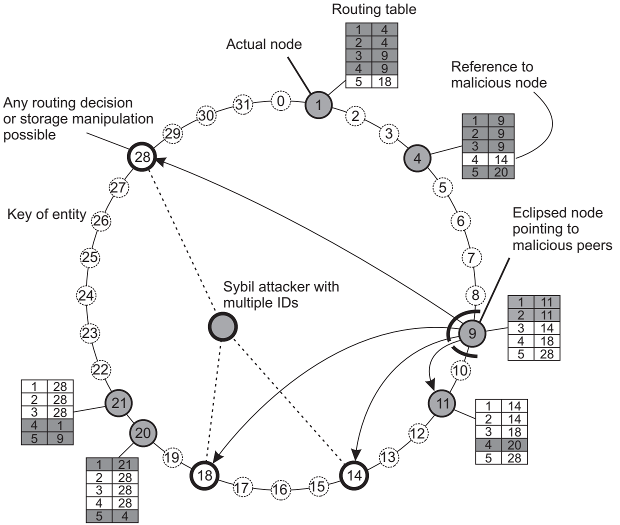
Security weaknesses specific to DHTs

- Sybil attacks
  - an attacker introduces a large number of bogus nodes that can subvert protocols based on redundancy
- Eclipse attacks
  - an attacker tries to corrupt the routing tables of honest nodes by filling them with references to malicious nodes
- Routing and storage attacks
  - various attacks where malicious nodes do not follow the routing and storage protocols correctly

Bibliography

G. Urdaneta, G. Pierre, and M. van Steen. A survey of DHT security techniques. *ACM Computing Surveys*, 43(2), Jan. 2011.
http://www.disi.unitn.it/~montreso/ds/papers/DhtSecuritySurvey.pdf

Example of attacks



Defenses against Sybil attacks

- Collusion is easier
- Possible defenses:
  - Centralized certification
  - Distributed registration
  - Physical network characteristics
  - Social networks
  - Computational puzzles
- You can only reduce the impact of Sybil attacks, not eliminate them completely

Defenses against eclipse attacks

- Effect of eclipse attack ("table poisoning") is measured by:

$$\frac{\text{percentage of malicious entries in routing tables}}{\text{percentage of malicious users in the network}}$$

- Possible defenses:
  - Constrained neighbor selection
    - Original Chord: only one node may fit in a finger table entry – good
    - Random Chord: several nodes may fit in finger table entry – bad
    - Pastry: some table entries may be filled by any node sharing a short prefix – bad
    - Kademlia: table entries are filled by fast-responding peers – good
  - In-degree anonymous auditing
    - Malicious nodes have larger in-degree

Defenses against routing and storage attacks

- Redundant routing
  - Possible approaches:
    - Multiple paths
    - Wide paths
    - Multiple wide paths
  - Wide paths require one good node per hop, multiple paths require a path with only good nodes
- Redundant storage
  - Storing replicas "numerically close" to each other
    - Chord, Pastry, Kademlia
    - Pros: easier to maintain consistency
    - Cons: malicious node may control a region of space
  - Storing replicas spread over the identifier space
    - Tapestry, several other proposals
    - Pros: most difficult to subvert an area
    - Cons: requires additional tables

Why Kademlia?

Generic reasons

- Relative security: wide searches
- Replicated storage

The reality is that Kademlia is insecure

- Successful (academic) attacks on Kad/BitTorrent
- Successful infiltrations on the Storm BotNet

The real reasons

- For BitTorrent, damage is limited anyway (decentralized tracking)
- Many alternative ways to obtain peers (PEX, multiple trackers)

Comparison

	CAN	**Chord**	**Tapestry**	**Pastry**
Architecture	d-dimens. space	ring	Plaxton tree	Plaxton tree
Routing hops	$O(dN^{1/d})$	$O(\log N)$	$O(\log_b N)$	$O(\log_b N)$
Routing state	$2d$	$\log N$	$\log_b N$	$B \log_b N$
Join cost	$2d$	$(\log N)^2$	$\log_b N$	$\log_b N$

	Kademlia	**Viceroy**	**Koorde**	**Kelips**
Architecture	Tree	Butterfly network	de Brujin graph	n-dimens. space
Routing hops	$O(\log N)$	$O(\log n)$	$O\left(\frac{\log n}{\log \log n}\right)$	$O(1)$
Routing state	$k \log N$	$\log N$	$\log N$	$\sqrt{n}$
Join cost	$k \log N$	$\log N$	$\log N$	$\sqrt{n}$

Conclusions

- The DHT abstraction is doing well, both inside clouds and in P2P networks
- Kademlia seems to be the winner. Main reasons:
  - Performance
  - Relative security

