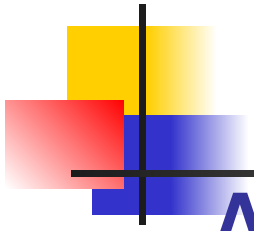


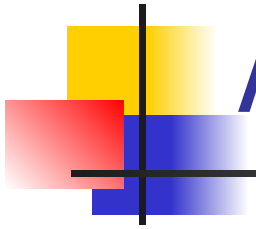
Architectural Models for Distributed Computing

CS237 Spring 2021
TuTh 5:00-6:20p.m.

Prof. Nalini Venkatasubramanian
nalini@uci.edu

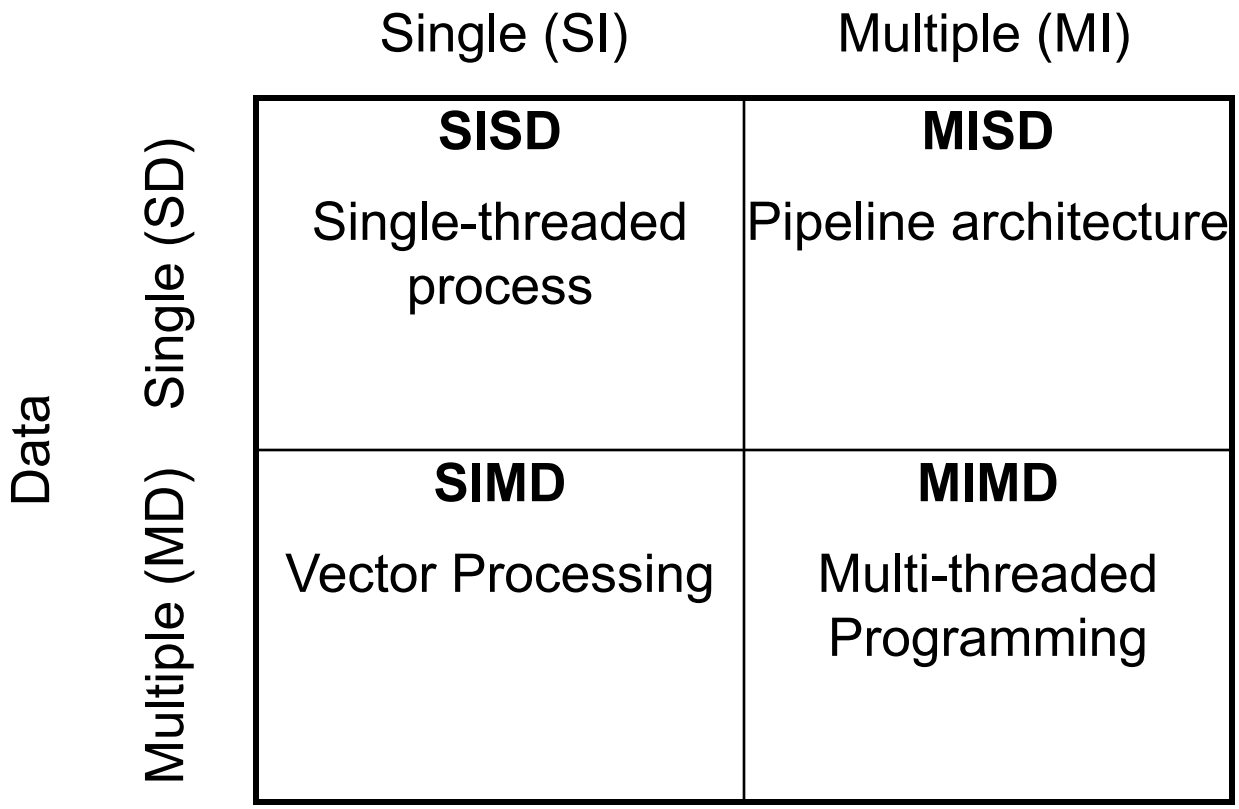
Acknowledgements: Slides modified from Kurose/Ross book slides.
Sukumar Ghosh, U at IOWA,
Mark Jelasity, Tutorial at SASO'07
Keith Ross, Tutorial at INFOCOM
Anwitaman Datta, Tutorial, ICDCN

Distributed Computing Architectures



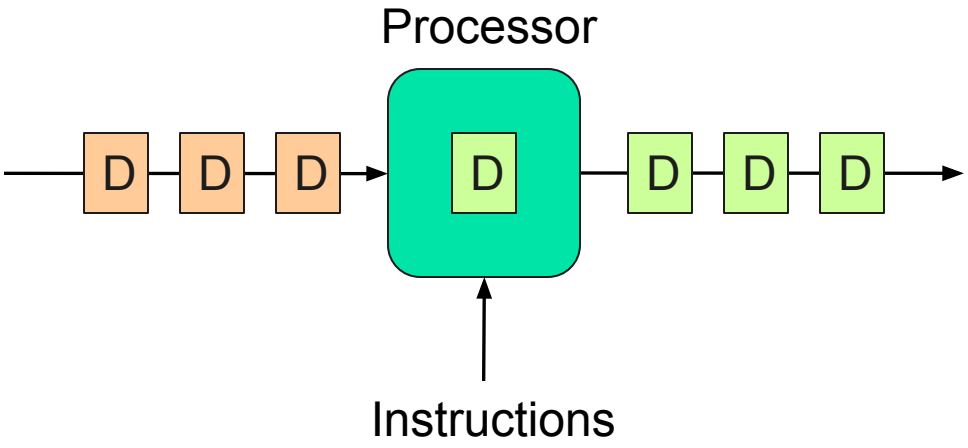
- Parallel Computing Architectures
  - SISD, SIMD, MISD, MIMD
  - Symmetric vs. Asymmetric Multiprocessing
- Client-Server Architectures
  - Client-Proxy-Server Architectures
- Peer-to-Peer Systems
- Cluster based Systems
- Edge/Cloud Architectures

Flynn's Taxonomy for Parallel Computing



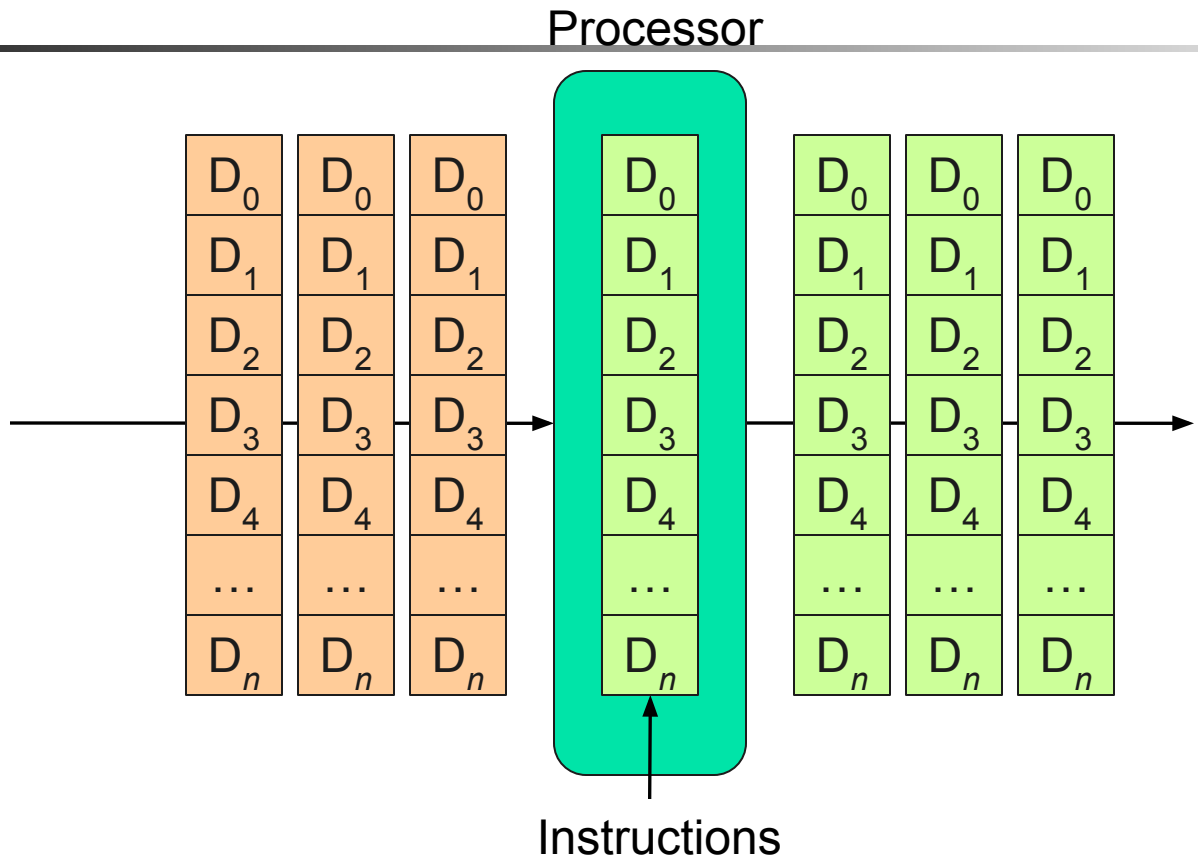
Parallelism – A Practical Realization of Concurrency

SISD (Single Instruction Single Data Stream)



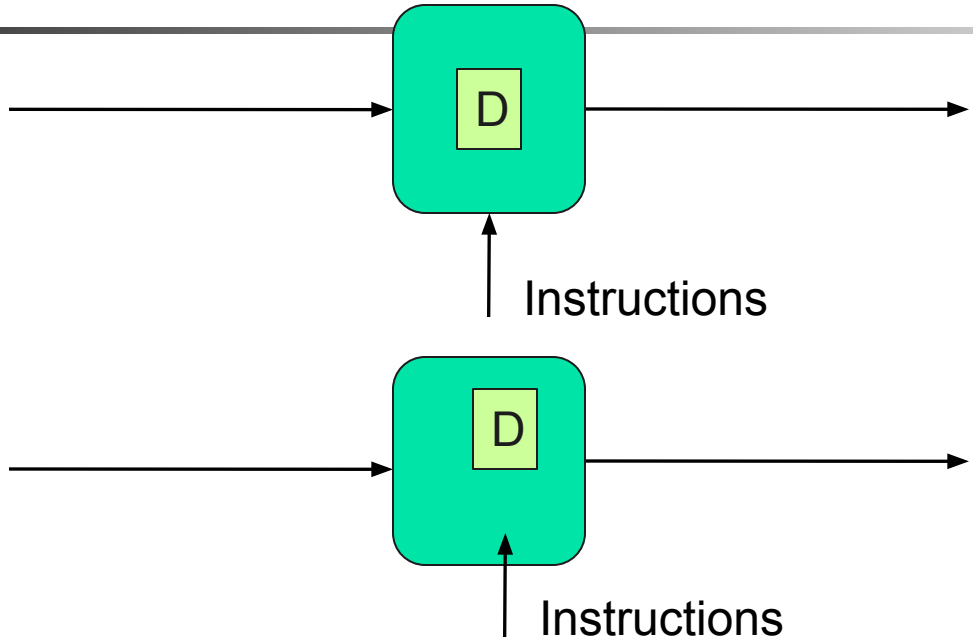
A sequential computer which exploits no parallelism in either the instruction or data streams.
Examples of SISD architecture are the traditional uniprocessor machines (currently manufactured PCs have multiple processors) or old mainframes.

SIMD



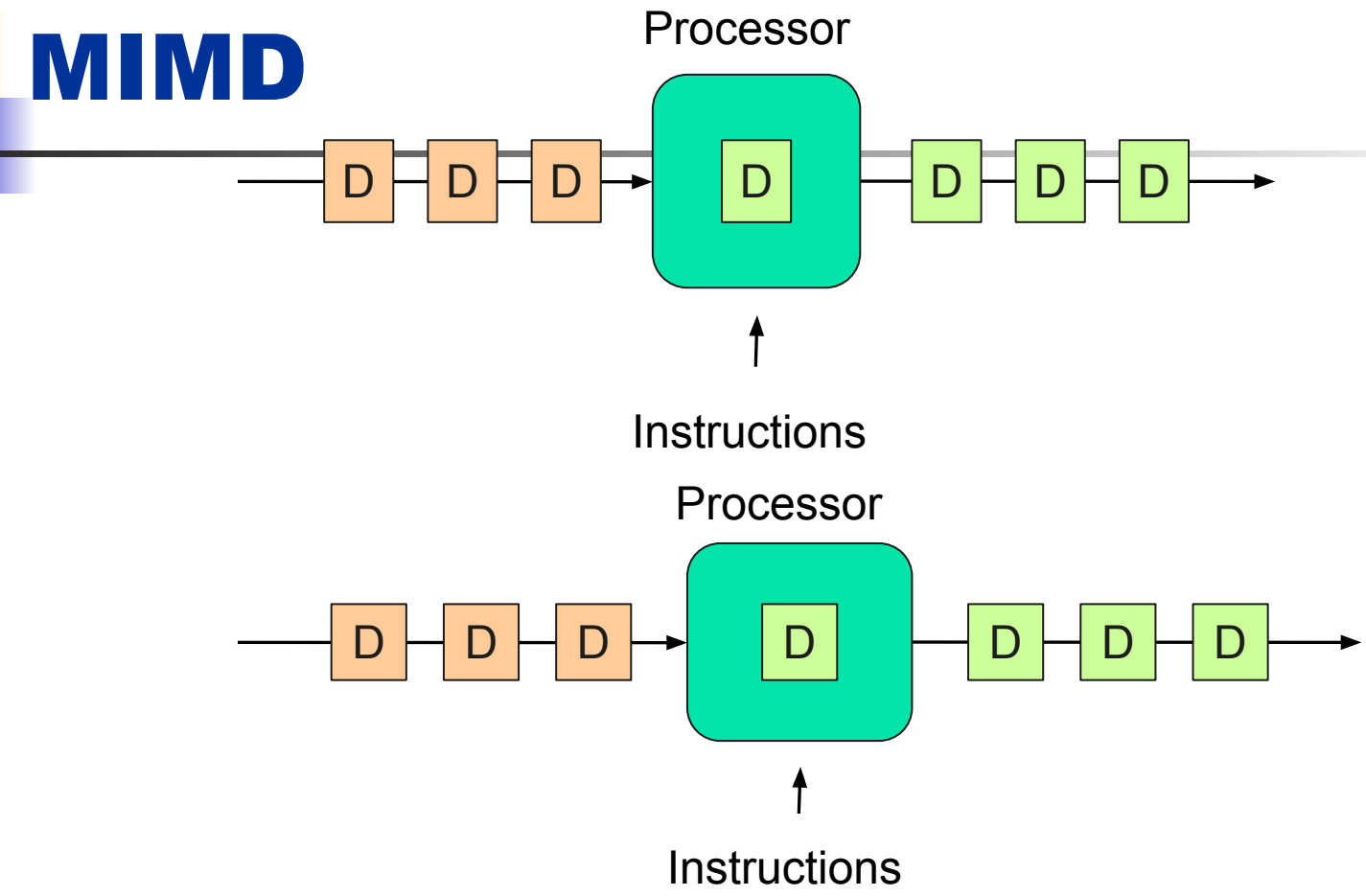
A computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelized.
For example, an array processorFor example, an array processor or GPU.

MISD (Multiple Instruction Single Data)



Multiple instructions operate on a single data stream.
Uncommon architecture which is generally used for fault tolerance.
Heterogeneous systems operate on the same data stream and aim to agree on the result.
Examples include the Space Shuttle flight control computer.

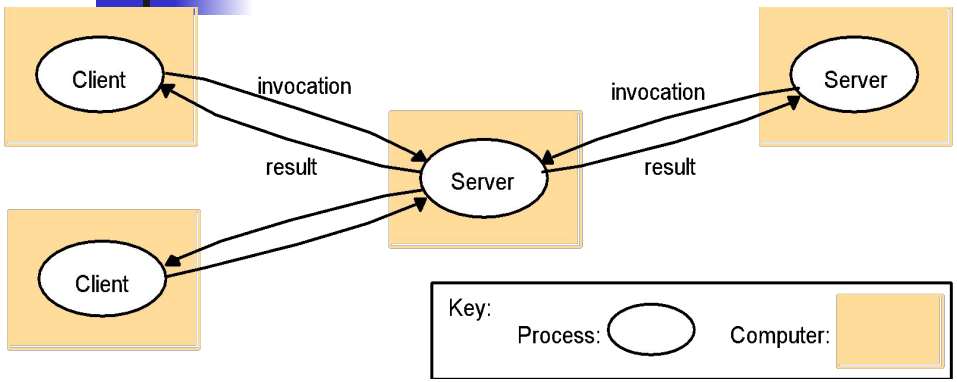
MIMD



Multiple autonomous processors simultaneously executing different instructions on different data.

Distributed systems are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space.

Classic Client/Server System

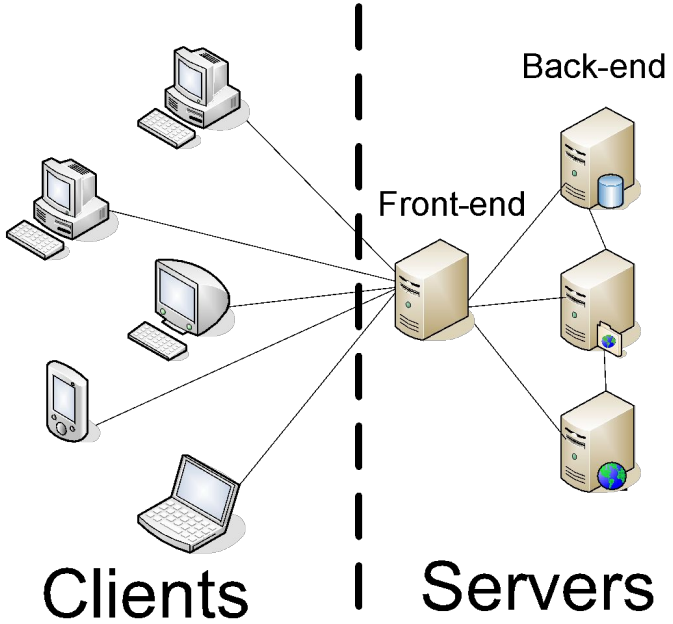


Every entity has its dedicated different role (Client/Server)

- request-response paradigm

Uses well-known reliable servers, popular

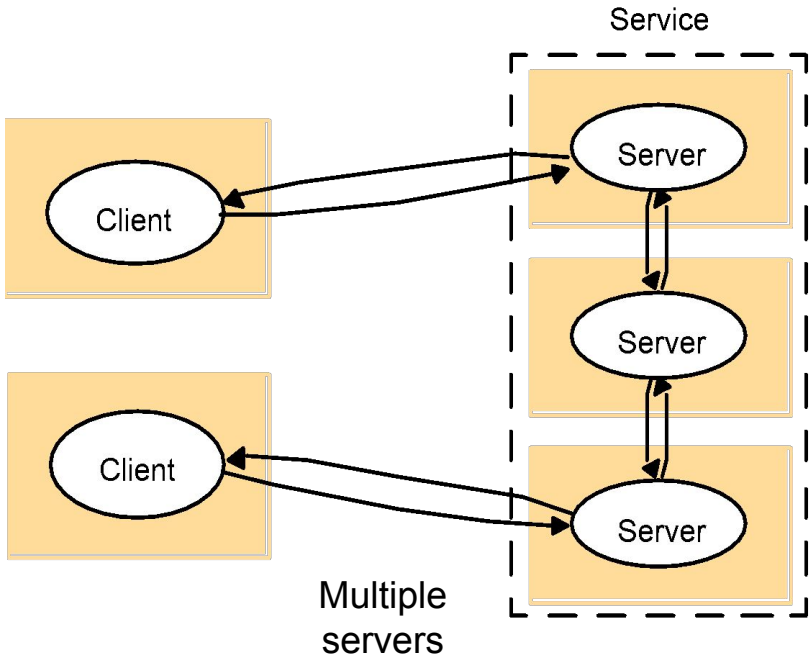
- Web Server
- FTP Server
- Media Server
- Database Server
- Application Server

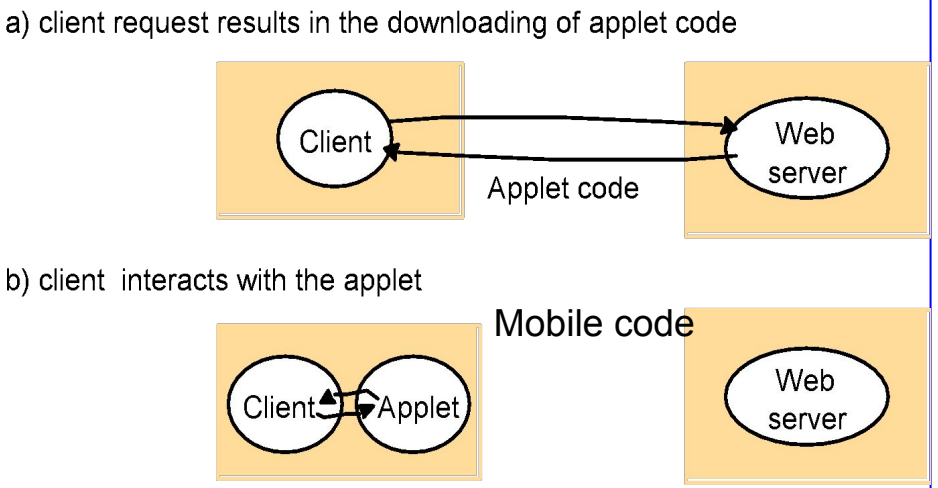


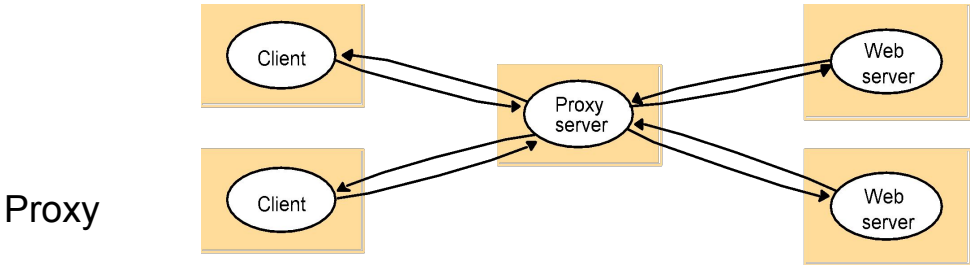
- Scalability – a concern
- Ineffective Utilization of resources at client
- Servers – administered entities

Architectural Models

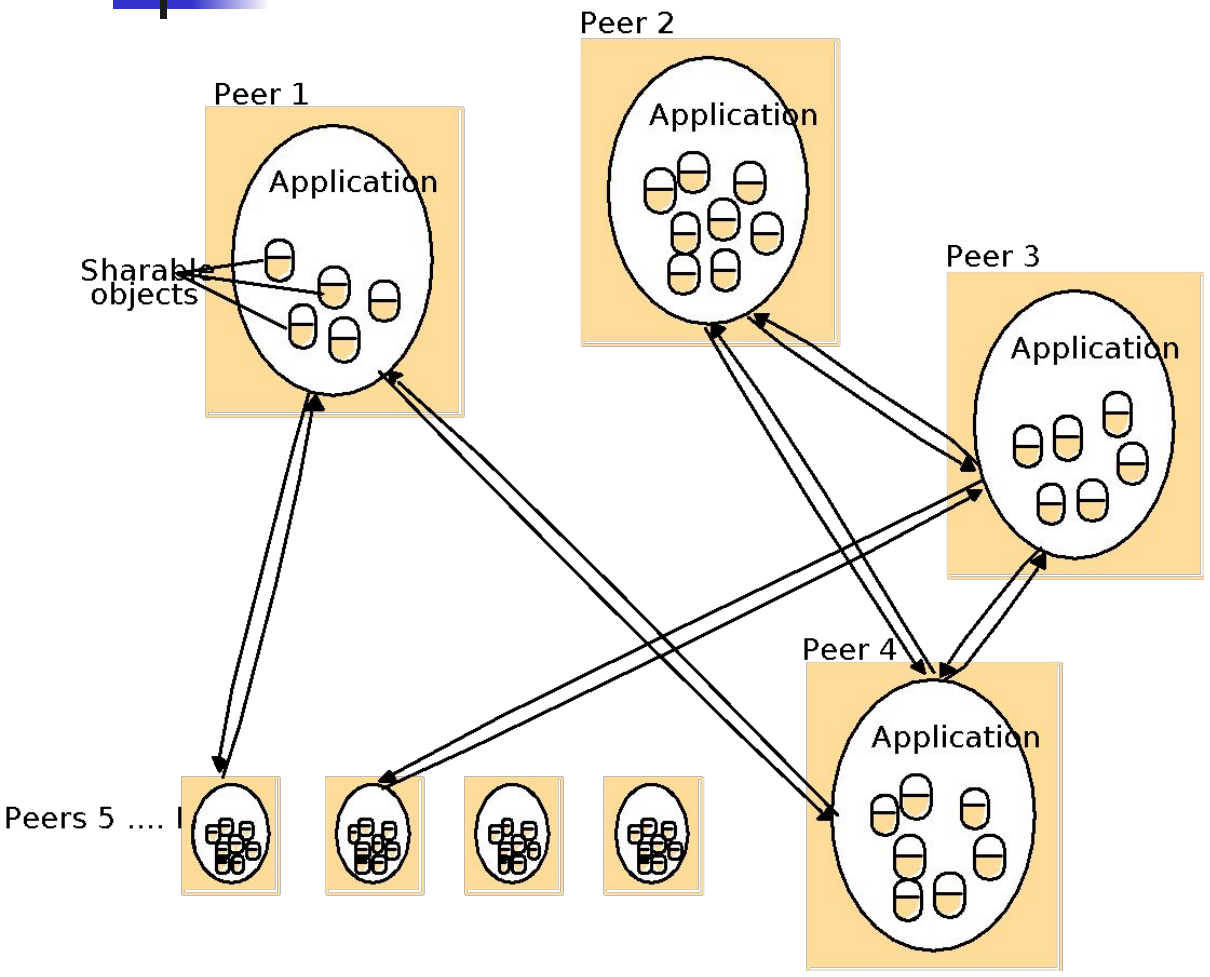
- Multiple servers, proxy servers and caches, mobile code, ...



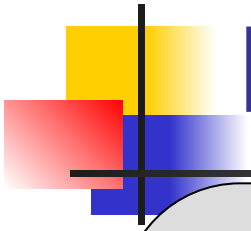




Architectural Models: Peer-to-peer



- No single node server as a server
- All nodes act as client (and server) at a time



P2P Systems

Use the vast resources of machines at the edge of the Internet to build a network that allows resource sharing without any central authority.

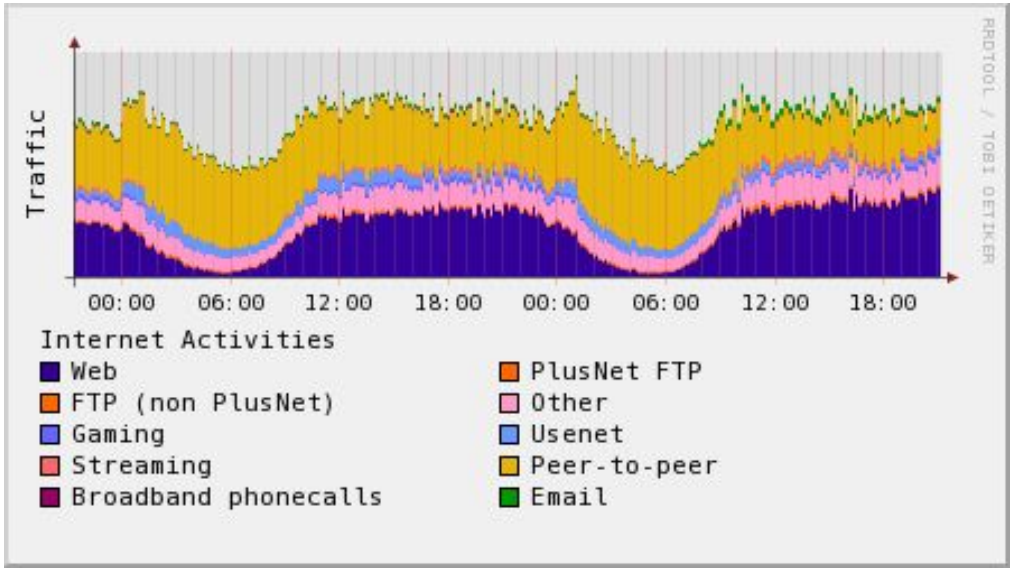
Began as a system for sharing pirated music/movies

2000s - P2P rising in popularity

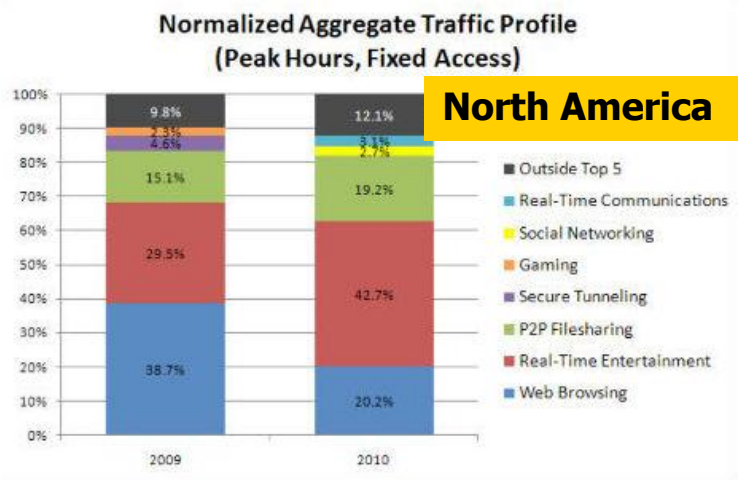
http://www.marketingvox.com/p4p-will-make-4-a-speedier-net-profs-say-040562/

Change of Yearly Internet Traffic

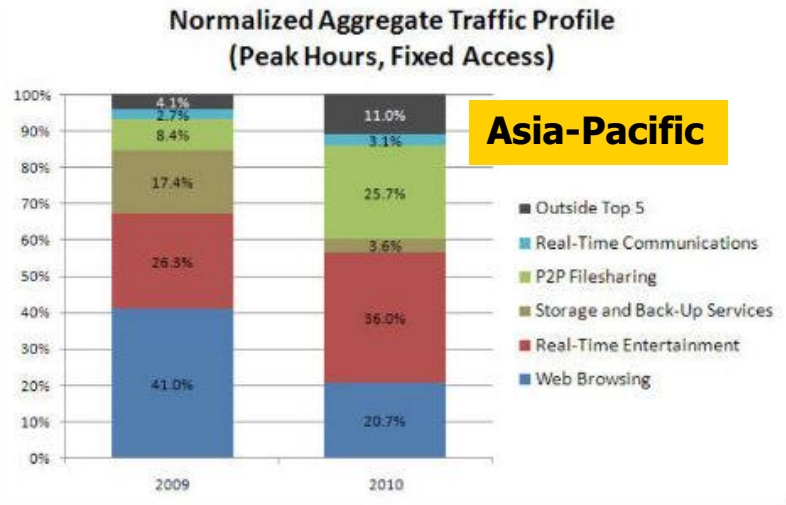
Daily Internet Traffic (2006)



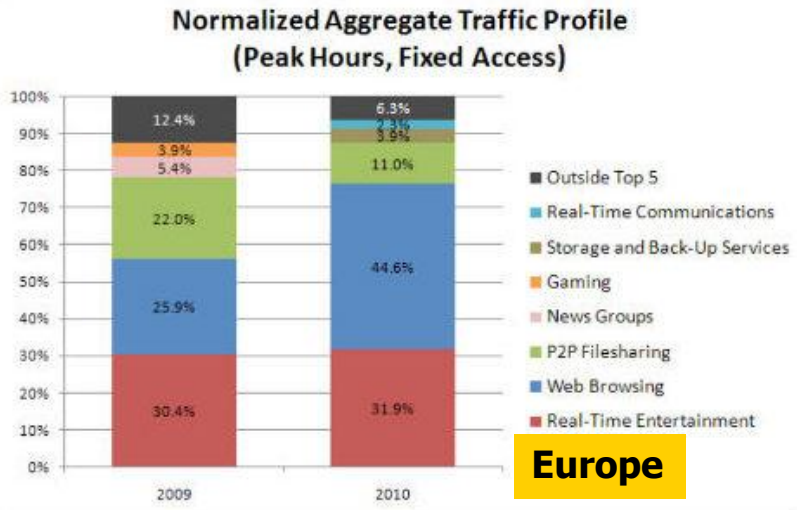
Across the world



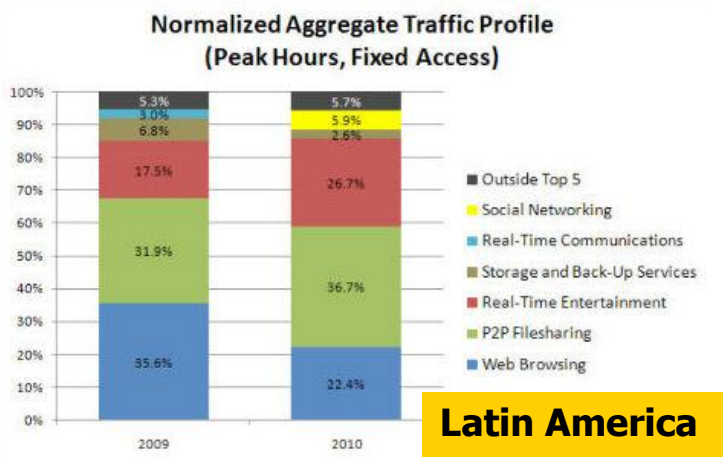
North America



Asia-Pacific



Europe



Latin America

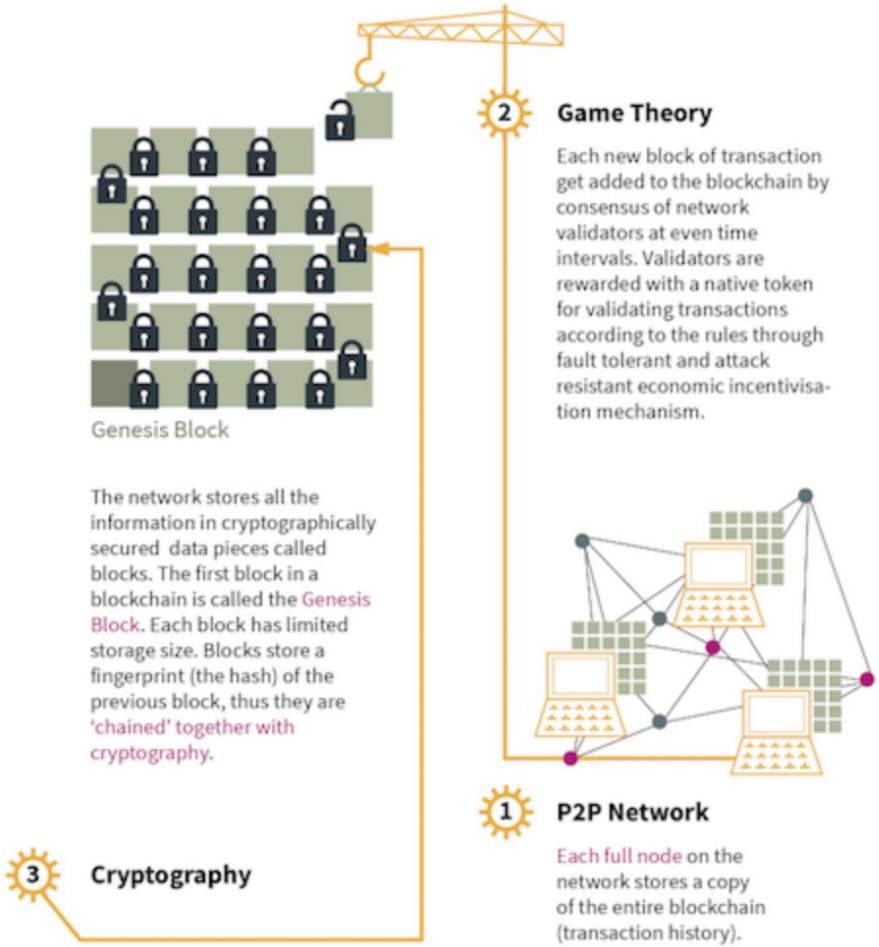
P2P applications today

- Bitcoin and alternatives such as Ether, Nxt and Peercoin are peer-to-peer-based digital cryptocurrencies.
- Dalesa, a peer-to-peer web cache for LANs (based on IP multicasting).
- FAROO, a peer-to-peer web search engine
- Filecoin is an open source, public, cryptocurrency and digital payment system intended to be a blockchain-based cooperative digital storage and data retrieval method.
- I2P, an overlay network used to browse the Internet anonymously.
- Infinit is an unlimited and encrypted peer to peer file sharing application for digital artists written in C++.
- The InterPlanetary File System (IPFS) is a protocol and network designed to create a content-addressable, peer-to-peer method of storing and sharing hypermedia distribution p
- JXTA, a peer-to-peer protocol designed for the Java platform.
- Netsukuku, a Wireless community network designed to be independent from the Internet.
- Open Garden, connection sharing application that shares Internet access with other devices using Wi-Fi or Bluetooth.
- Research like the Chord project, the PAST storage utility, the P-Grid, and the CoopNet content distribution system.
- Tradepal and M-commerce applications that power real-time marketplaces.
- The U.S. Department of Defense is conducting research on P2P networks as part of its modern network warfare strategy.[50] In May, 2003, Anthony Tether, then director of DAR
- WebTorrent is a P2P streaming torrent client in JavaScript for use in web browsers, as well as in the WebTorrent Desktop stand alone version that bridges WebTorrent and BitTo
- Tor (anonymity network)
- Microsoft in Windows 10 uses a proprietary peer to peer technology called "Delivery Optimization" to deploy operating system updates using end-users PCs either on the local r
- Artisoft's LANtastic was built as a peer-to-peer operating system. Machines can be servers and workstations at the same time.

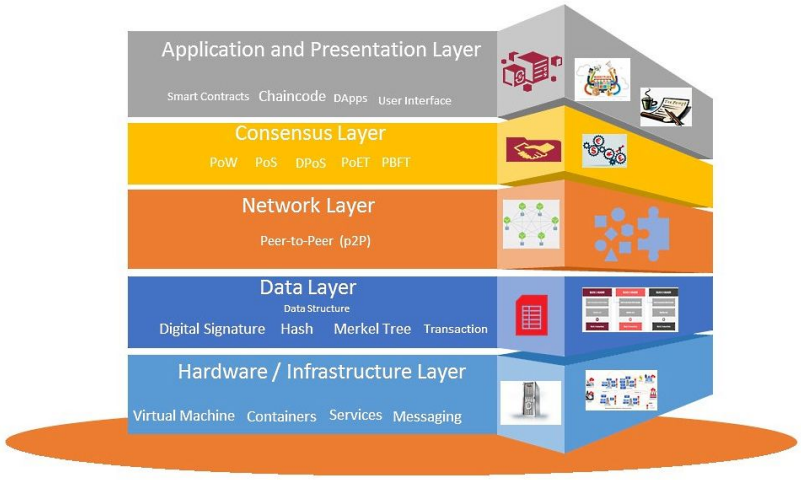
How P2P Networks Inspired Blockchain

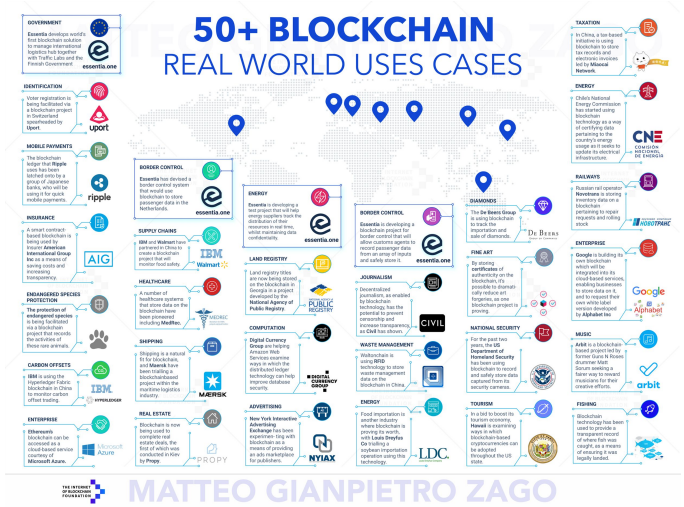
January 17th 2020 last updated

Why is it called Blockchain?