Table of contents

Gnutella: brief history

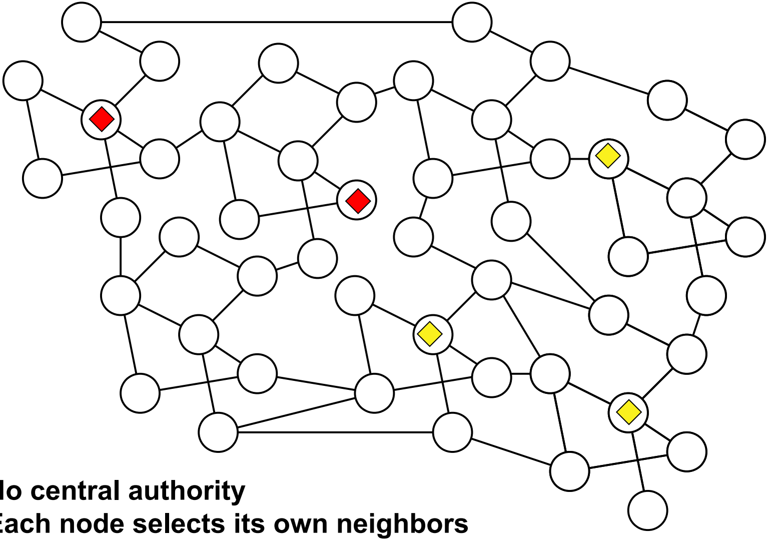
- Nullsoft (a subsidiary of AOL) released Gnutella on March 14th, 2000, announcing it on Slashdot
- AOL removed Gnutella from Nullsoft servers on March 15th, 2000
- After a few days, the Gnutella protocol was reverse-engineered
- Napster was shutdown in early 2001, spurring the popularity of Gnutella
- On October 2010, LimeWire (a popular client) was shutdown by court's order

Gnutella

Gnutella is a protocol for peer-to-peer search, consisting of:

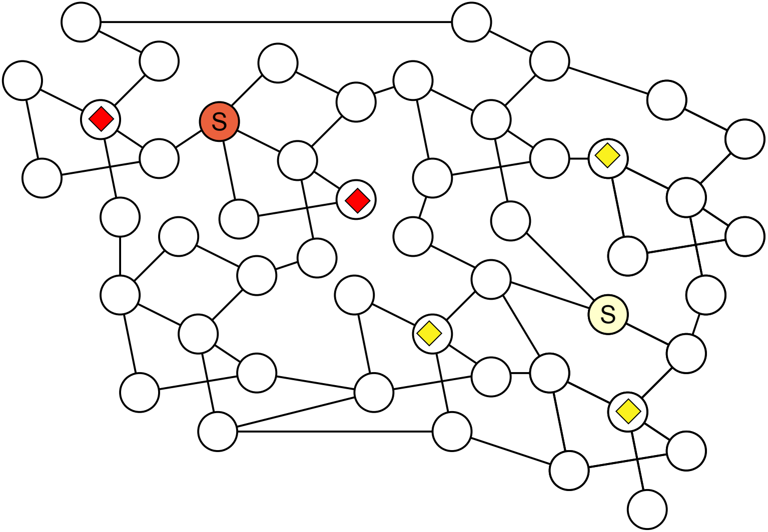
- A set of message formats
  - 5 basic message types
- A set of rules governing the exchange of messages
  - Broadcast
  - Back-propagate
  - Handshaking
- An hostcache for node bootstrap