From the Book "**Token Economy**" by Shermin Voshmgir, 2019
Excerpts available on **https://blockchainhub.net**

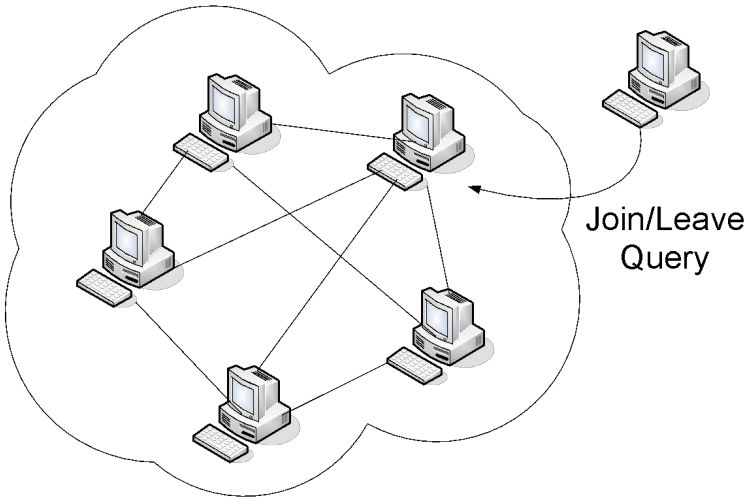




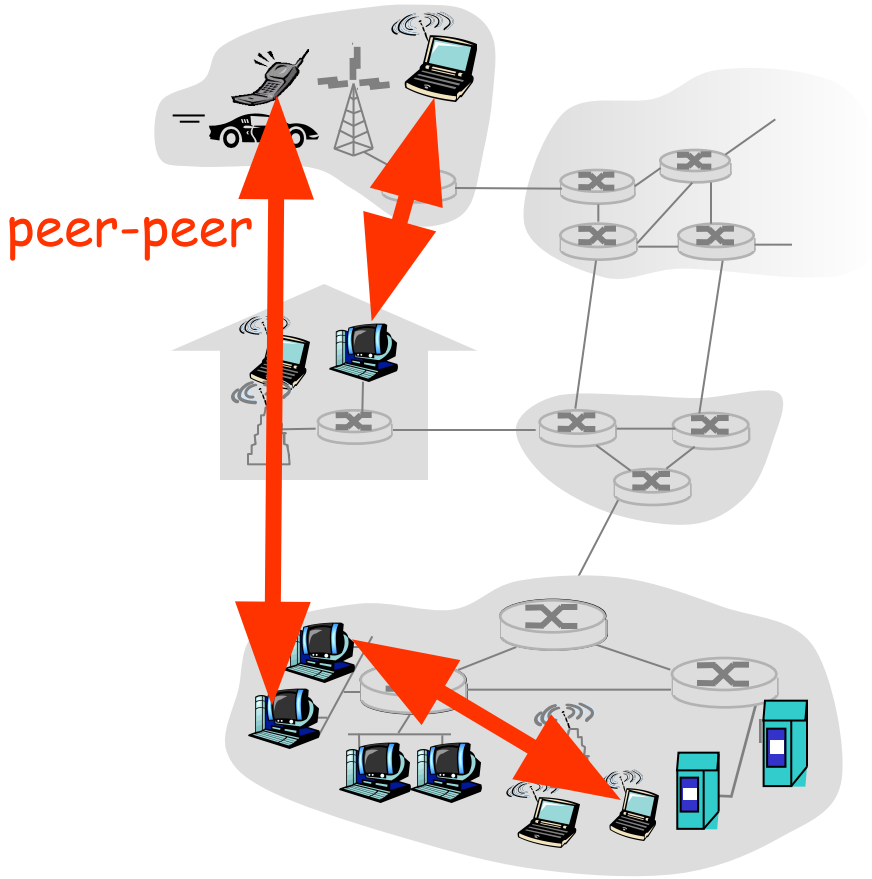
A Blockchain Tutorial
https://folk.uio.no/romanvi/Papers/bc-tutorial-debs-master.pdf

Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

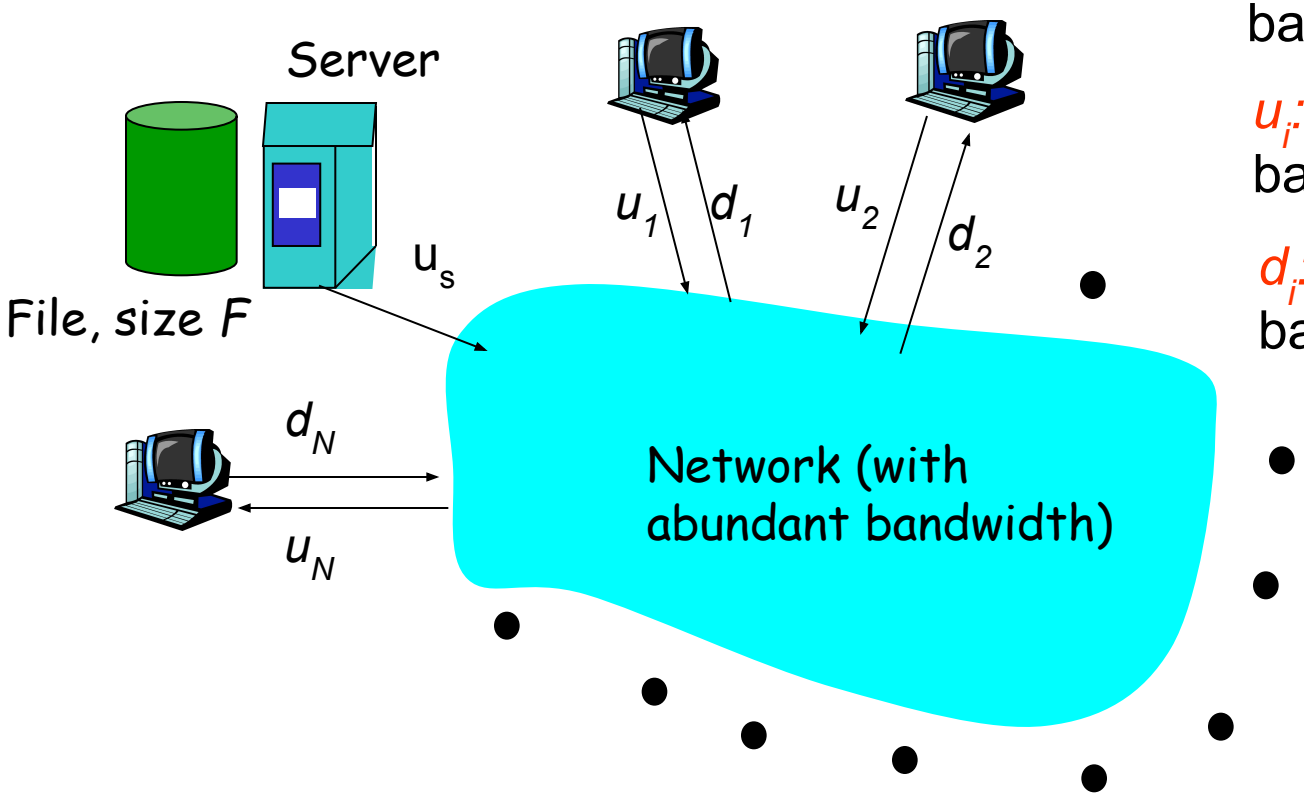


P2P System



File Distribution: Server-Client vs P2P

Question : How much time to distribute file from one server to *N peers*?



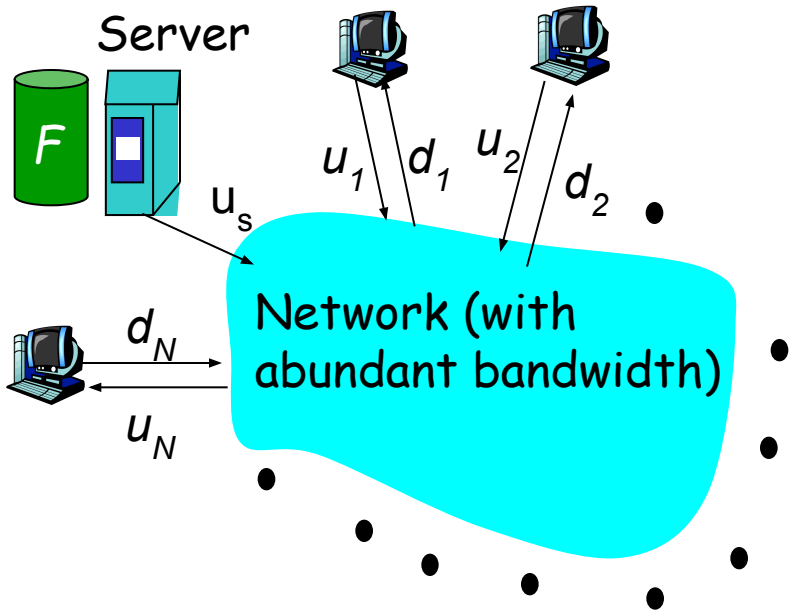
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download

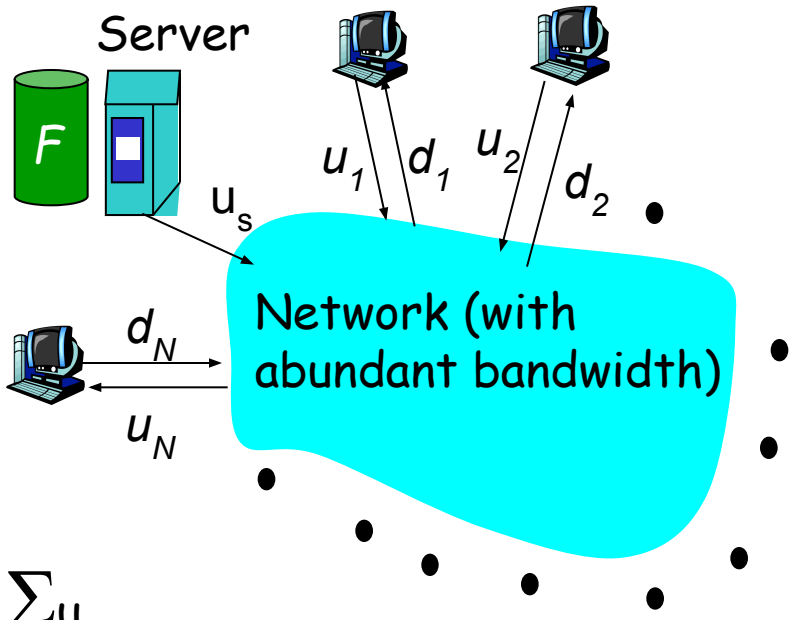


Time to distribute F to N clients using client/server approach $= d_{cs} = \max\{NF/u_s, F/\min(d_i)\}$

increases linearly in N (for large N)

File distribution time: P2P

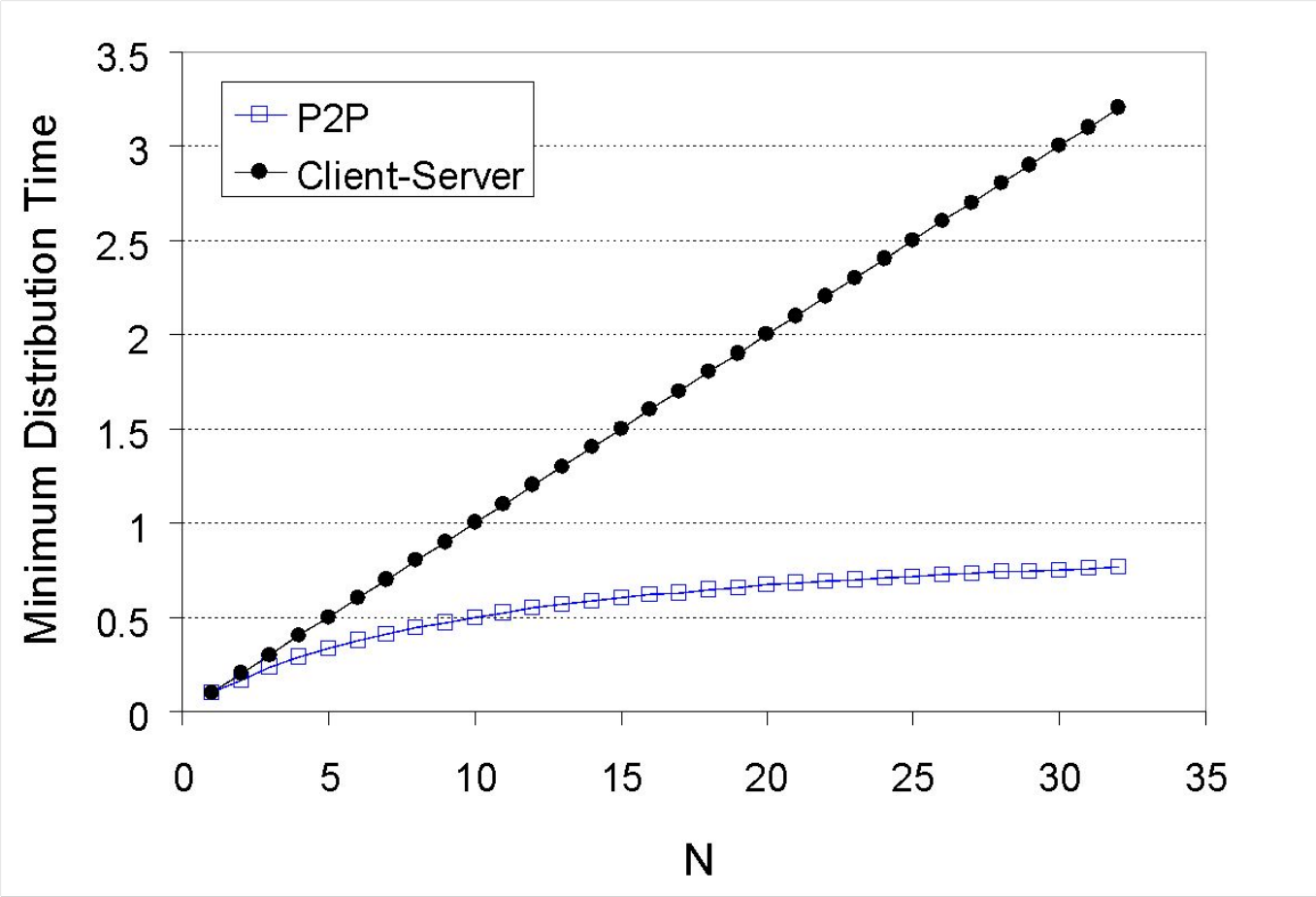
- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \sum u_i$



$$d_{P2P} = \max \left\{ F/u_s, F/\min(d_i) , NF/(u_s + \sum u_i) \right\}$$

Server-client vs. P2P: example

Client upload rate = u, F/u = 1 hour, u_s = 10u, $d_{min} \geq u_s$

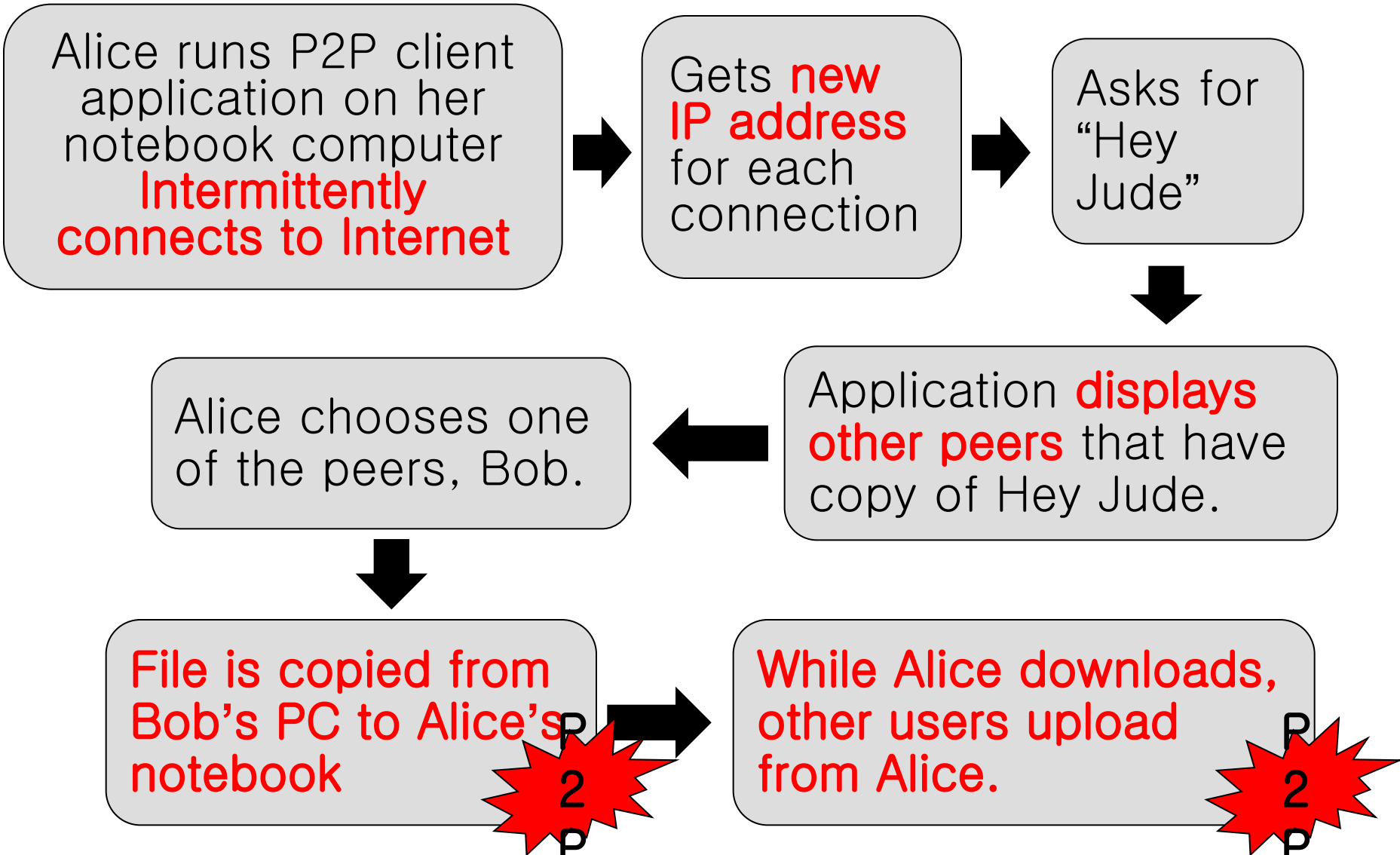


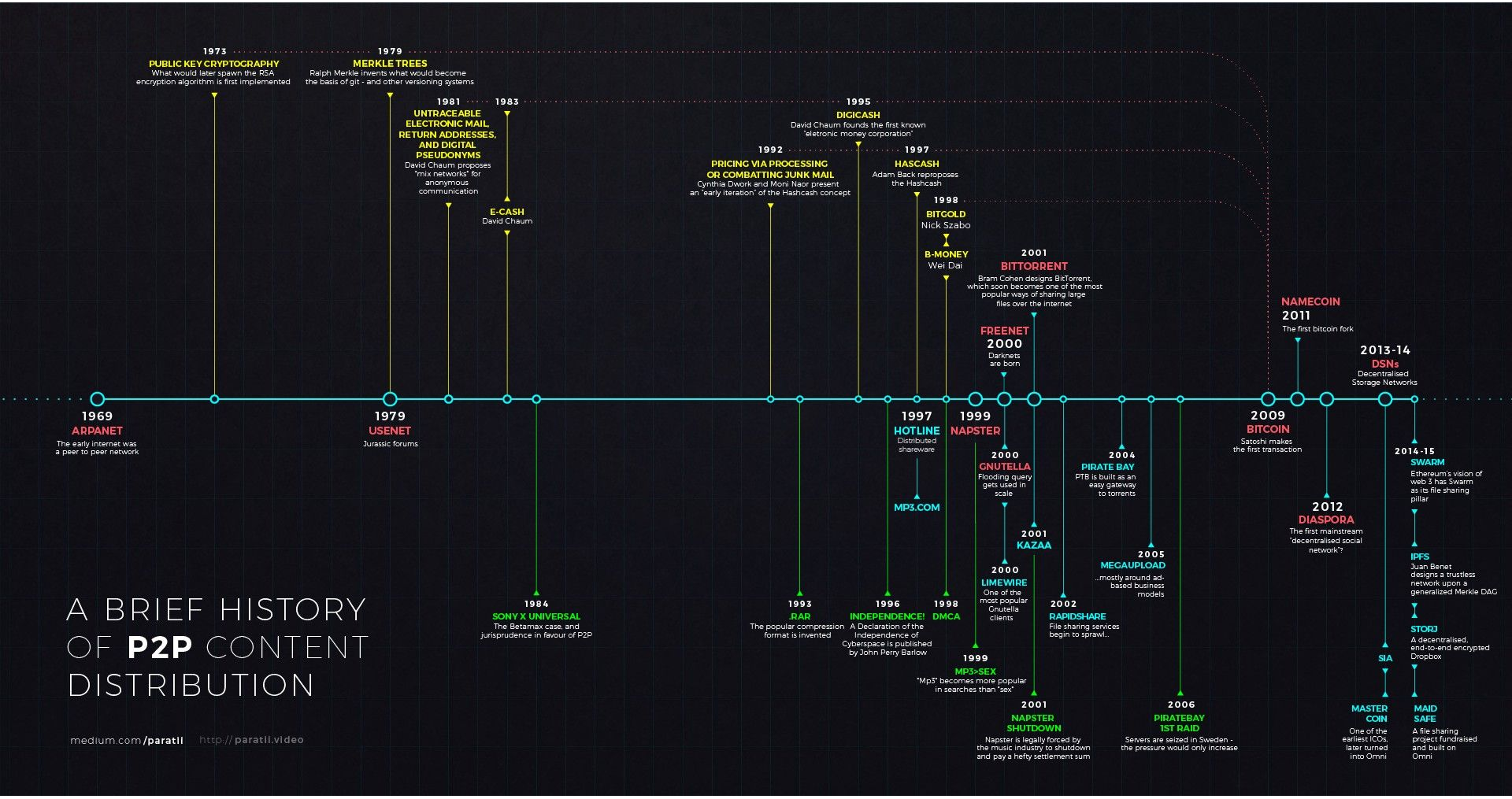
P2P Applications

P2P Applications

- P2P Search, File Sharing and Content dissemination
  - Napster, Gnutella, Kazaa, eDonkey, BitTorrent
  - Chord, CAN, Pastry/Tapestry, Kademlia,
  - Bullet, SplitStream, CREW, FareCAST
- P2P Communications
  - MSN, Skype, Social Networking Apps
- P2P Storage
  - OceanStore/POND, CFS (Collaborative FileSystems),TotalRecall, FreeNet, Wuala
- P2P Distributed Computing
  - Seti@home

P2P File Sharing



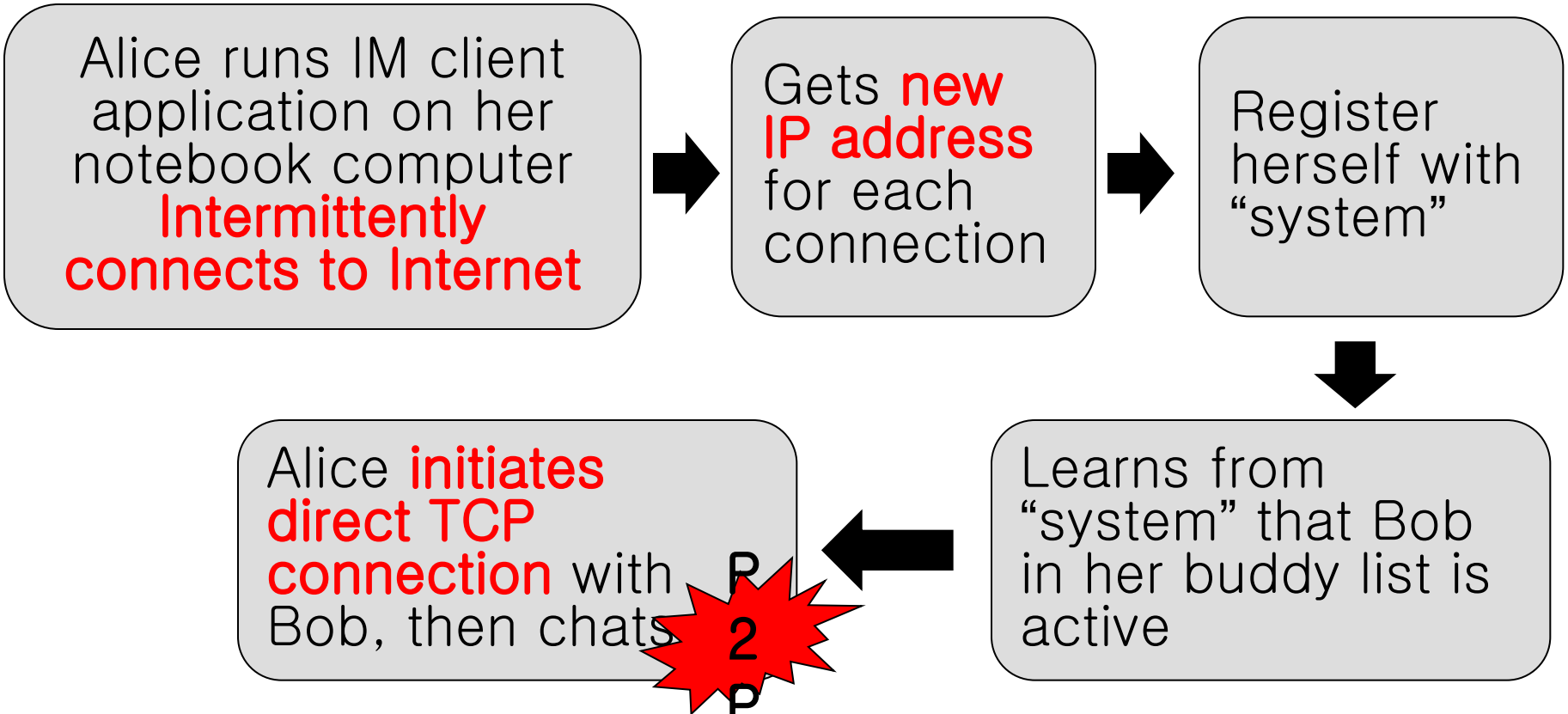


A BRIEF HISTORY OF **P2P** CONTENT DISTRIBUTION

medium.com/paratii http://paratii.video

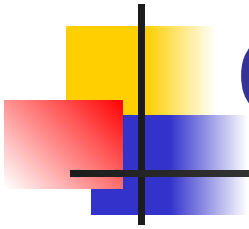
P2P Communication

- Instant Messaging
- Skype is a VoIP P2P system



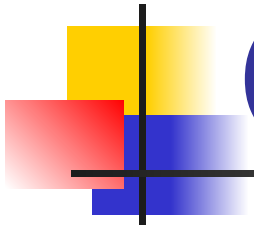
P2P/Grid Distributed Processing

- seti@home
  - Search for ET intelligence
  - Central site collects radio telescope data
  - Data is divided into work chunks of 300 Kbytes
  - User obtains client, which runs in background
  - **Peer sets up TCP connection to central computer, downloads chunk**
  - **Peer does FFT on chunk, uploads results, gets new chunk**
- Not P2P communication, but exploit Peer computing power
- Crowdsourcing – Human-oriented P2P



Characteristics of P2P Systems

- Exploit edge resources.
  - Storage, content, CPU, Human presence.
- Significant autonomy from any centralized authority.
  - Each node can act as a Client as well as a Server.
- Resources at edge have intermittent connectivity, constantly being added & removed.
  - Infrastructure is untrusted and the components are unreliable.



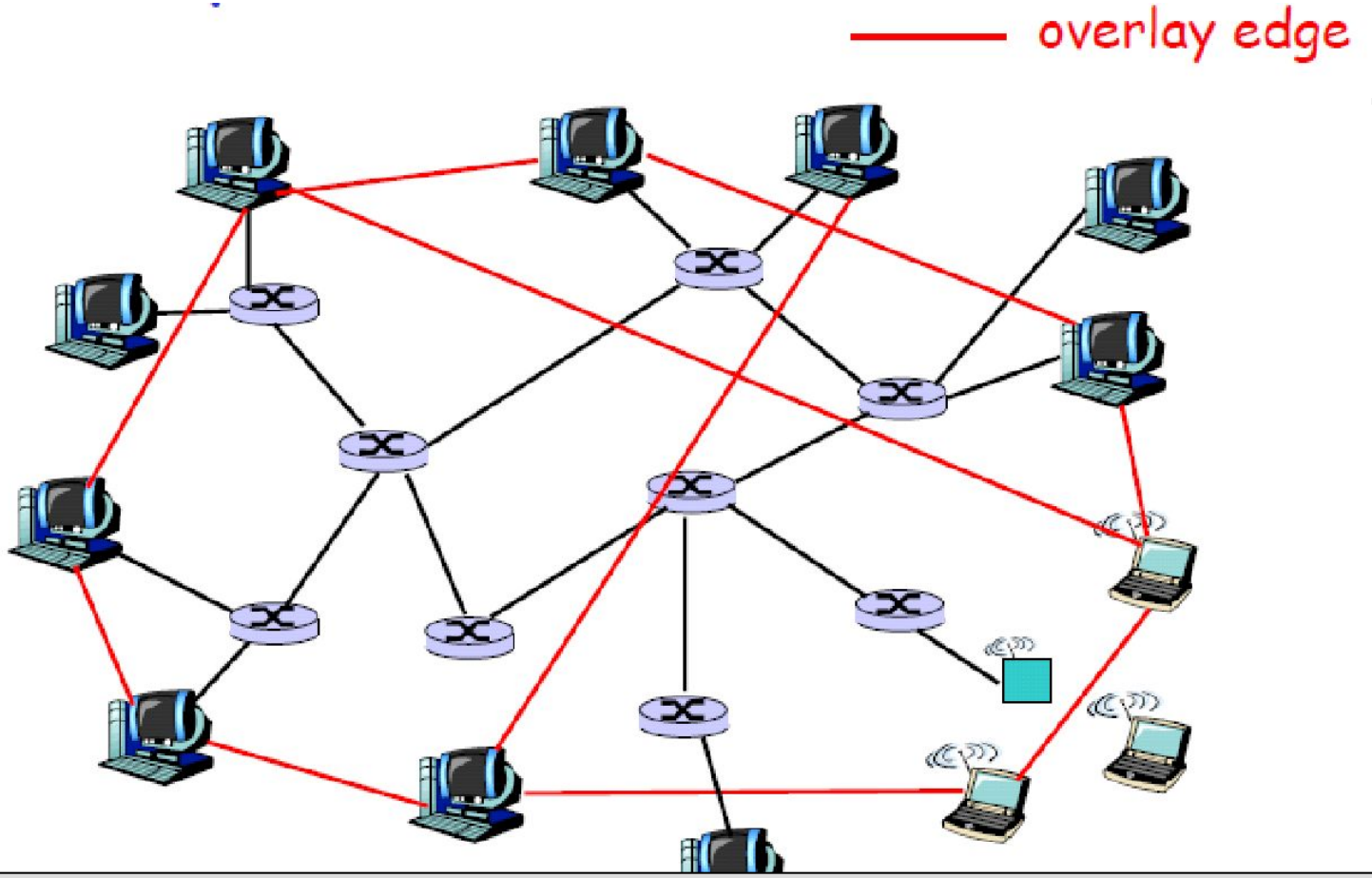
(Promising) properties of P2P

- Self-organizing
- Massive scalability
- Autonomy : non single point of failure
- Resilience to Denial of Service
- Load distribution
- Resistance to censorship

"The ultimate form of democracy on the Internet"

"The ultimate threat to copyright protection on the Internet"

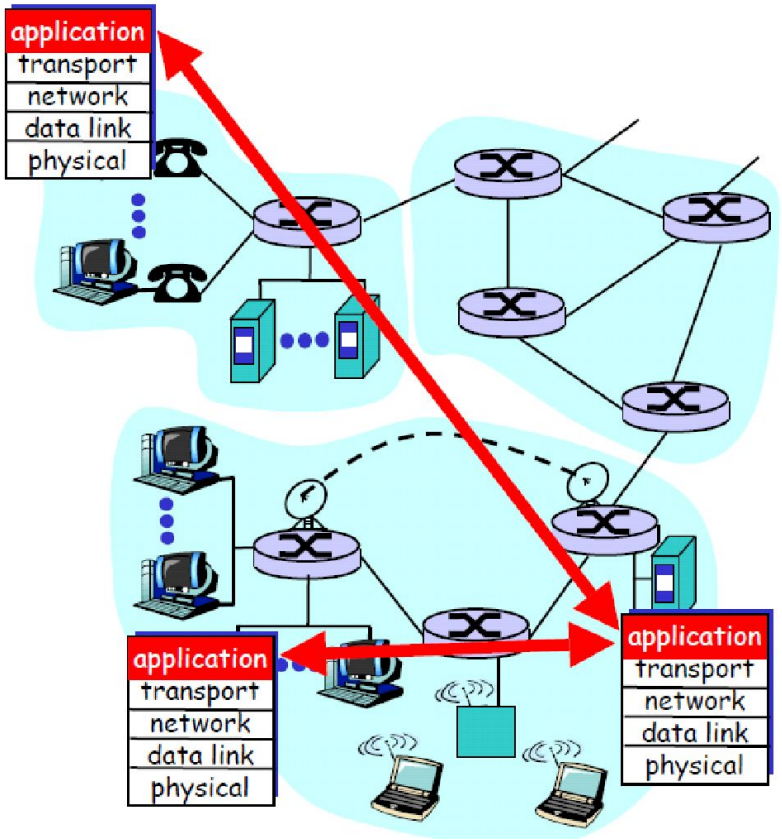
Overlay Network



A P2P network is an overlay network. Each link between peers consists of one or more IP links.

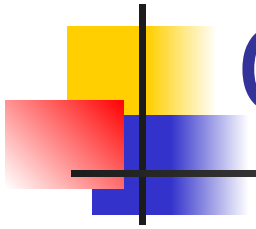
Overlays : All in the application layer

- **Tremendous design flexibility**
  - Topology, maintenance
  - Message types
  - Protocol
  - Messaging over TCP or UDP
- **Underlying physical network is transparent to developer**
  - But some overlays exploit proximity



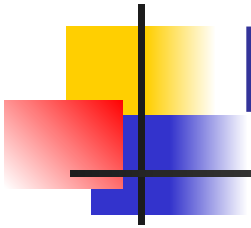
Overlay Graph

- 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 aliveness while messaging
  - If neighbor goes down, may want to establish new edge
  - New incoming node needs to bootstrap
  - **Could be a challenge under high rate of churn**
    - **Churn : dynamic topology and intermittent access due to node arrival and failure**



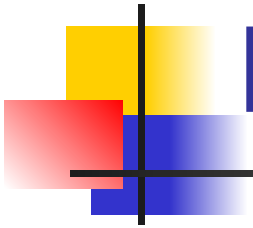
Overlay Graph

- **Unstructured overlays**
  - e.g., new node randomly chooses existing nodes as neighbors
- **Structured overlays**
  - e.g., edges arranged in restrictive structure
- **Hybrid Overlays**
  - Combines structured and unstructured overlays
    - SuperPeer architectures where superpeer nodes are more stable typically
    - Get metadata information from structured node, communicate in unstructured manner



Key Issues

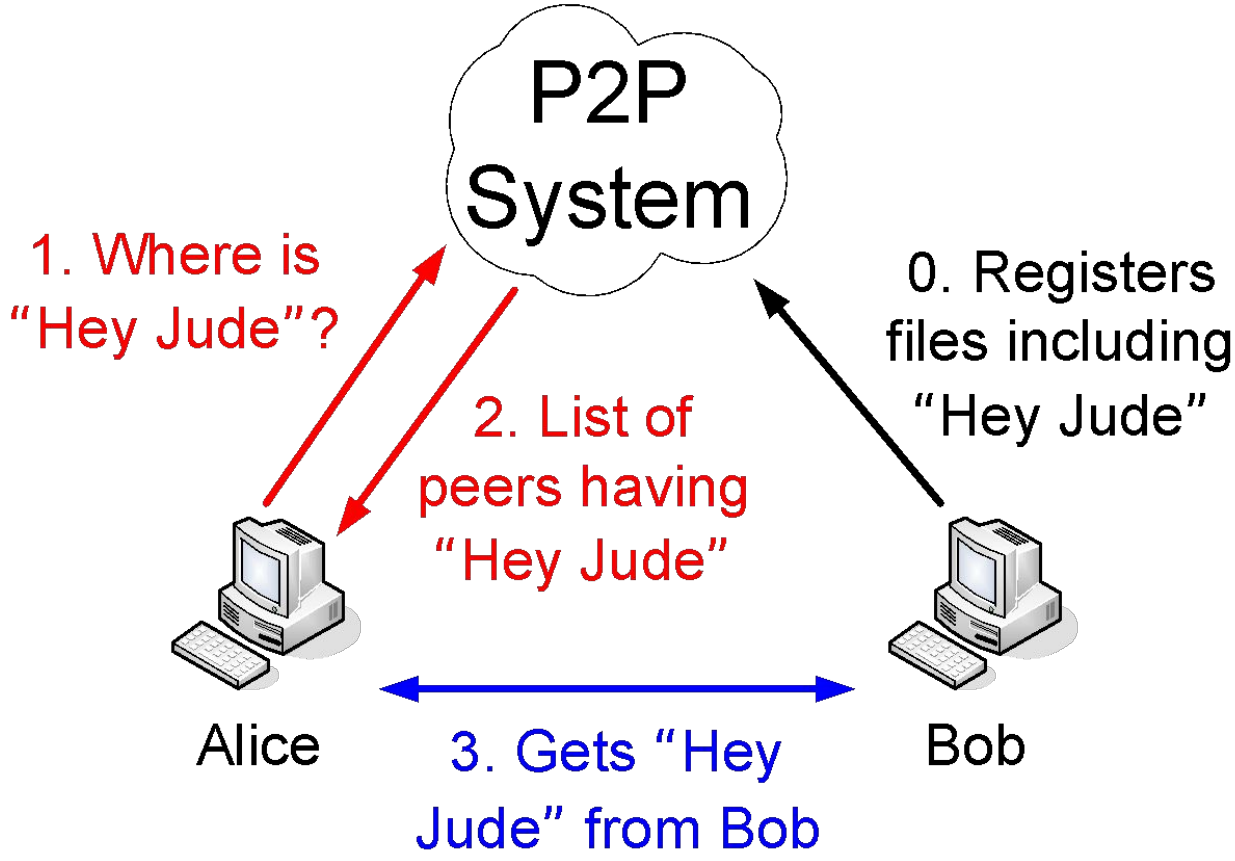
- Lookup
  - How to find out the appropriate content/resource that a user wants
- Management
  - How to maintain the P2P system under high rate of churn efficiently
  - Application reliability is difficult to guarantee
- Throughput
  - Content distribution/dissemination applications
  - How to copy content fast, efficiently, reliably



Lookup Issue

- Centralized vs. decentralized
- How do you locate data/files/objects in a large P2P system built around a dynamic set of nodes in a scalable manner without any centralized server or hierarchy?
- Efficient routing even if the structure of the network is unpredictable.
  - Unstructured P2P : Napster, Gnutella, Kazaa
  - Structured P2P : Chord, CAN, Pastry/Tapestry, Kademlia