Gnutella topology: unstructured

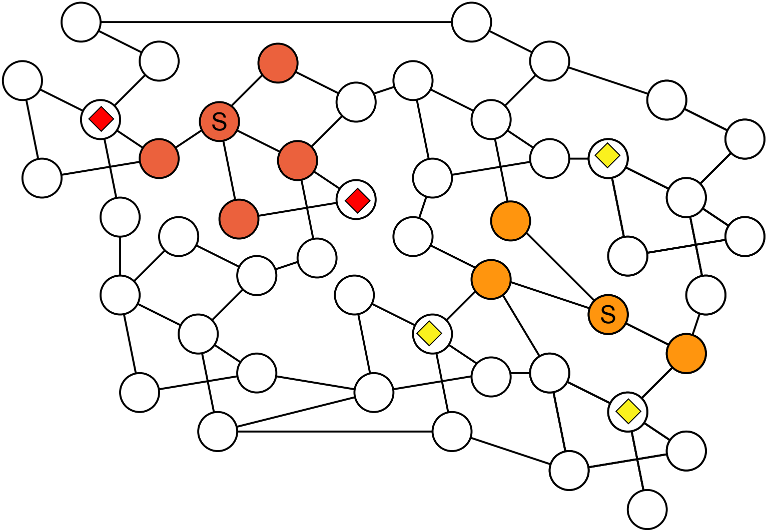


No central authority
Each node selects its own neighbors

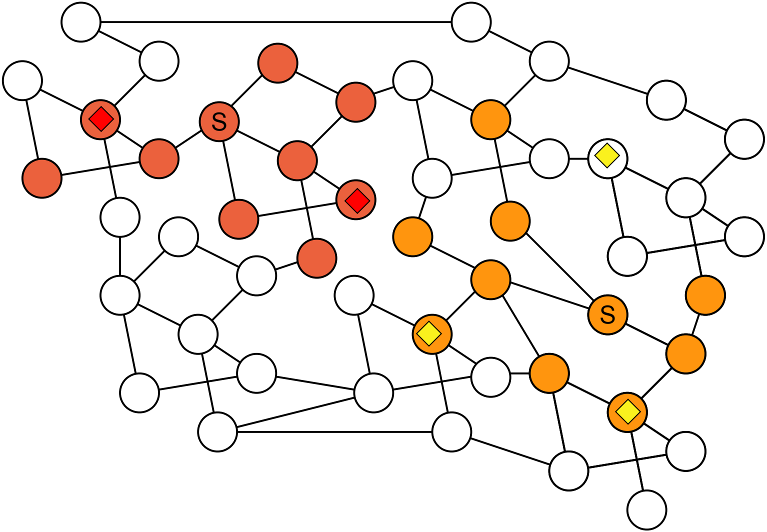
Gnutella routing



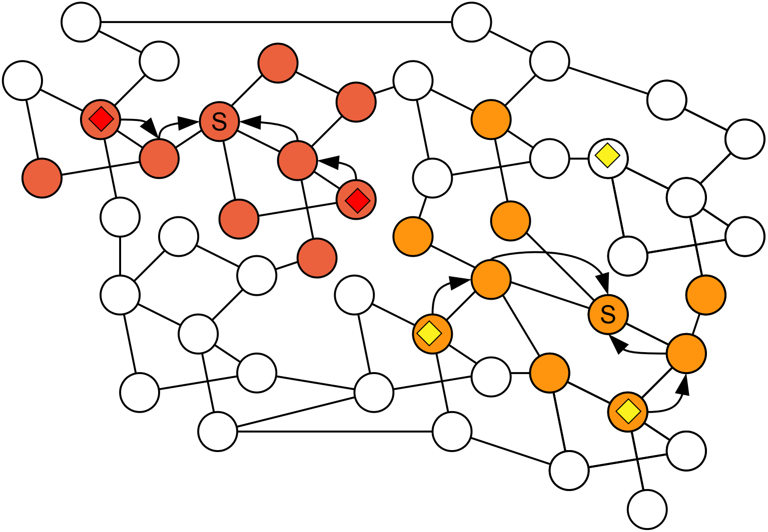
Gnutella routing



Gnutella routing



Gnutella routing



Gnutella messages

Each message is composed of:

- A 16-byte ID field uniquely identifying the message
  - randomly generated
  - not related to the address of the requester (anonymity)
  - used to detect duplicates and route back-propagate messages
- A message type field
  - PING, PONG
  - QUERY, QUERYHIT
  - PUSH (for firewalls)
- A Time-To-Live (TTL) Field
- Payload length

Gnutella messages

- PING (broadcast)
  - Used to maintain information about the nodes currently in the network
  - Originally, a "who's there" flooding message
  - A peer receiving a PING is expected to respond with a PONG message
- PONG (back-propagate)
  - A PING message has the same ID of the corresponding PING message
  - Contains:
    - address of connected Gnutella peer
    - total size and total number of files shared by this peer

Gnutella messages

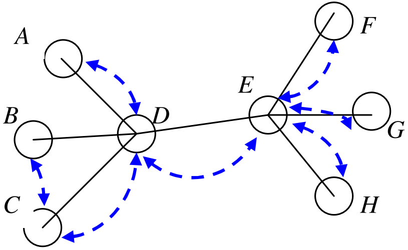
- QUERY (broadcast)
  - The primary mechanism for searching the distributed network
  - Contains the query string
  - A servent is expected to respond with a QUERYHIT message if a match is found against its local data set
- QUERYHIT (back-propagate)
  - The response to a query
  - Has the same ID of the corresponding QUERY message
  - Contains enough information to acquire the data matching the corresponding query
    - IP Address + port number
    - List of file names

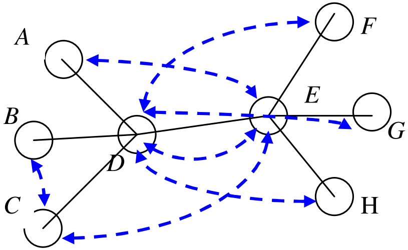
Beyond the original Gnutella

Several problems in Gnutella 0.4 (the original one):

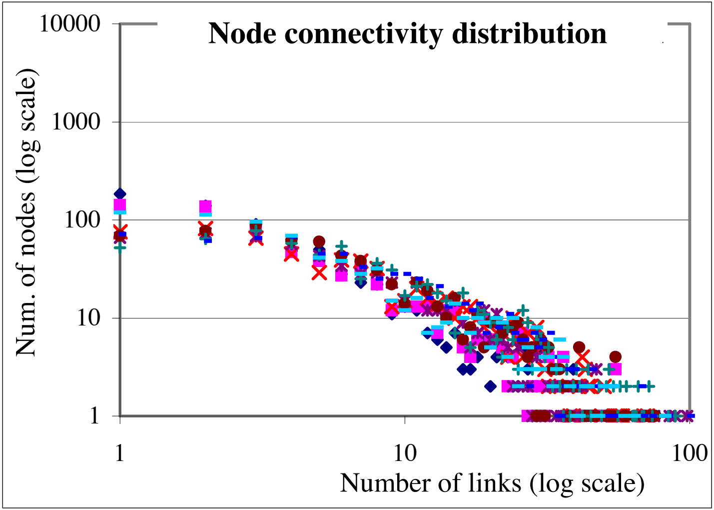
- What kind of topology is generated?
  - Is it planned ("engineered")?
  - Is it good?
- PING-PONG traffic
  - More than 50% of the traffic generated by Gnutella 0.4 is PING-PONG related
- Scalability
  - Each query generates a huge amount of traffic
    - e.g. $TTL = 6, d = 10 \Rightarrow 10^6$ messages
  - Potentially, each query is received multiple times from all neighbors

Gnutella overlay vs underlying topology

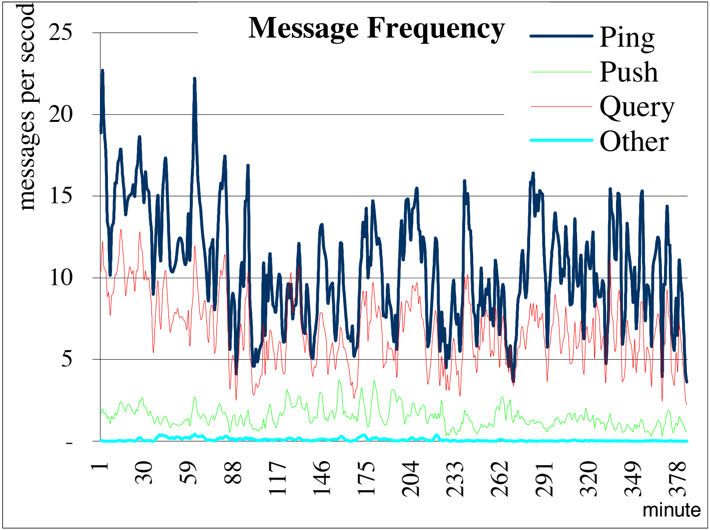




Traffic


Node connectivity distribution
Num. of nodes (log scale)
10000
1000
100
10
1
1
10
100
Number of links (log scale)

Connectivity (and robustness)



Gnutella conclusions

Gnutella 0.6:

- Superpeer-based organization
- Ping/pong caching
- Query routing

Summary:

- A milestone in P2P computing
  - Gnutella proved that full decentralization is possible
- But:
  - Gnutella is a patchwork of hacks
  - The ping-pong mechanism, even with caching, is just plain inefficient

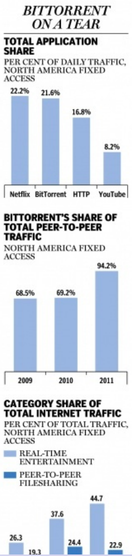
BitTorrent

- Interest on P2P system driven by file sharing applications
  - end users become content provider
- Main focus is to efficiently discover content
  - different generations of P2P…
    - centralized (Napster), unstructured (Gnutella), structured (DHT)
  - …with different problems
    - single point of failure (centralized), low success rate (unstructured), high management traffic (structured)

But…what happens when you find the content?

BitTorrent

- Designed for efficient content download
- Search features not included
- Large portion of the Internet traffic is due to BitTorrent
- Basic concept: file swarming



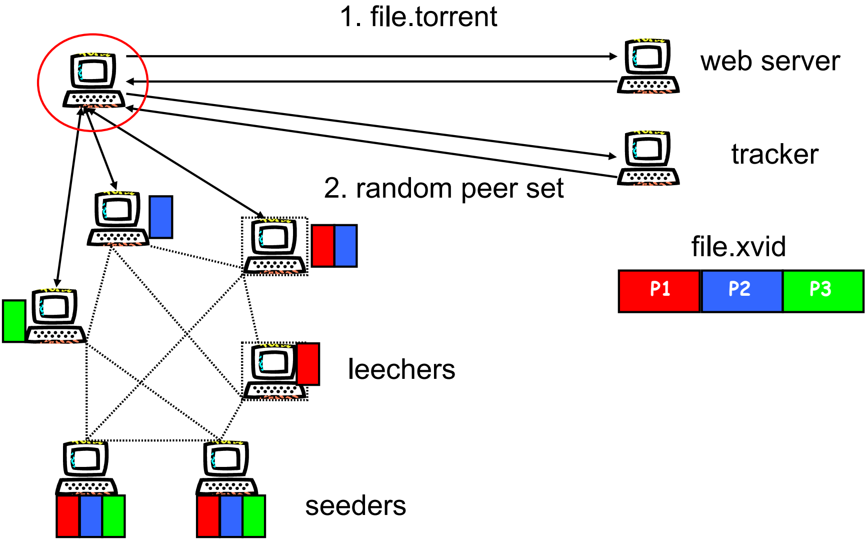
Bibliography

B. Cohen. Incentives build robustness in BitTorrent. In *Proc. of the Workshop on Economics of P2P Systems*, 2003.
http://www.disi.unitn.it/~montreso/ds/papers/BitTorrent.pdf

Legal (!) applications

- Music, video and the like
  - BitTorrent Inc
  - SubPop Records
  - Norwegian Broadcasting Corporation
- Software
  - Linux distributions
  - Blizzard: Diablo III, StarCraft II, World of Warcraft (game updates)
- Web services
  - Amazon S3 equipped with built-in BitTorrent support
  - Facebook, Twitter use BitTorrent to distribute updates to their servers