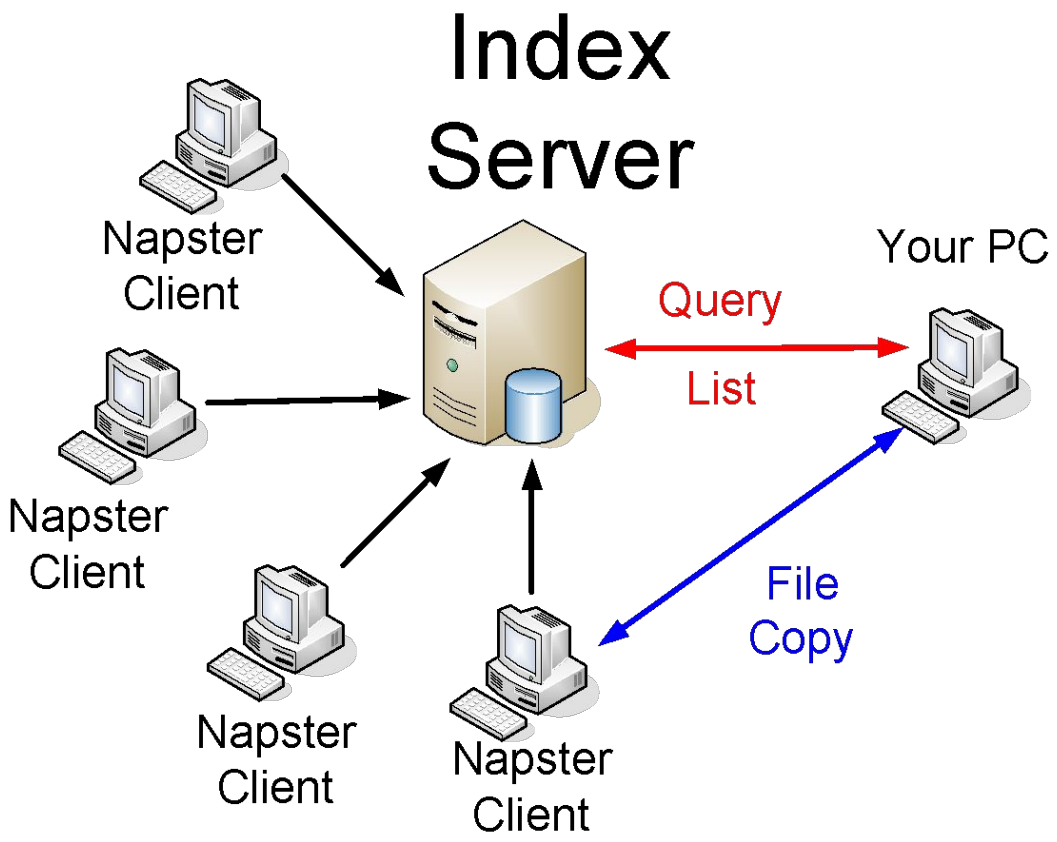
Lookup Example : File Sharing Scenario



Napster

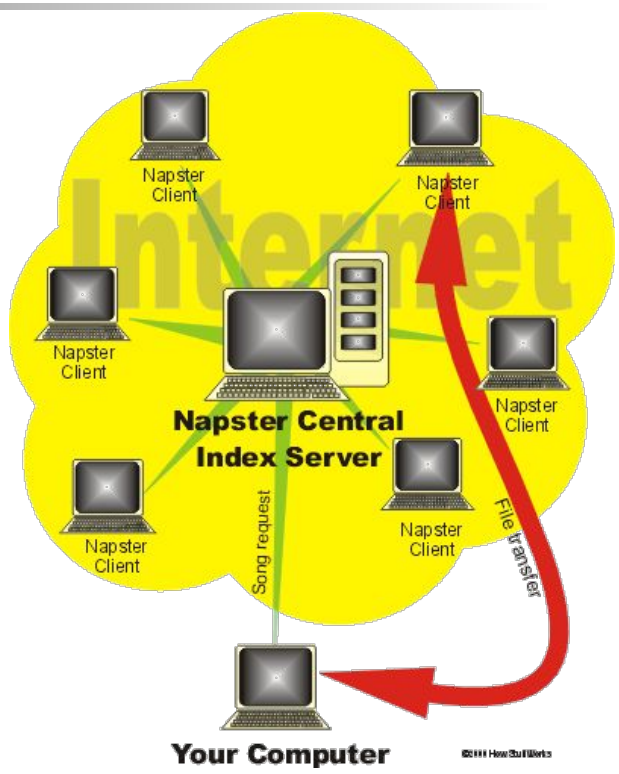
- First P2P file-sharing application (June 1999)
- Only MP3 sharing possible
- **Based on central index server**
- Clients register and give list of files to share
- Searching based on keywords
  - Response : List of files with additional information, e.g. peer's bandwidth, file size

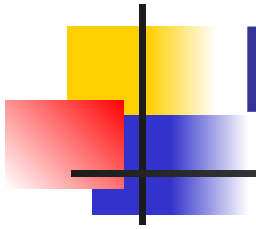
Napster Architecture



Centralized Lookup

- Centralized directory services
  - Steps
    - Connect to Napster server.
    - Upload list of files to server.
    - Give server keywords to search the full list with.
    - Select "best" of correct answers. (ping)
  - **Performance Bottleneck**
- Lookup is centralized, but files are copied in P2P manner





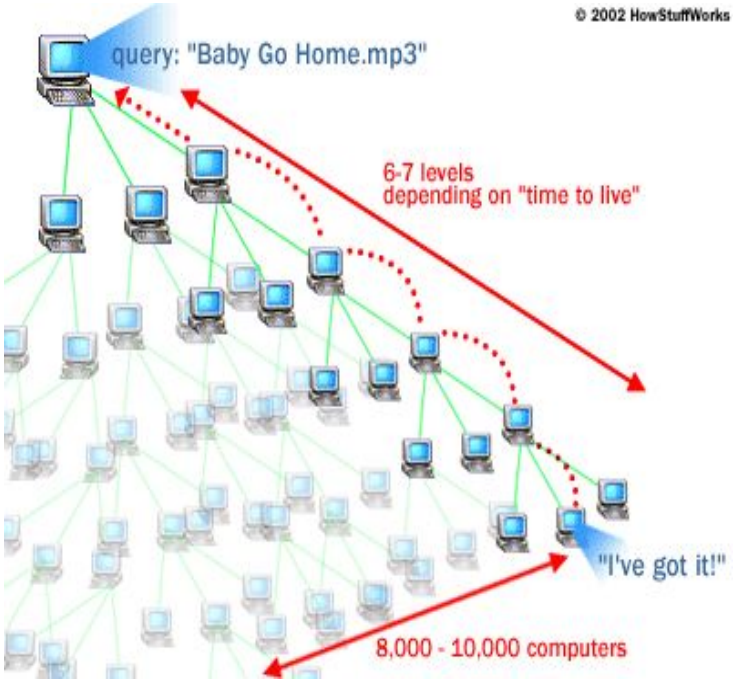
Pros and cons of Napster

- Pros
  - Fast, efficient and overall search
  - Consistent view of the network
- Cons
  - Central server is a single point of failure
  - Central server is a bottleneck; maybe expensive to maintain (designed to share MP3 files - few MBs).
  - Susceptible to DOS attacks
  - Lawsuits (copyrights infringement?)

Gnutella

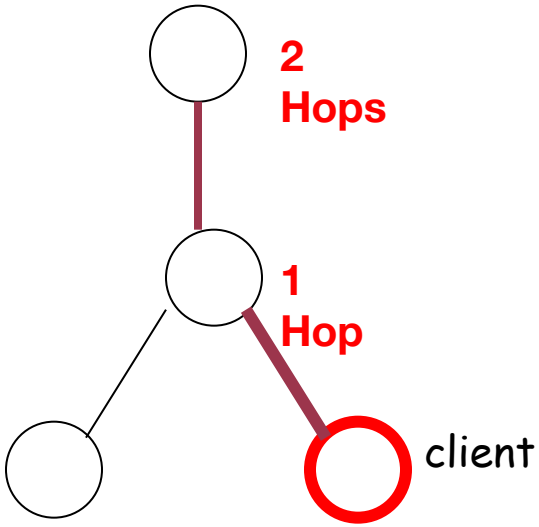
Originally developed at Nullsoft (AOL)

- **Fully distributed system**
  - No index server – address Napster's weaknesses
  - All peers are fully equal
  - A peer needs to know another peer, that is already in the network, to join
  - Ping/Pong Protocol
- **Flooding based search**
  - **Variation: Random walk based search**
  - Direct download
- Open protocol specifications



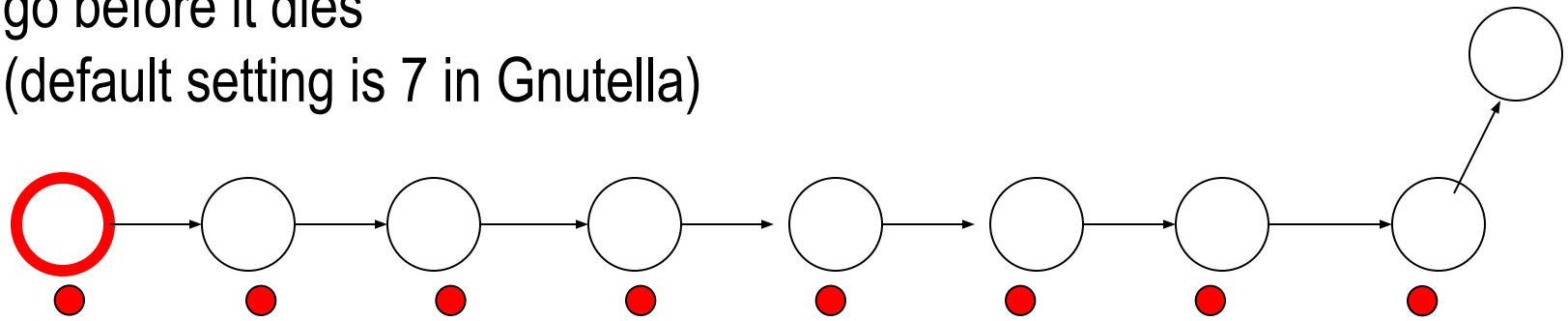
Gnutella : Terms

Servent: A Gnutella node. Each servant is both a server and a client.

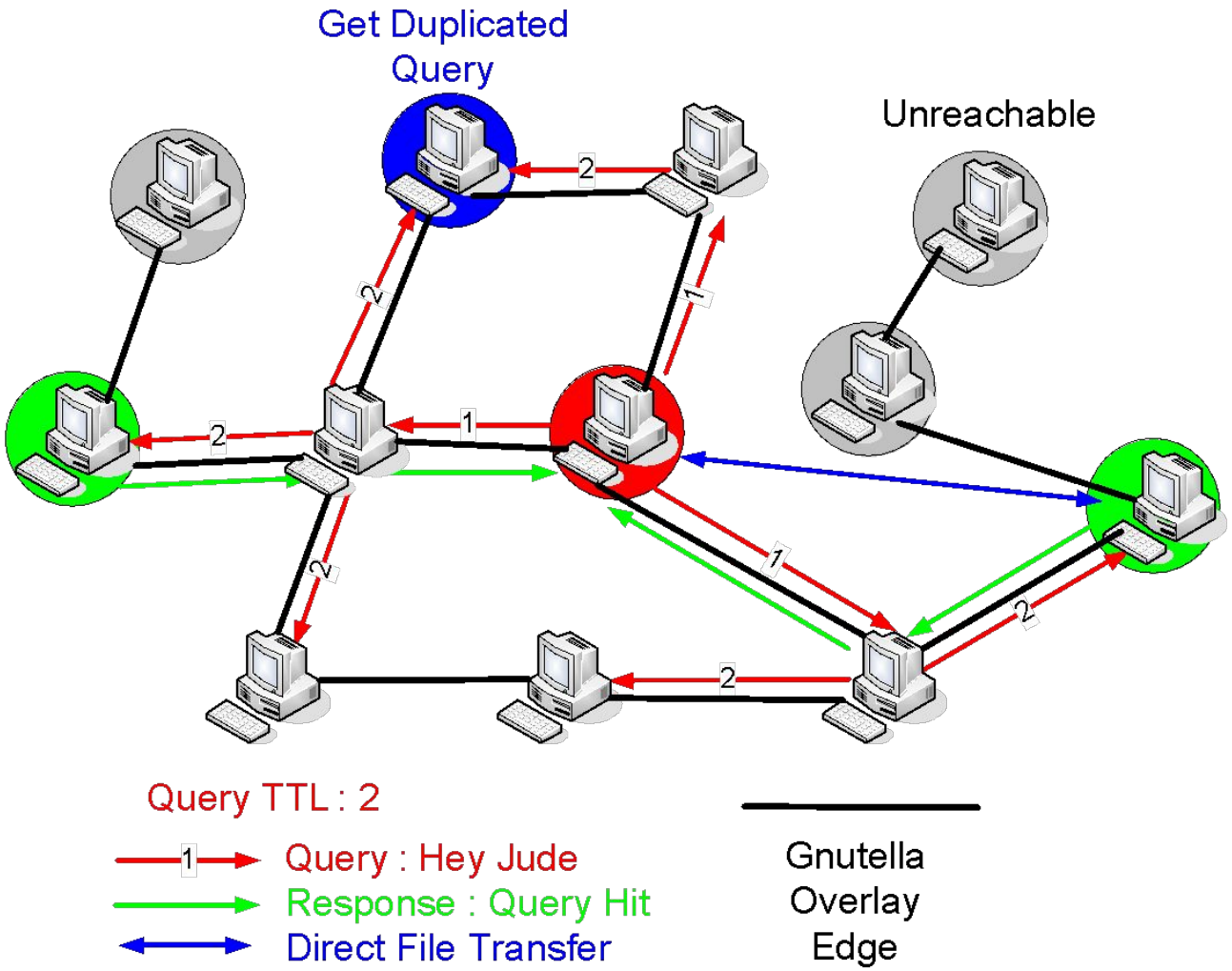


Hops: a hop is a pass through an intermediate node

TTL: how many hops a packet can go before it dies
(default setting is 7 in Gnutella)



Gnutella operation : Flooding based lookup



Gnutella : Scenario

Step 0: Join the network

Step 1: Determining who is on the network

- **"Ping"** packet is used to announce your presence on the network.
- Other peers respond with a **"Pong"** packet.
- Also forwards your Ping to other connected peers
- A Pong packet also contains:
  - an IP address
  - port number
  - amount of data that peer is sharing
  - Pong packets come back via same route

Step 2: Searching

- Gnutella "**Query**" ask other peers (usually 7) if they have the file you desire
- A Query packet might ask, ***"Do you have any content that matches the string 'Hey Jude"?***
- Peers check to see if they have matches & respond (if they have any matches) & send packet to connected peers if not (usually 7)
- Continues for TTL (how many hops a packet can go before it dies, typically 7)

Step 3: Downloading

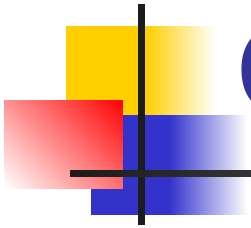
- Peers respond with a "**QueryHit**" (contains contact info)
- File transfers use direct connection using HTTP protocol's GET method

Gnutella : Reachable Users by flood based lookup

T : TTL, *N* : Neighbors for Query

	T=1	*T=2*	*T=3*	*T=4*	*T=5*	*T=6*	*T=7*
N=2	2	4	6	8	10	12	14
N=3	3	9	21	45	93	189	381
N=4	4	16	52	160	484	1,456	4,372
N=5	5	25	105	425	1,705	6,825	**27,305**
N=6	6	36	186	936	4,686	**23,436**	**117,186**
N=7	7	49	301	1,813	**10,885**	**65,317**	**391,909**
N=8	8	64	456	3,200	**22,408**	**156,864**	**1,098,056**

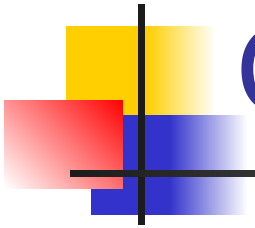
(analytical estimate)



Gnutella : Lookup Issue

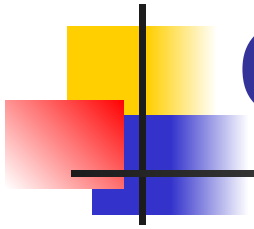
- Simple, but lack of scalability
  - Flooding based lookup is extremely wasteful with bandwidth
  - Enormous number of redundant messages
  - All users do this in parallel: local load grows linearly with size
- Sometimes, existing objects may not be located due to limited TTL

Possible extensions to make Gnutella efficient



- Controlling topology to allow for better search
  - Random walk, Degree-biased Random Walk
- Controlling placement of objects
  - Replication (1 hop or 2 hop)

Gnutella Topology



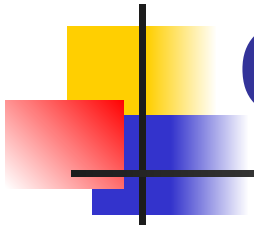
- The topology is dynamic, I.e. constantly changing.
- How do we model a constantly changing topology?
  - Usually, we begin with a static topology, and later account for the effect of churn.
- A Random Graph?
- A Power Law Graph?

Random graph: Erdös-Rényi model

- A random graph G(n, p) is constructed by starting with a set of n vertices, and adding edges between pairs of nodes at random.
- Every possible edge occurs independently with probability p.

- *Is Gnutella topology a random graph?*
  - *NO*

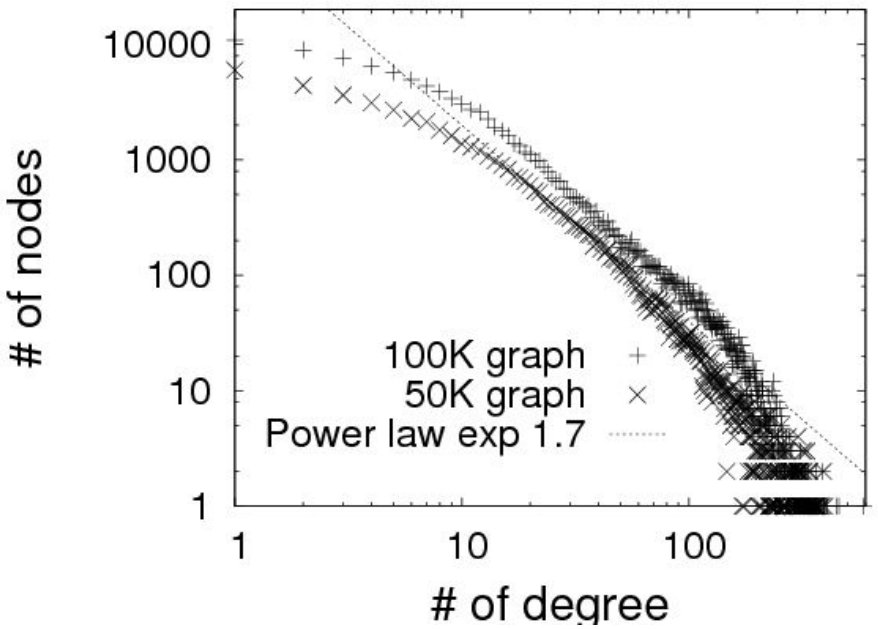
Gnutella : Power law graph



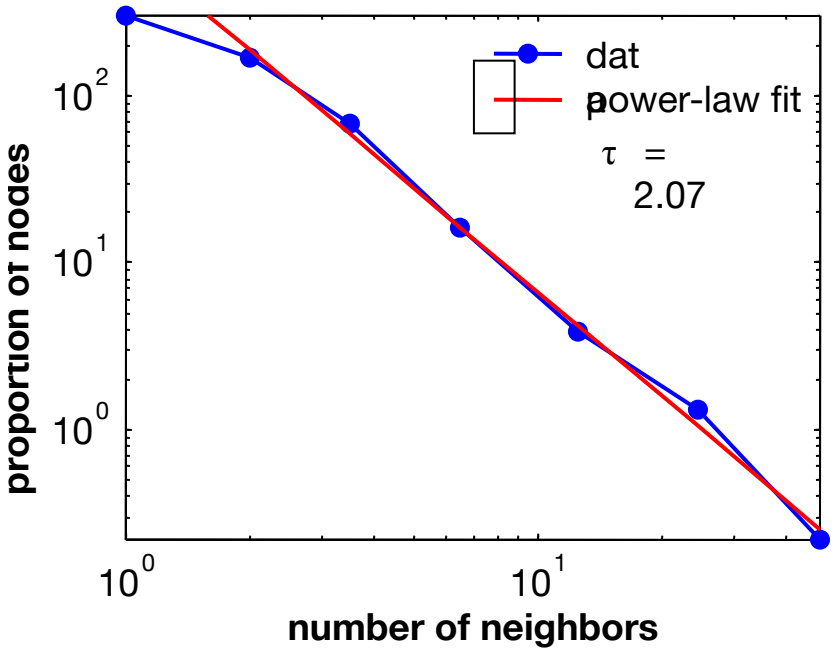
- Gnutella topology is actually a power-law graph.
  - Also called scale-free graph
- What is a power-law graph?
  - The number of nodes with degree k = ck^{-r}
  - Ex) WWW, Social Network, etc
  - Small world phenomena – low degree of separation (approx. log of size)

Power-law Examples

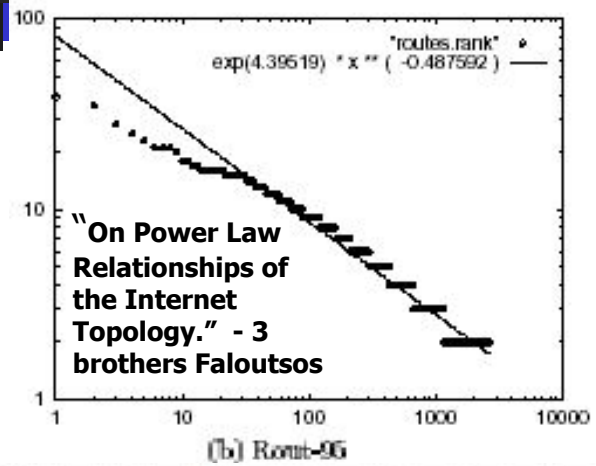
Facebook
power-law friends distribution



Gnutella
power-law link distribution



Other examples of power-law



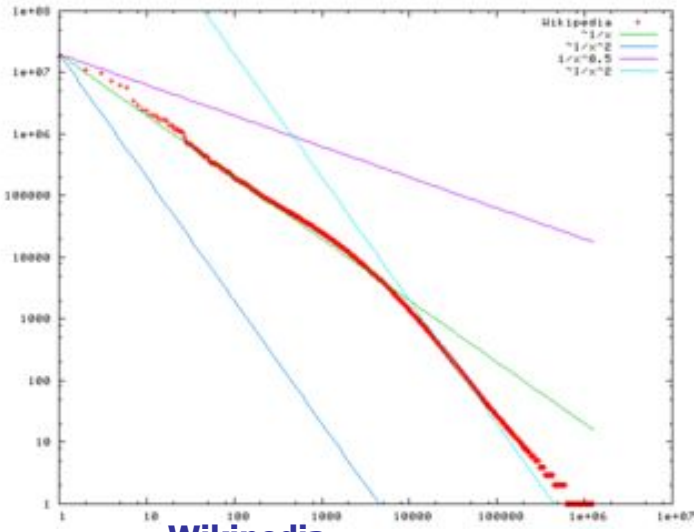
"On Power Law Relationships of the Internet Topology." - 3 brothers Faloutsos