BitTorrent architecture



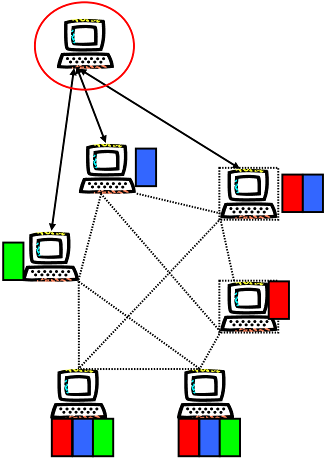
Torrent file

A torrent file is a *bencoded* dictionary with the following keys:

- `announce` – the URL of the tracker
- `name` – suggested file/directory name
- `piece length` – number of bytes per piece (commonly 256KB)
- `pieces` – a concatenation of each piece's SHA-1 hash.
- Exactly one of `length` or `files`:
    - `length` – size of the file (in bytes)
    - `files` – a list of files with the following keys:
        - `path` - pathname of the file
        - `length` - size of the file (in bytes)

BitTorrent architecture

- Peer Selection
  - "Which peers to upload to"
  - Efficiency criteria:
    - Maximize service capacity
    - Foster reciprocation and prevent free riders
- Piece selection
  - "Which pieces to download from selected peer"
  - Should guarantee *piece diversity*
    - Always find an interesting piece in selected peer
    - Do not bias peer selection



Piece selection

- The order in which pieces are selected by peers is critical
- A bad algorithm could create a situation where all peers have all pieces that are currently available and none of the missing ones
- If the original seed disappears, the download cannot be completed!

Policies

- Strict Priority
  - A piece is broken into sub-pieces (typically 16KB in size)
  - Policy: *Until a piece is assembled, only download sub-pieces for that piece from the same source*
  - This policy lets complete pieces assemble quickly
- Rarest first
  - Policy: *Determine the pieces that are most rare among your peers and download those first*
  - This ensures that the most common pieces are left till the end to download
  - Rarest first also ensures that a large variety of pieces are downloaded from the seed

Policies

- Random first piece
  - Initially, a peer has nothing to trade
  - Important to get a complete piece ASAP
  - Rare pieces are typically available at fewer peers, so downloading a rare piece initially is not a good idea
  - Policy: *Select a random piece of the file and download it*
- Endgame mode
  - Policy: *When all the sub-pieces that a peer doesn't have are actively being requested, these are requested from* ***every*** *peer*
  - When the sub-piece arrives, the replicated requests are canceled
  - This ensures that a download doesn't get prevented from completion due to a single peer with a slow transfer rate
  - Some bandwidth is wasted; in practice, not too much

Peer selection

Choking

- Choking is a temporary refusal to upload; download occurs as normal
- One of BitTorrent's most powerful idea
- It ensures that nodes cooperate and eliminates(?) the free-ride problem
- When a node is unchoked, upload restart
- Connection is kept open to reduce setup costs
- Based on game-theoretic tit-for-tat strategy in repeated games

Prisoner's Dilemma

Two men are arrested, but the police do not possess enough information for a conviction. Following the separation of the two men, the police offer both a similar deal:

	Prisoner B stays silent	Prisoner B confesses
Prisoner A stays silent	Both serve 1 months	A serves 1 year B goes free
Prisoner A confesses	B serves 1 year A goes free	Both serve 3 months

Prisoner's Dilemma

Single-iteration game

- What is the best strategy?

Prisoner's Dilemma

Single-iteration game

- What is the best strategy?
- "Confessing" is a dominant strategy
  - If the other prisoner confesses, the best move is to confess
  - If the other prisoner stay silent, the best move is to confess

Prisoner's Dilemma

Single-iteration game

- What is the best strategy?
- "Confessing" is a dominant strategy
  - If the other prisoner confesses, the best move is to confess
  - If the other prisoner stay silent, the best move is to confess

What about iterated games?

Prisoner's Dilemma

Single-iteration game

- What is the best strategy?
- "Confessing" is a dominant strategy
  - If the other prisoner confesses, the best move is to confess
  - If the other prisoner stay silent, the best move is to confess

What about iterated games?

- Robert Axelrod's "The evolution of cooperation"
- Tournament of computer programs playing PD
- The winner: Tit-for-tat, Anatol Rapoport

Prisoner's Dilemma

Tit-for-tat

- Be nice at the beginning
- Do onto others as they do onto you:
- If the other prisoner confesses, you must retaliate back
- Have a recovery mechanism to ensure eventual cooperation

How to translate this in BitTorrent?

Choking/unchoking

Goal: have several bidirectional connections running continuously

- Upload to peers who have uploaded to you recently
  - "Do onto others as they do onto you"
- Unused connections are uploaded to on a trial basis to see if better transfer rates could be found using them
  - "Be nice at the beginning"
  - "Have a recovery mechanism to ensure eventual cooperation"

Choking/unchoking specifics

- A peer always unchokes a fixed number of its peers (default: 4)
- Decision to choke/unchoke done based on current download rates, averaged over the last 20s
- Evaluation on who to choke/unchoke is performed every 10s
  - Prevents wasting of resources by rapidly choking/unchoking peers
  - Enough for TCP to ramp up transfers to their full capacity
- Which peer is the optimistic unchoke is rotated every 30s
  - Used to discover if a currently choked peer would be better

Additional details

Anti-snubbing:

- A peer is said to be snubbed if each of its peers chokes it
- To handle this, snubbed peer stops uploading to its peers
- Optimistic unchoking done more often
  - Hope is that will discover a new peer that will upload to us

Seeding:

- Once download is complete, a peer has no download rates to use for comparison nor has any need to use them
- The question is, which nodes to upload to?
- Policy: Upload to those with the best upload rate.
  - This ensures that pieces get replicated faster

Improvements over the tracker bottleneck

- Trackerless BitTorrent (i.e., w/o a centralized tracker):
  - Based on variants of Kademlia DHT
  - Tracker run by a normal end-host
  - Vuze DHT vs Mainline DHT
- Peer Exchange (PEX):
  - Each peer directly update other peers as to which peers are currently in the swarm
  - Epidemic sampling!
  - Three incompatible version of PEX (Vuze, BitComet, Mainline)
- Multitracking
  - Multiple trackers in the torrent file

Five months in a torrent's lifetime

- Analysis of a tracker log
- 1.77GB Linux Redhat 9 distribution
- Five months - April-August 2003
- 180.000 downloads

Bibliography

M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. Al Hamra, and L. Garcés-Erice.

Dissecting bittorrent: Five months in a torrent's lifetime.
In *Passive and Active Network Measurement*, volume 3015 of *Lecture Notes in Computer Science*, pages 1–11. Springer.