(b) Route-96

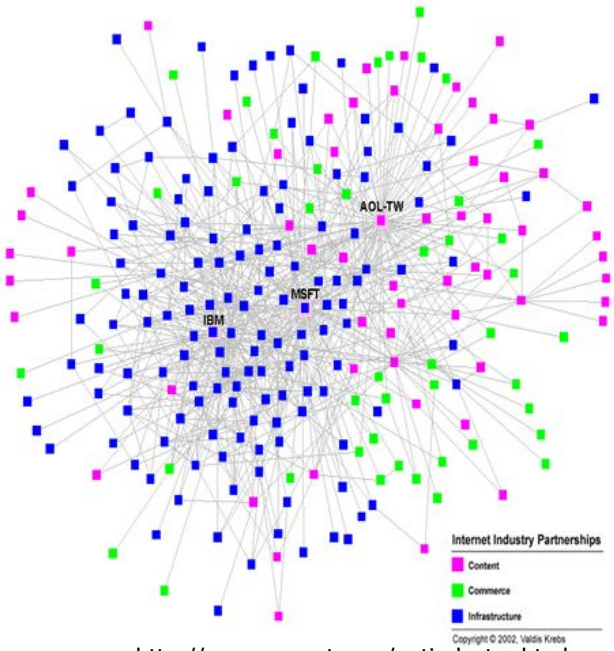
Frequent Word	Number of Occurrences	Percentage of Total
the	7,398,934	5.9
of	3,893,790	3.1
to	3,364,653	2.7
and	3,320,687	2.6
in	2,311,785	1.8
is	1,559,147	1.2
for	1,313,561	1.0
The	1,144,860	0.9
that	1,066,503	0.8
said	1,027,713	0.8

Frequencies from 336,310 documents in the 1GB TREC Volume 3 Corpus
125,720,891 total word occu

Dictionaries



Wikipedia



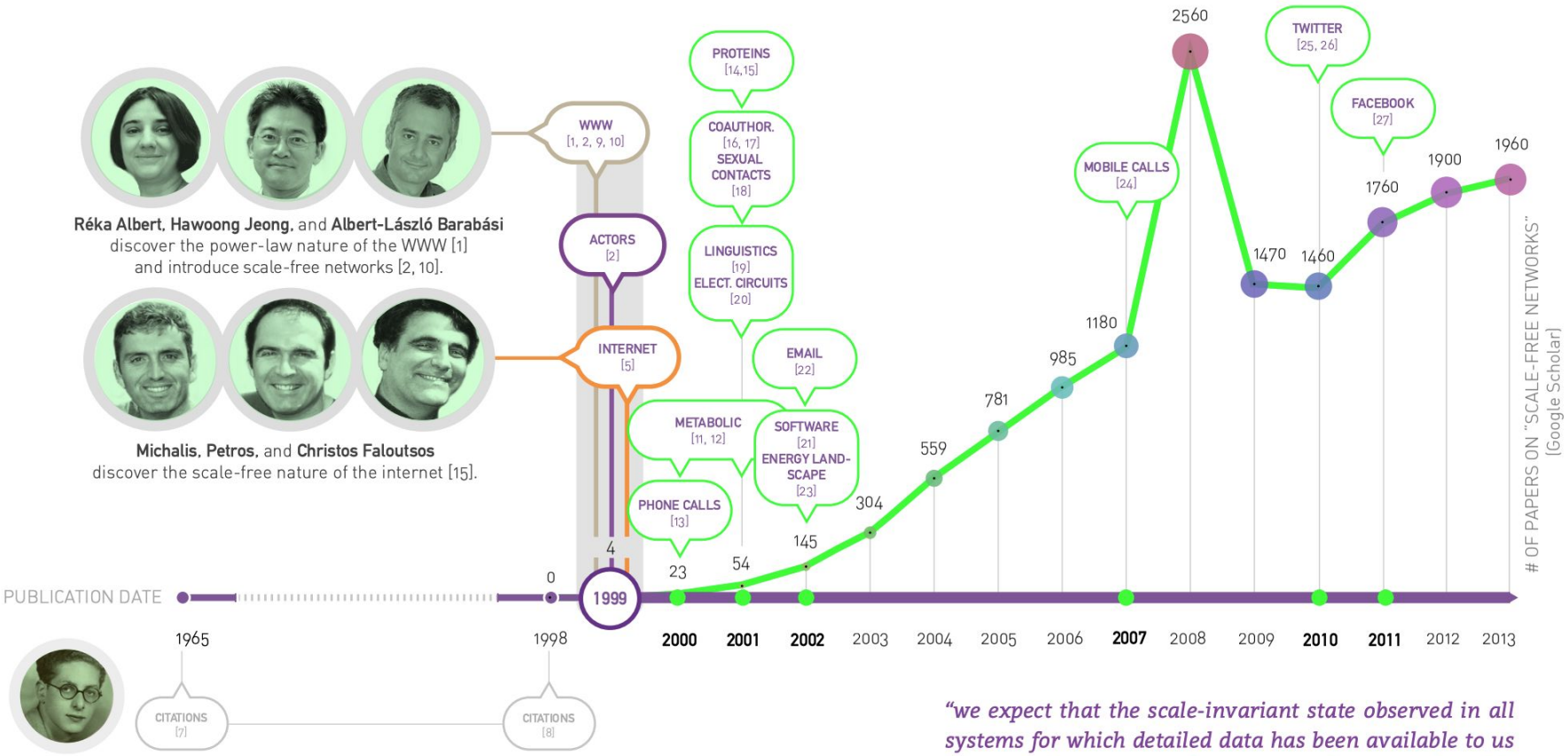
Internet Industry partnerships

http://www.orgnet.com/netindustry.html

https://barabasi.com/f/623.pdf

BOX 4.2

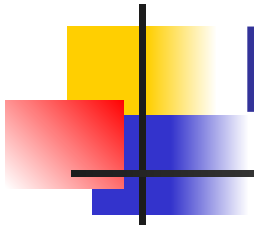
TIMELINE: SCALE-FREE NETWORKS



"we expect that the scale-invariant state observed in all systems for which detailed data has been available to us is a generic property of many complex networks, with applicability reaching far beyond the quoted examples."

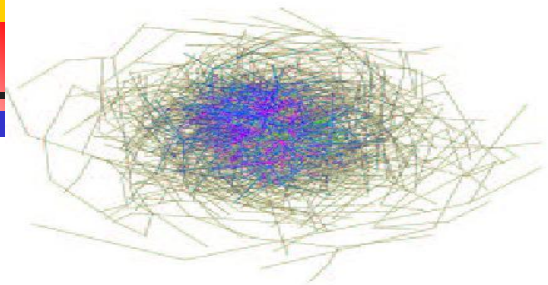
Barabási and Albert, 1999

Possible Explanation of Power-Law graph

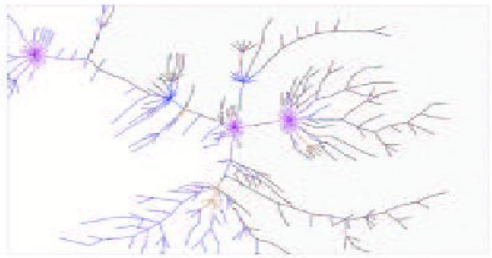


- Continued growth
  - Nodes join at different times.
- Preferential Attachment
  - The more connections a node has, the more likely it is to acquire new connections ("Rich gets richer").
    - Popular webpages attract new pointers.
    - Popular people attract new followers.

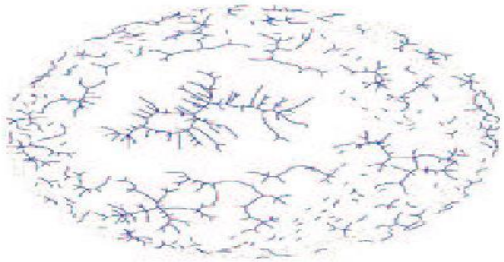
Power-Law Overlay Approach



Full Network



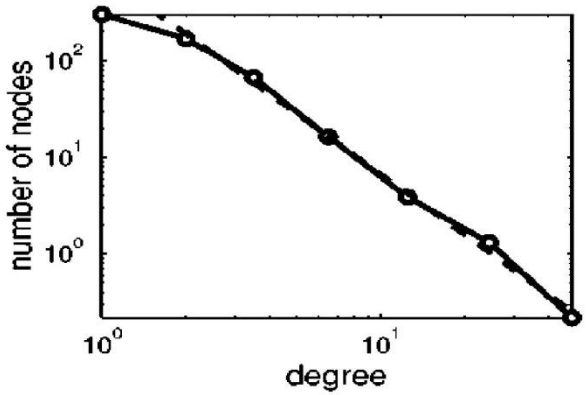
30% Random Removed



Top 4% Removed

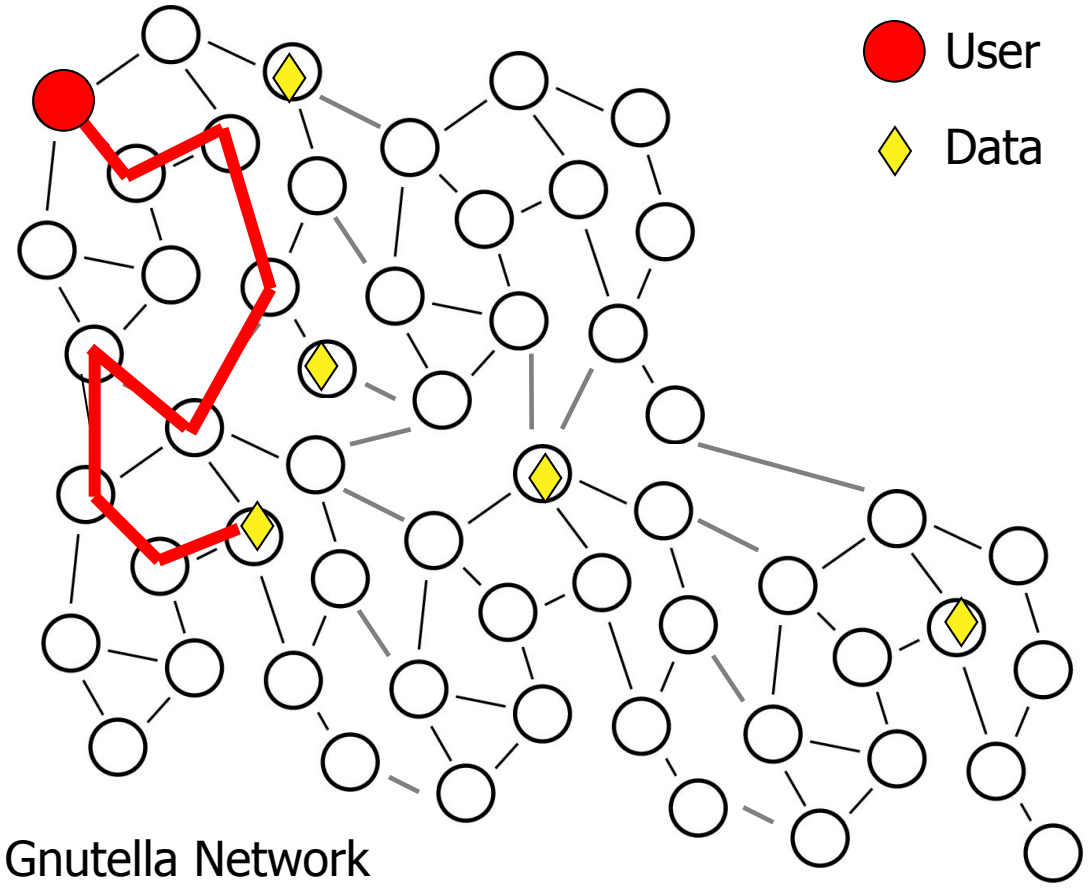
- Power-law graphs are
  - Resistant to random failures
  - Highly susceptible to directed attacks (to "hubs")
- Even if we can assume random failures
  - Hub nodes become bottlenecks for neighbor forwarding
  - And situation worsens …

$$y = C\,x^{-a} : \log(y) = \log(C) - a\log(x)$$



Scale Free Networks. Albert Laszlo Barabasi and Eric Bonabeau. Scientific American. May-2003.

Gnutella : Random Walk-based Lookup

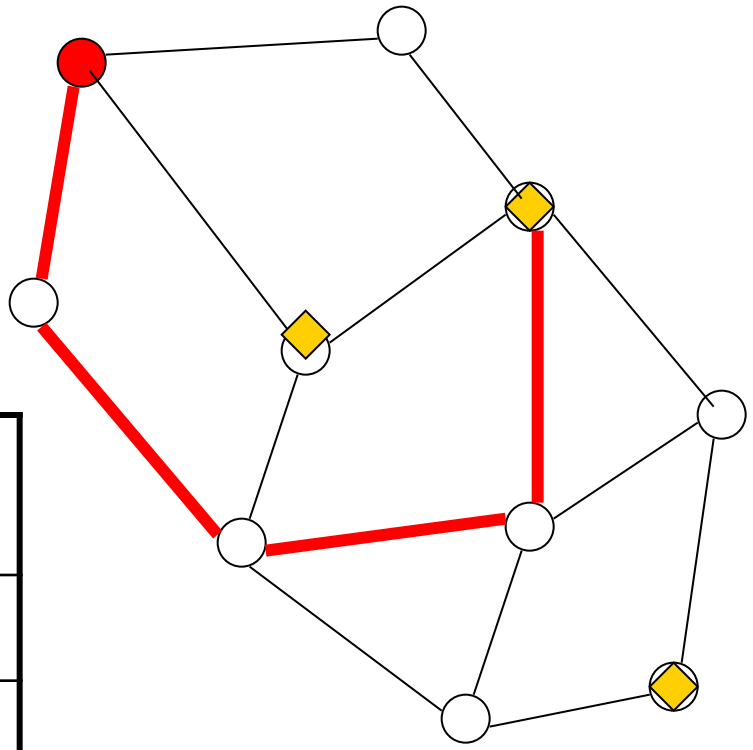


Simple analysis of Random Walk based Lookup

Let p = Population of the object. i.e. the fraction of nodes hosting the object (<1)

T = TTL (time to live)

Hop count **h**	Probability of success	Ex 1) popular	Ex 2) rare
1	p	0.3	0.0003
2	(1-p)p	0.21	0.00029
3	$(1-p)^2p$	0.147	0.00029
T	$(1-p)^{T-1}p$	….	….



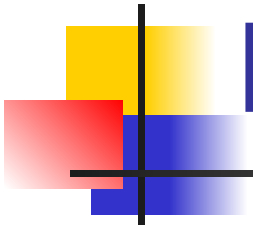
P = 3/10

Expected hop counts of the Random Walk based lookup

- Expected hop count E(h)
  $= 1p + 2(1-p)p + 3(1-p)^2p + \ldots + T(1-p)^{T-1}p$
  $= \mathbf{(1-(1-p)^T)/p - T(1-p)^T}$

- With a large TTL, E(h) = 1/p, which is intuitive.
  - If p is very small (rare objects), what happens?
- With a small TTL, there is a risk that search will time out before an existing object is located.

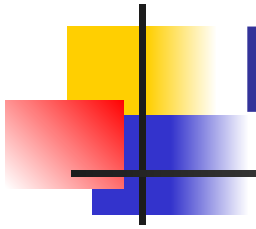
Extension of Random Walk based Lookup



- Multiple walkers
- Replication
- Biased Random Walk

Multiple Walkers

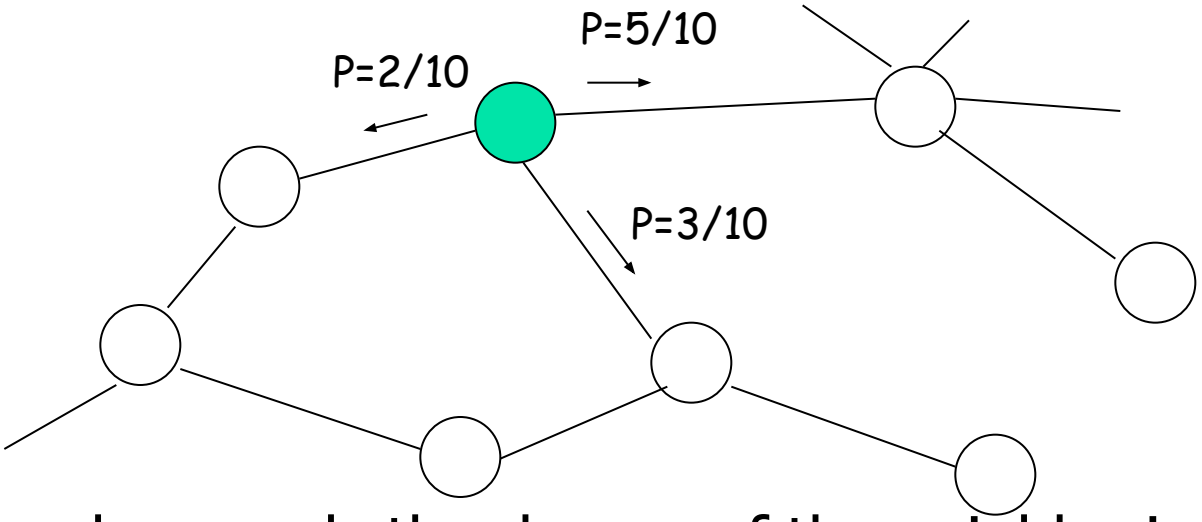
- Assume they all k walkers start in unison.
- Probability that none could find the object after one hop = $(1-p)^k$.
- The probability that **none succeeded after T hops = $(1-p)^{kT}$**.
- **So the probability that at least one walker succeeded is $1-(1-p)^{kT}$.**
  - A typical assumption is that the search is abandoned as soon as at least one walker succeeds
- As k increases, the overhead increases, but the delay decreases. There is a tradeoff.



Replication

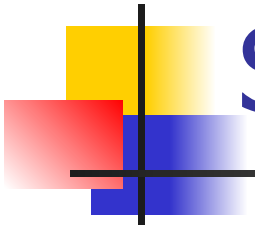
- One (Two or multiple) hop replication
  - Each node keeps track of the indices of the files belonging to its immediate (or multiple hop away) neighbors.
  - As a result, high capacity / high degree nodes can provide useful clues to a large number of search queries.

Biased Random Walk



- Each node records the degree of the neighboring nodes.
- Select highest degree node, that has not been visited
- This first climbs to highest degree node, then climbs down on the degree sequence
- Lookup **easily gravitates towards high degree nodes** that hold more clues.