http://www.disi.unitn.it/~montreso/ds/papers/FiveMonths.pdf

Network: Number of active peers over time

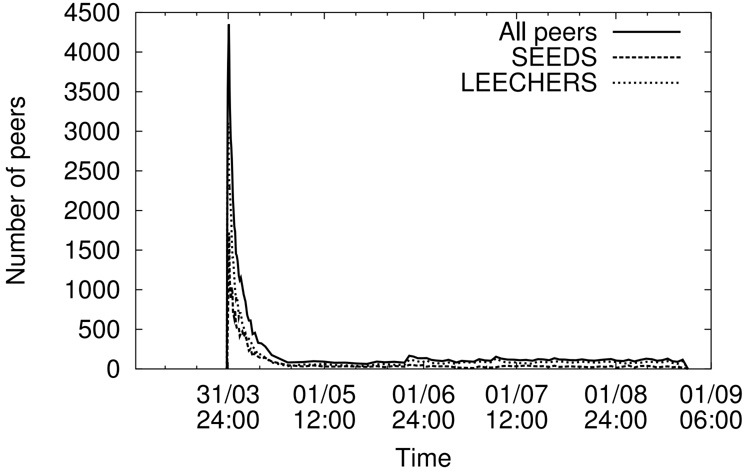


Figure: Complete trace

Network: Number of active peers over time

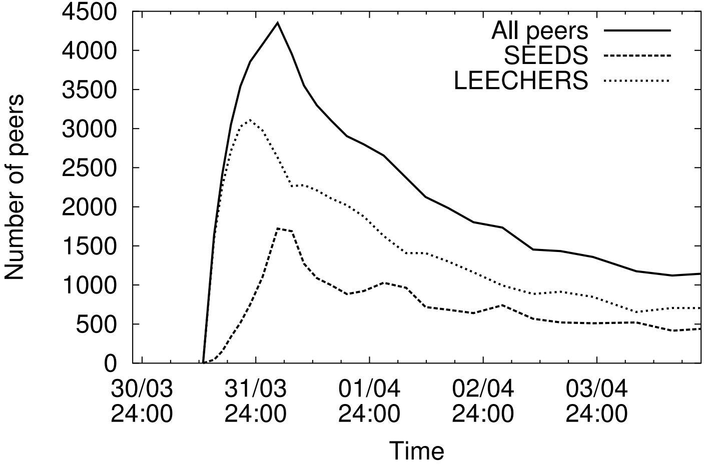


Figure: First five days

Network: Proportion of seeders and leechers

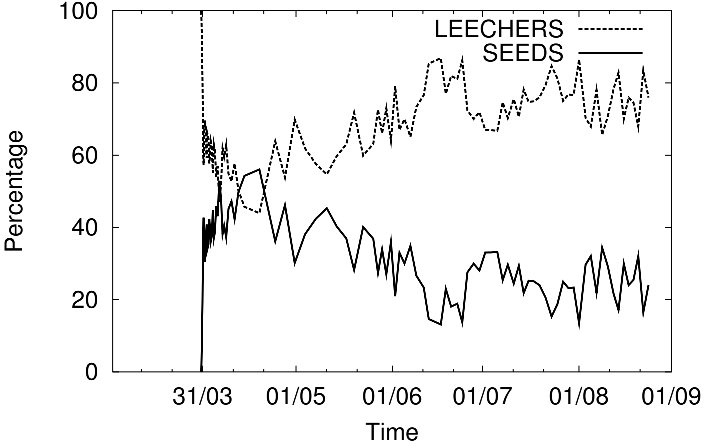


Figure: Complete trace

Client: Cumulative download and upload evolution

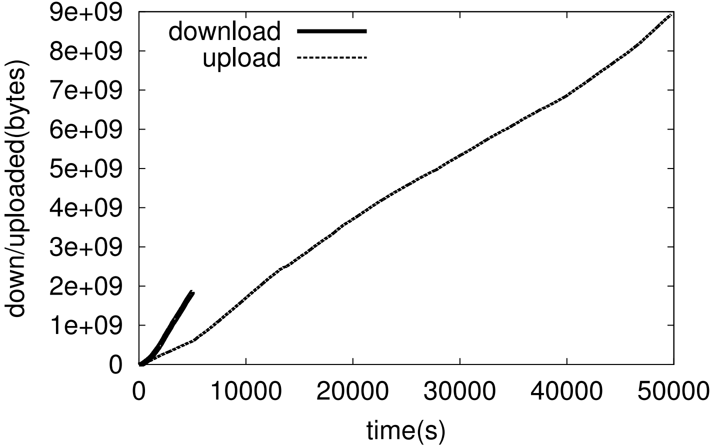


Figure: Complete torrent

Client: Cumulative download and upload evolution

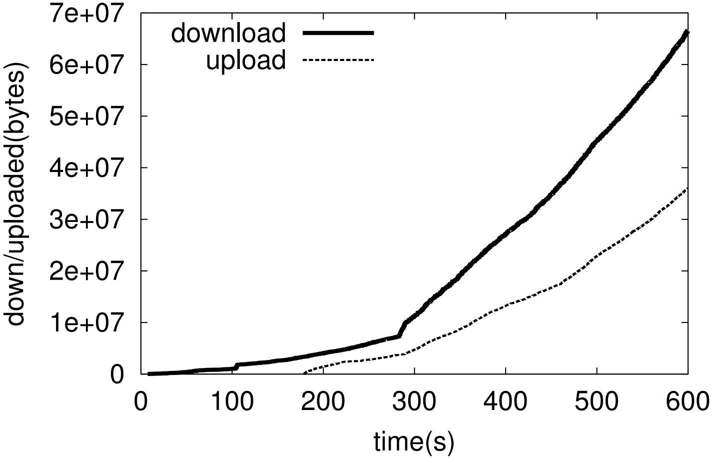


Figure: First ten minutes

Client: Number of connected peers

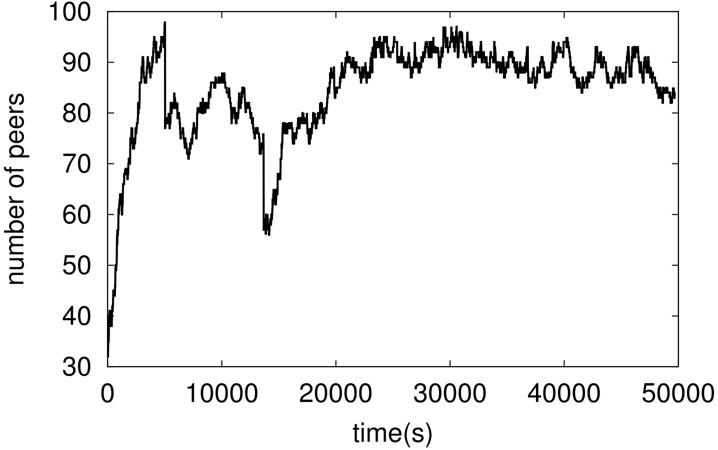


Figure: Around 14 hours

Cheating BitTorrent

- Tit-for-tat strategy has been designed to foster reciprocation
- Nevertheless, its incentives are not robust to strategic clients
- Two examples:
  - BitTyrant
    - a strategic client that tries to improve download/upload rate
  - BitThief
    - a client that never uploads anything

Bibliography

- M. Piatek, T. Isdal, T. E. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent?
  In *Proc. of NSDI'07*, Cambridge, Massachusetts, USA, Apr. 2007. USENIX.
  http://www.disi.unitn.it/~montreso/ds/papers/BitTyrant.pdf
- T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free riding in BitTorrent is cheap.
  In *Proc. of HotNets-V*, Irvine, USA, Nov. 2006. Usenix.
  http://www.disi.unitn.it/~montreso/ds/papers/BitThief.pdf

BitTyrant

How to improve performance?

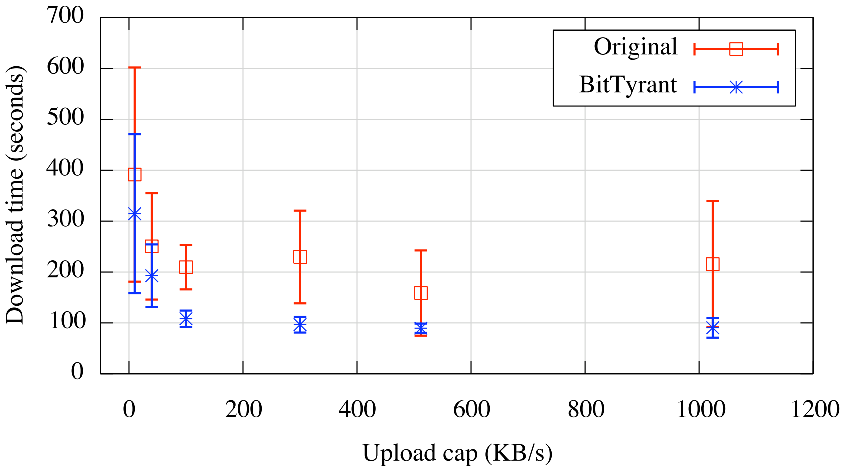
- Maximize reciprocation bandwidth per connection
- Maximize number of reciprocating peers
- Deviate from equal split

Unchoking algorithm

- d_p: download rate of connection p
- u_p: upload rate of connection p
- Each round, rank peers by the ratio u_p/d_p and unchoke the first k such that the upload capacity is reached:

$$\sum_{i=1}^{k} u_i \leq cap$$

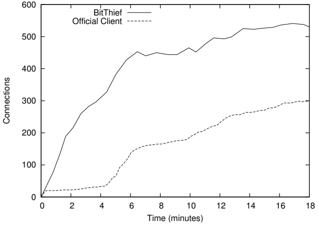
BitTyrant


Original
BitTyrant
Download time (seconds)
Upload cap (KB/s)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

BitThief

Download only: benefits

- no copyright issues (only contributors are sued)
- conserve resources
- spoil the community



Gains from optimistic unchoking:

- Ask for as many clients as possible
  - Increment tracker polling
  - Decentralized tracking, PEX
- Connect to all available clients
  - higher chance of being unchoked
- Always pretend to be a newcomer
  - Advertise no pieces
  - Download whatever available
  - Most clients are nice

Gains from free sharing of seeders:

- Seeders select peers in two ways:
  - highest bandwidth
  - round robin
- BitThief report high upload rate

BitThief

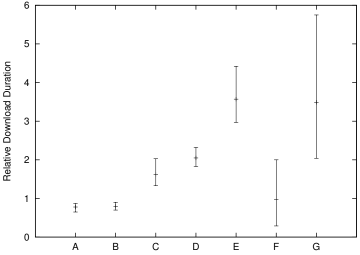


Figure: With seeders

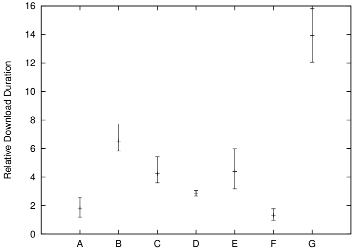


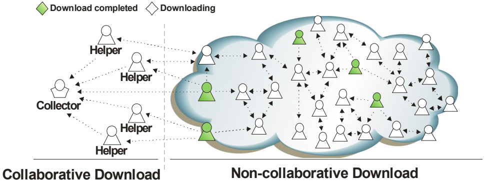
Figure: Without seeders

	Size	Seeders	Leechers	μ	σ
A	170MB	10518 (303)	7301 (98)	13	4
B	175MB	923 (96)	257 (65)	14	8
C	175MB	709 (234)	283 (42)	19	8
D	349MB	465 (156)	189 (137)	25	6
E	551MB	880 (121)	884 (353)	47	17
F	31MB	N/A (29)	N/A (152)	52	13
G	798MB	195 (145)	432 (311)	88	5

Tribler

Problem:

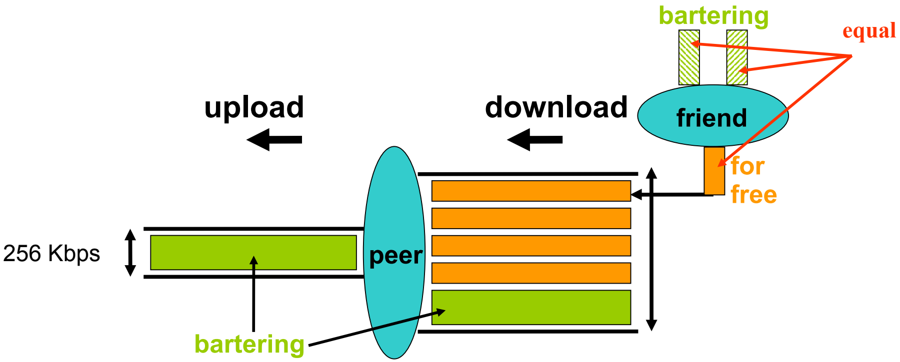
- Most users have different upload/download speeds
- Tit-for-tat may restrict the download speed
- Solution: let your friends help you for free



Bibliography

J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. Van Steen, and H. Sips. TRIBLER: a social-based peer-to-peer system. *Concurrency and Computation: Practice and Experience*, 20(2):127–138, 2008.
http://www.disi.unitn.it/~montreso/ds/papers/Tribler.pdf

Tribler


bartering
equal
upload
download
friend
for
free
256 Kbps
peer
bartering

Reading Material

- E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes.
  IEEE Communications Surveys and Tutorials, 7(2):72–93, 2005.
  http://www.disi.unitn.it/~montreso/ds/papers/P2PSurvey.pdf