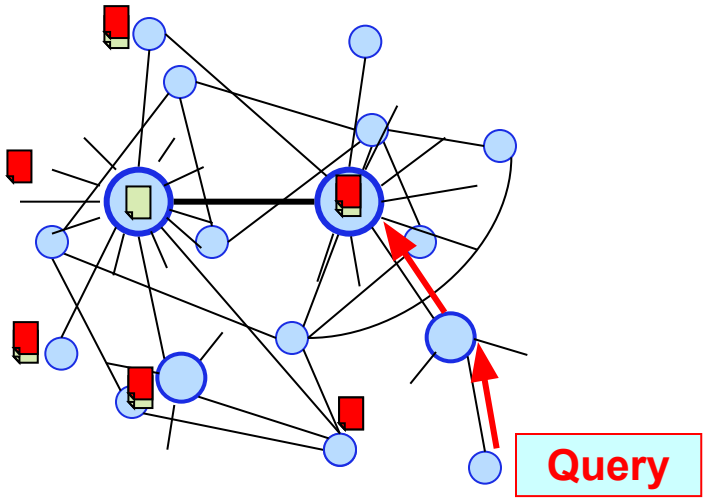
GIA : Making Gnutella-like P2P Systems Scalable

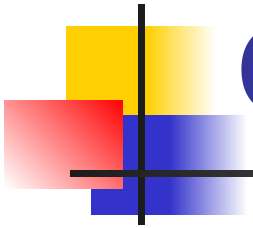


- GIA is short name of "gianduia"
- Unstructured, but take node capacity into account
  - High-capacity nodes have room for more queries: so, send most queries to them
- Will work only if high-capacity nodes:
  - Have correspondingly more answers, and
  - Are easily reachable from other nodes

GIA Design

- Make high-capacity nodes easily reachable
  - Dynamic topology adaptation converts them into high-degree nodes
- Make high-capacity nodes have more answers
  - One-hop replication
- Search efficiently
  - Biased random walks
- Prevent overloaded nodes
  - Active flow control





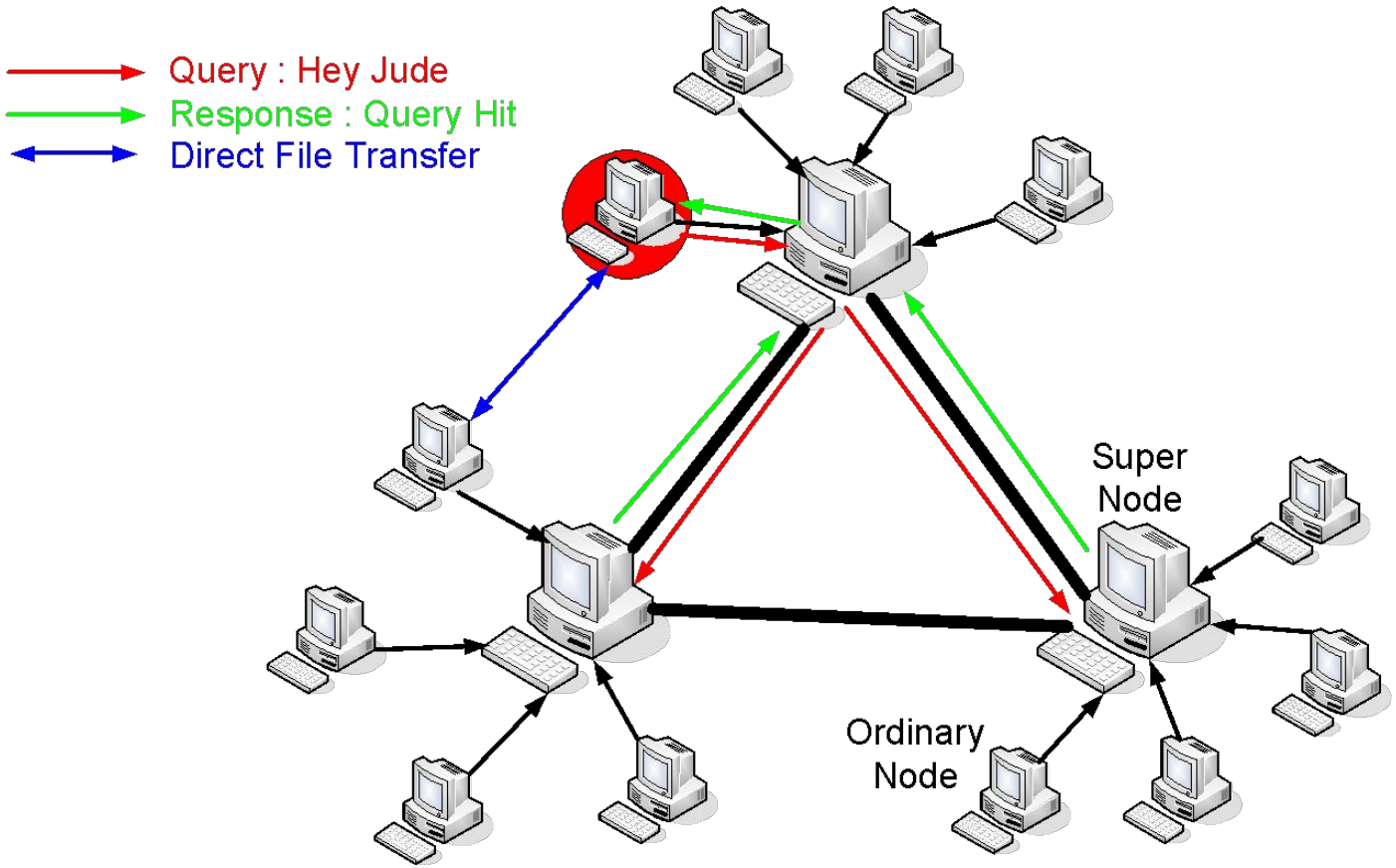
GIA : Active Flow Control

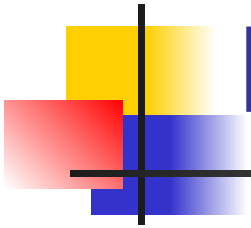
- Accept queries based on capacity
  - Actively allocation "tokens" to neighbors
  - Send query to neighbor only if we have received token from it
- Incentives for advertising true capacity
  - High capacity neighbors get more tokens to send outgoing queries
  - Allocate tokens with start-time fair queuing. Nodes not using their tokens are marked inactive and this capacity id redistributed among its neighbors.

KaZaA

- Created in March 2001
  - Uses proprietary FastTrack technology
- Combines strengths of Napster and Gnutella
- Based on "Supernode Architecture"
- Exploits heterogenity of peers
  - Two kinds of nodes
  - Super Node / Ordinary Node
- Organize peers into a hierarchy
  - Two-tier hierarchy

KaZaA architecture



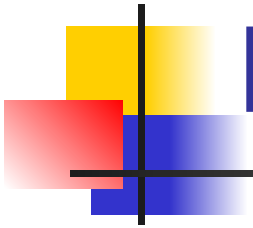


KaZaA : SuperNode

- Nodes that have more connection bandwidth and are more available are designated as supernodes
  - Each supernode manages around **100-150** children
  - Each supernode connects to **30-50** other supernodes

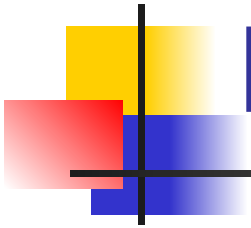
KaZaA : Overlay Maintenance

- New node goes through list until it finds operational supernode
  - Connects, obtains more up-to-date list, with 200 entries.
  - Gets Nodes in list are “close” to the new node.
  - The new node then pings 5 nodes on list and connects with the one
- If supernode goes down, a node obtains updated list and chooses new supernode



KaZaA : Metadata

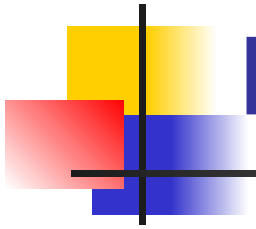
- Each supernode acts as a mini-Napster hub, tracking the content (files) and IP addresses of its descendants
  - For each file: File name, File size, Content Hash, File descriptors (used for keyword matches during query)
  - Content Hash:
    - When peer A selects file at peer B, peer A sends ContentHash in HTTP request
    - If download for a specific file fails (partially completes), ContentHash is used to search for new copy of file.



KaZaA : Operation

- Peer obtains address of an SN
  - e.g. via bootstrap server
- Peer sends request to SN and uploads metadata for files it is sharing
- The SN starts tracking this peer
  - Other SNs are not aware of this new peer
- Peer sends queries to its own SN
- SN answers on behalf of all its peers, forwards query to other SNs
- Other SNs reply for all their peers

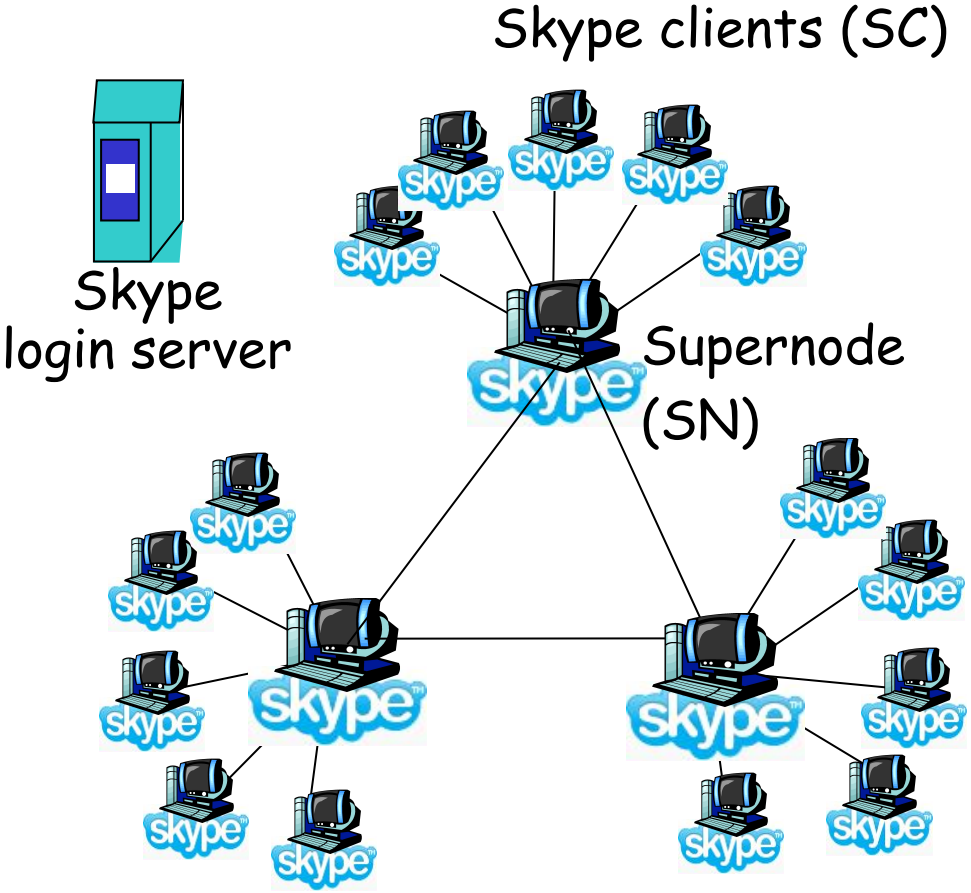
KaZaA : Parallel Downloading and Recovery



- If file is found in multiple nodes, user can select parallel downloading
  - Identical copies identified by ContentHash
- HTTP byte-range header used to request different portions of the file from different nodes
- Automatic recovery when server peer stops sending file
  - ContentHash

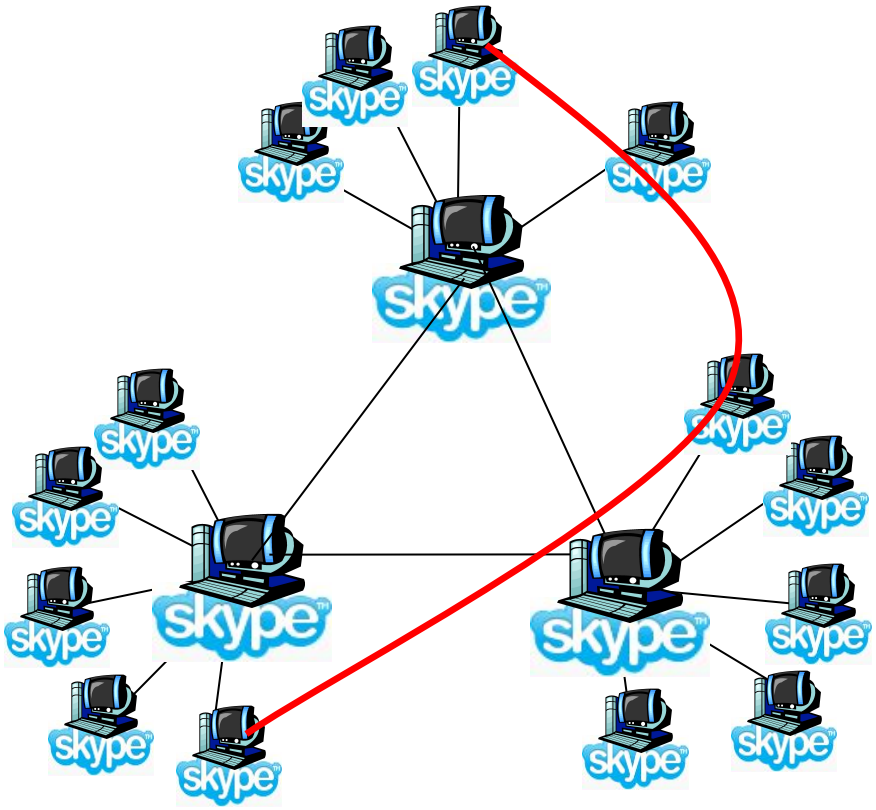
P2P Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SNs
- Index maps usernames to IP addresses; distributed over SNs



Peers as relays

- problem when both Alice and Bob are behind "NATs".
  - NAT prevents an outside peer from initiating a call to insider peer
- solution:
  - using Alice's and Bob's SNs, *relay* is chosen
  - each peer initiates session with relay.
  - peers can now communicate through NATs via relay



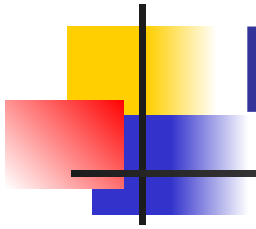
Unstructured vs Structured

- **Unstructured P2P networks** allow resources to be placed at any node. The network topology is arbitrary, and the growth is spontaneous.
- **Structured P2P networks** simplify resource location and load balancing by defining a topology and defining rules for resource placement.
  - Guarantee **efficient search for rare objects**

What are the rules???

Distributed Hash Table (DHT)

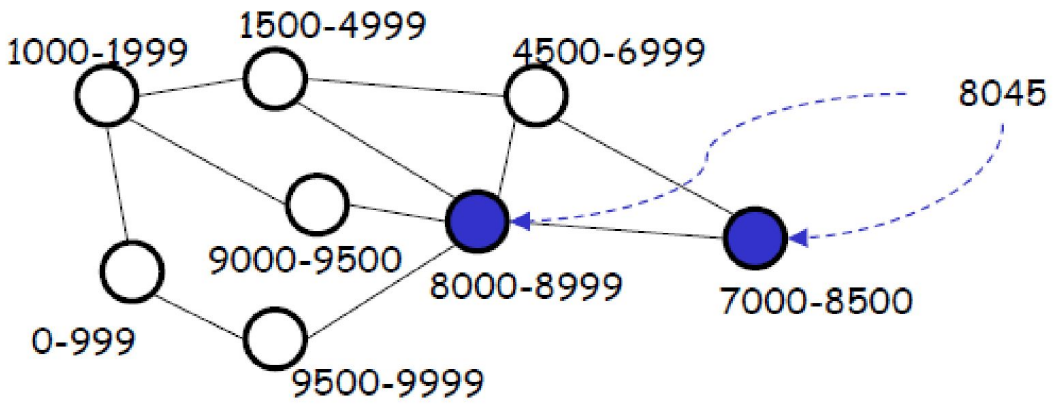
DHT overview: Directed Lookup



- Idea:
  - assign particular nodes to hold particular content (or pointers to it, like an information booth)
  - when a node wants that content, go to the node that is supposed to have or know about it
- Challenges:
  - Distributed: want to distribute responsibilities among existing nodes in the overlay
  - Adaptive: nodes join and leave the P2P overlay
    - distribute knowledge responsibility to joining nodes
    - redistribute responsibility knowledge from leaving nodes

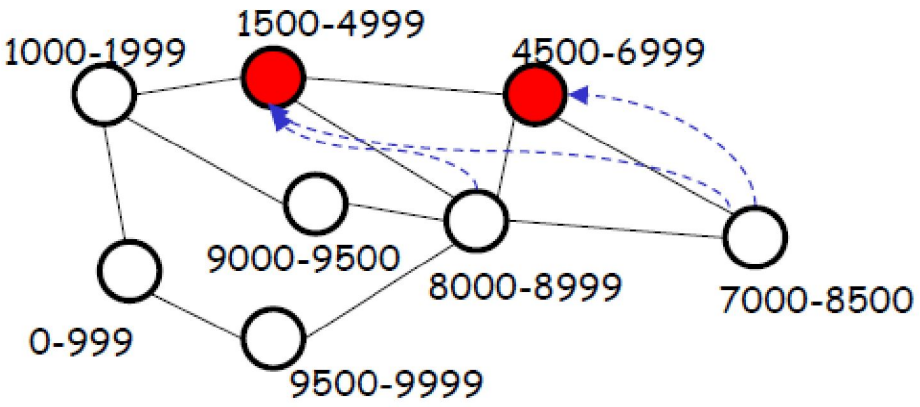
DHT overview: Hashing and mapping

- Introduce a hash function to map the object being searched for to a unique identifier:
  - e.g., h("Hey Jude") → 8045
- Distribute the range of the hash function among all nodes in the network
- Each node must "know about" at least one copy of each object that hashes within its range (when one exists)

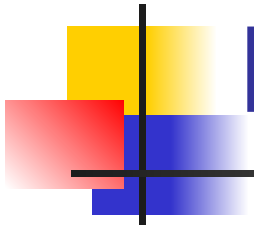


DHT overview: Knowing about objects

- Two alternatives
  - Node can cache each (existing) object that hashes within its range
  - Pointer-based: level of indirection – node caches pointer to location(s) of object

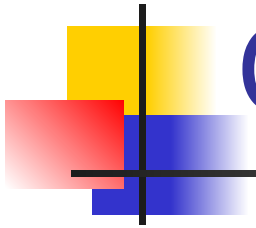


DHT overview: Routing



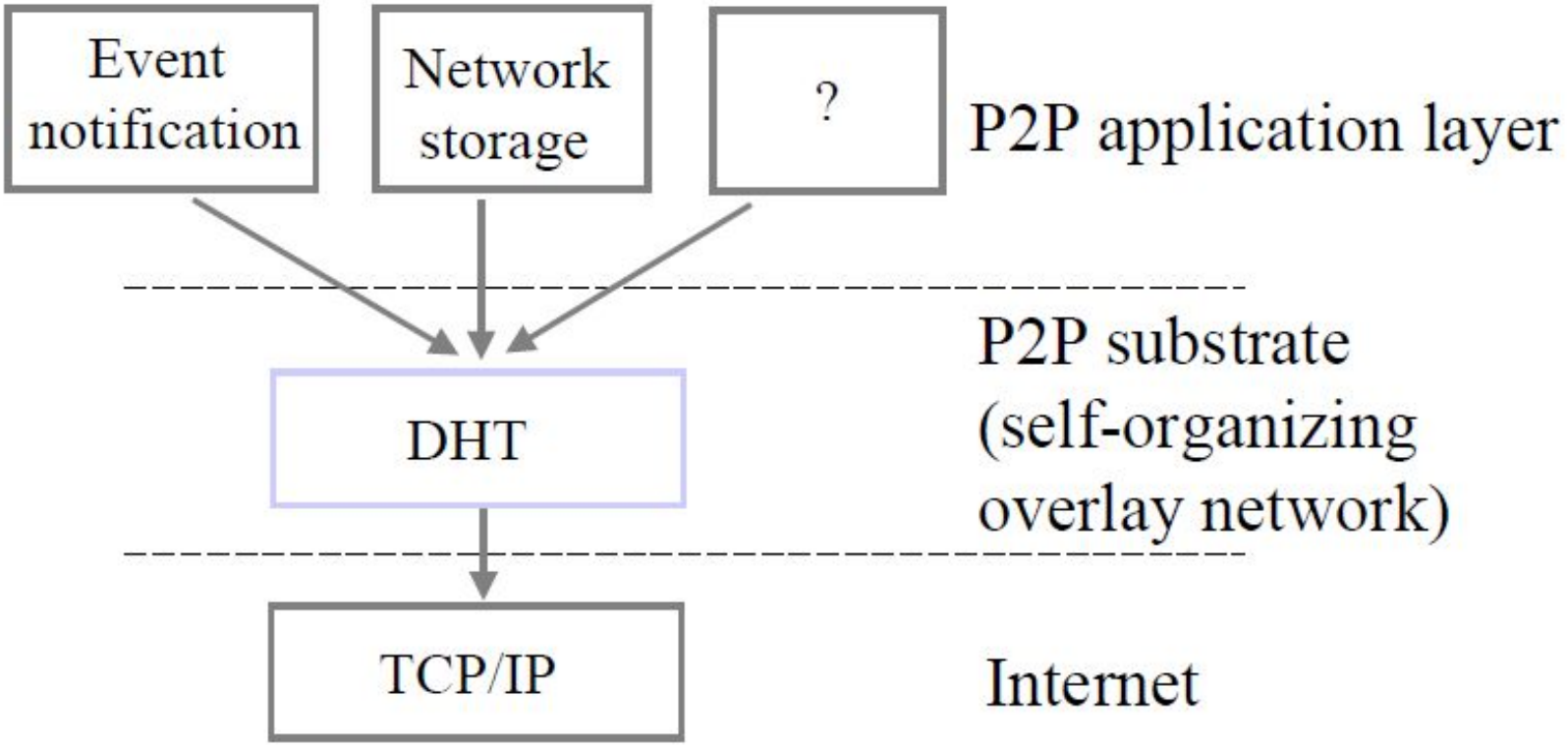
- For each object, node(s) whose range(s) cover that object must be reachable via a "short" path
  - by the querier node (assumed can be chosen arbitrarily)
  - by nodes that have copies of the object (when pointer-based approach is used)
- The different approaches (CAN, Chord, Pastry, Tapestry) differ fundamentally only in the routing approach
  - any "good" random hash function will suffice

DHT overview: Other Challenges



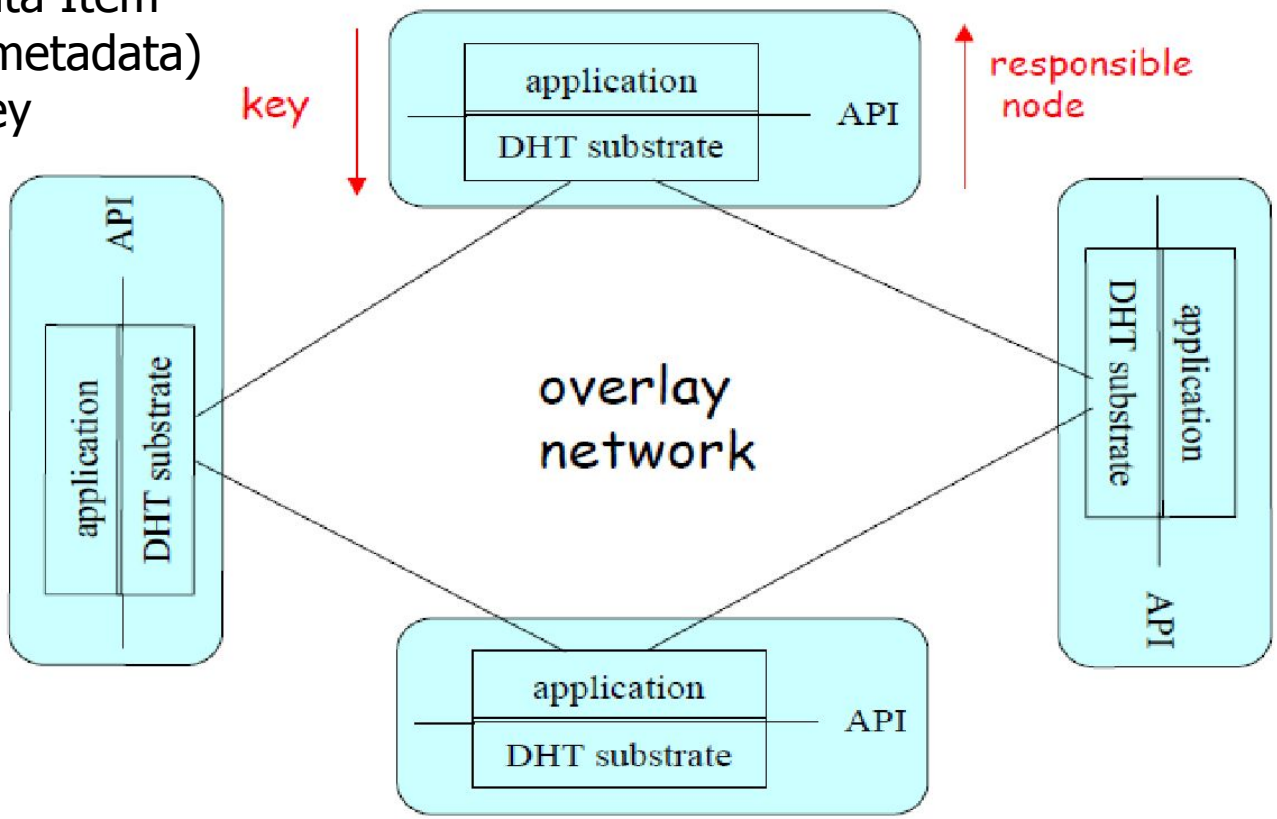
- # neighbors for each node should scale with growth in overlay participation (e.g., should not be O(N))
- DHT mechanism should be fully distributed (no centralized point that bottlenecks throughput or can act as single point of failure)
- DHT mechanism should gracefully handle nodes joining/leaving the overlay
  - need to repartition the range space over existing nodes
  - need to reorganize neighbor set
  - need bootstrap mechanism to connect new nodes into the existing DHT infrastructure

DHT overview: DHT Layered Architecture



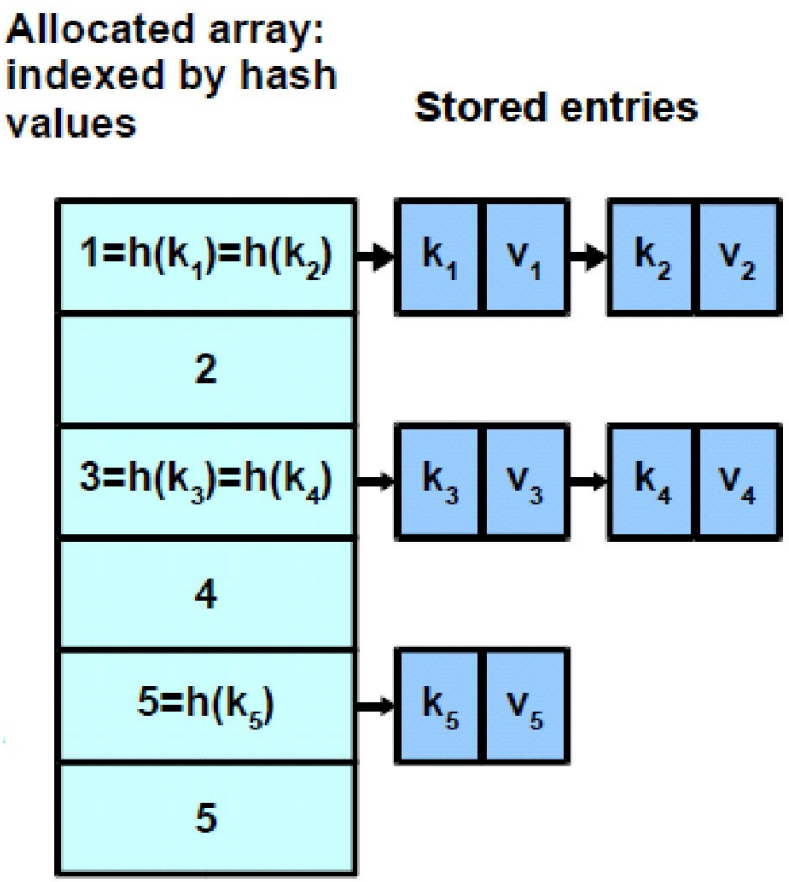
DHT overview: DHT based Overlay

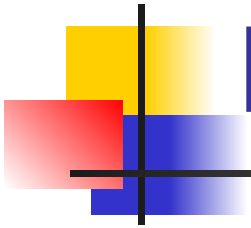
Each Data Item (file or metadata) has a key



Hash Tables

- Store arbitrary keys and satellite data (value)
  - put(key,value)
  - value = get(key)
- Lookup must be fast
  - Calculate hash function h() on key that returns a storage cell
  - Chained hash table: Store key (and optional value) there

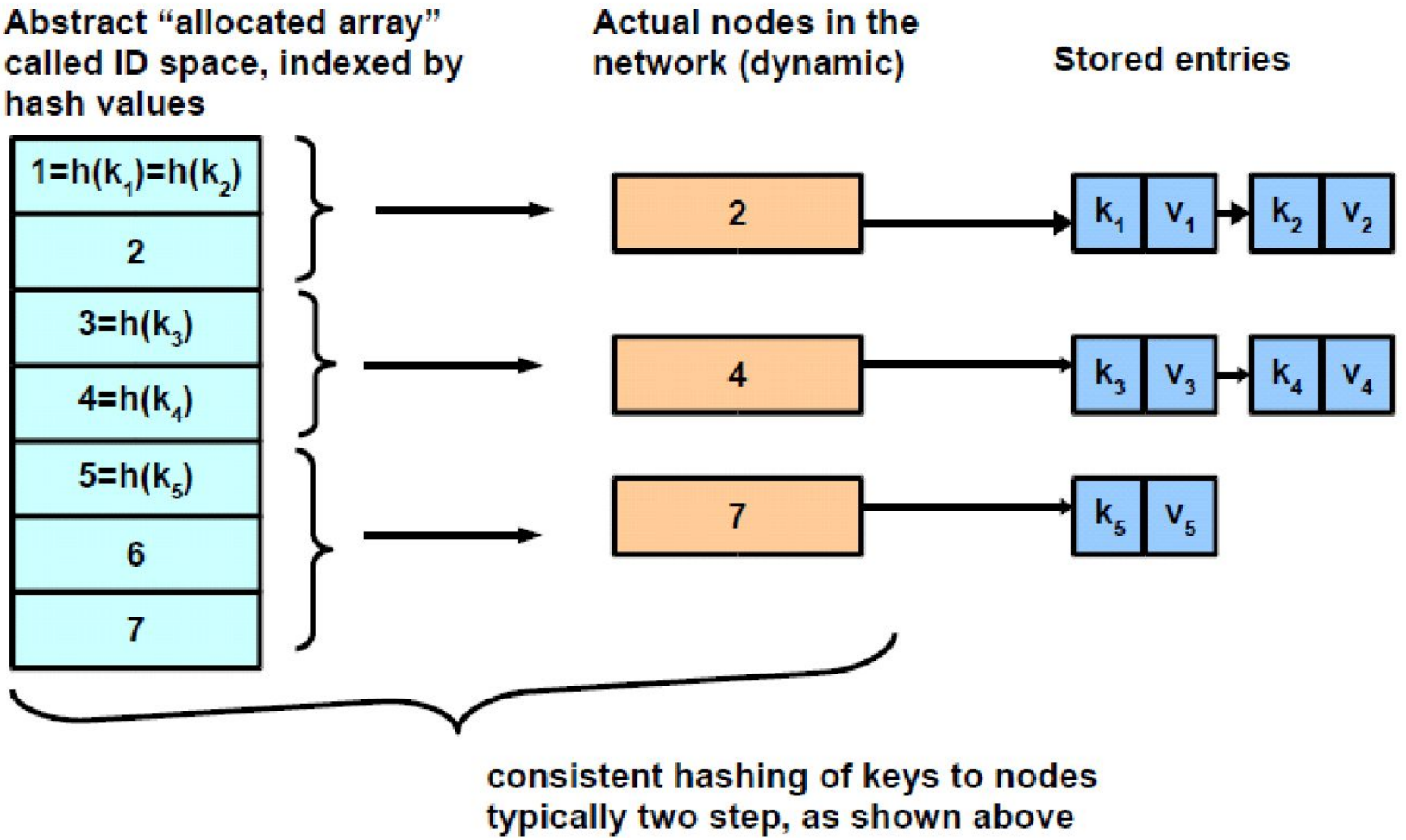




Distributed Hash Table

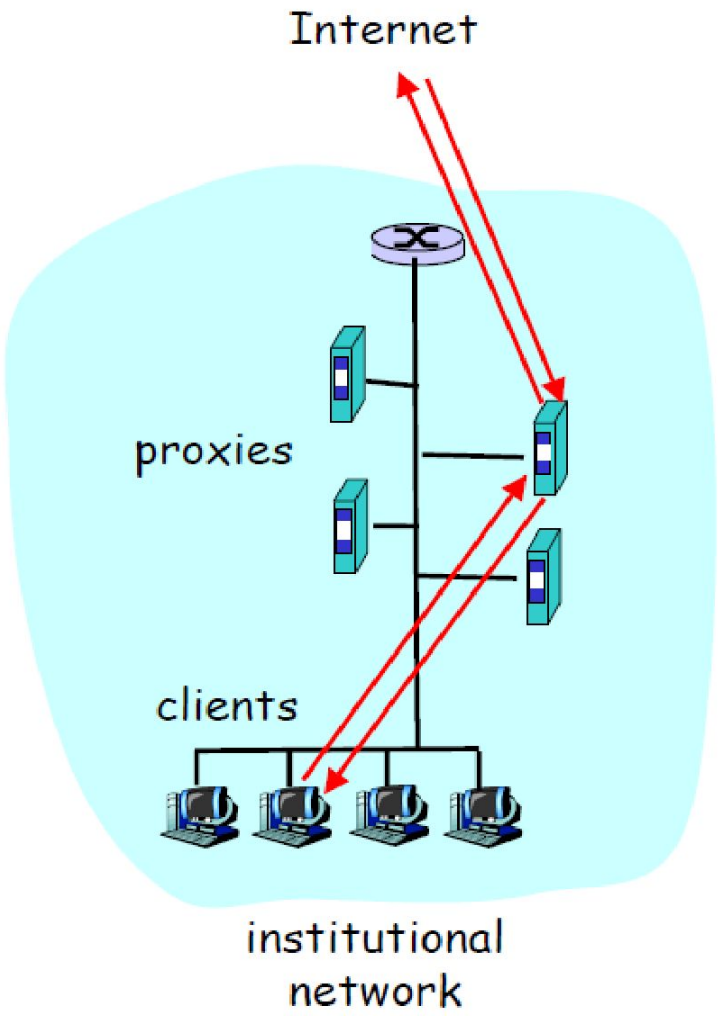
- Hash table functionality in a P2P network : lookup of data indexed by keys
  - Distributed P2P database
  - database has (key, value) pairs;
    - key: ss number; value: human name
    - key: content type; value: IP address
  - peers query DB with key
    - DB returns values that match the key
  - peers can also insert (key, value) peers
- Key-hash ☐ node mapping
  - Assign a unique live node to a key
  - Find this node in the overlay network quickly and cheaply

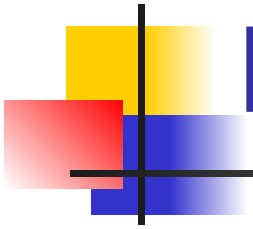
Distributed Hash Table



Old version of Distributed Hash Table : CARP

- 1997~
- Each proxy has unique name (proxy_n)
- Value=URL=u
- Get h(proxy_n,u) for all proxies as a key
- Assign u to proxy with highest h(proxy_n, u)





Problem of CARP

- Not good for P2P:
  - Each node needs to know name of all other up nodes
    - i.e., need to know O(N) neighbors
  - Hard to handle dynamic behavior of nodes (join/leave)
- But only O(1) hops in lookup

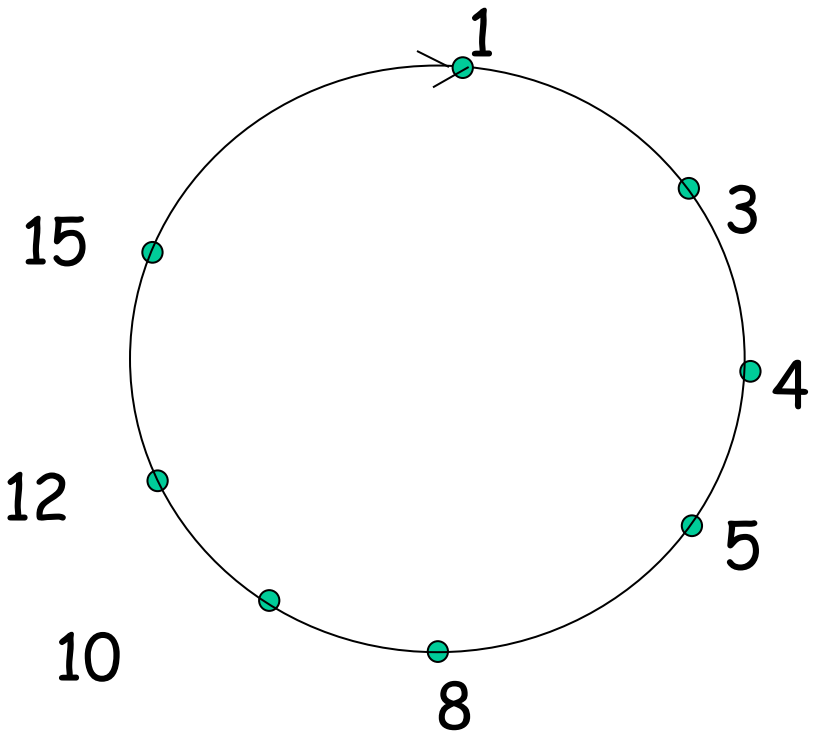
New concept of DHT: Consistent Hashing

- Node Identifier
  - assign integer identifier to each peer in range $[0,2^n-1]$.
    - Each identifier can be represented by n bits.
- Key : Data Identifier
  - require each key to be an integer in same range.
  - to get integer keys, hash original value.
    - e.g., key = h("Hey Jude.mp3"),
- Both node and data are placed in a same ID space ranged in $[0,2^n-1]$.

Consistent Hashing : How to assign key to node?

- central issue:
  - assigning (key, value) pairs to peers.
- rule: assign key to the peer that has the closest ID.
  - E.g. Chord: closest is the immediate successor of the key.
  - E.g. CAN : closest is the node whose responsible dimension includes the key.
- e.g.,: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

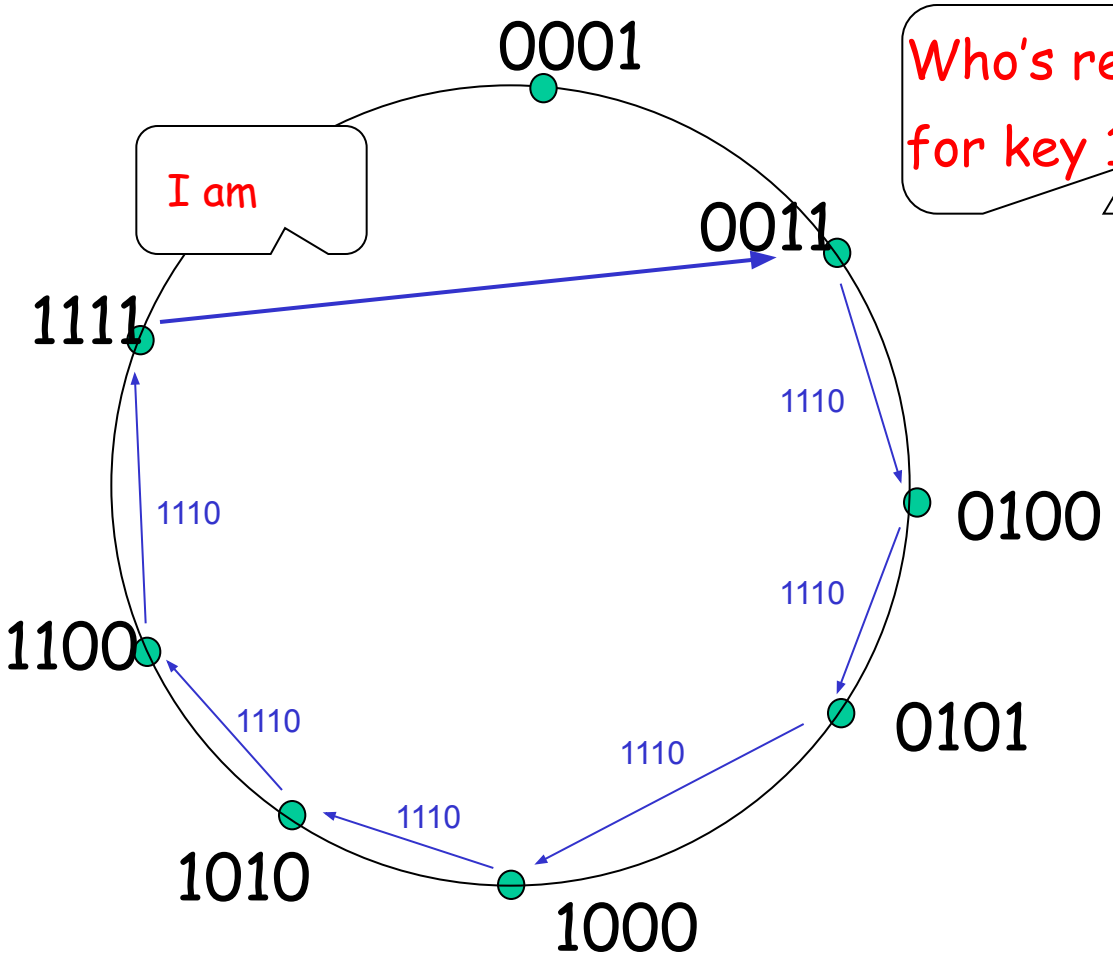
Circular DHT (1)



- each peer *only* aware of immediate successor and predecessor.
- Circular "overlay network"

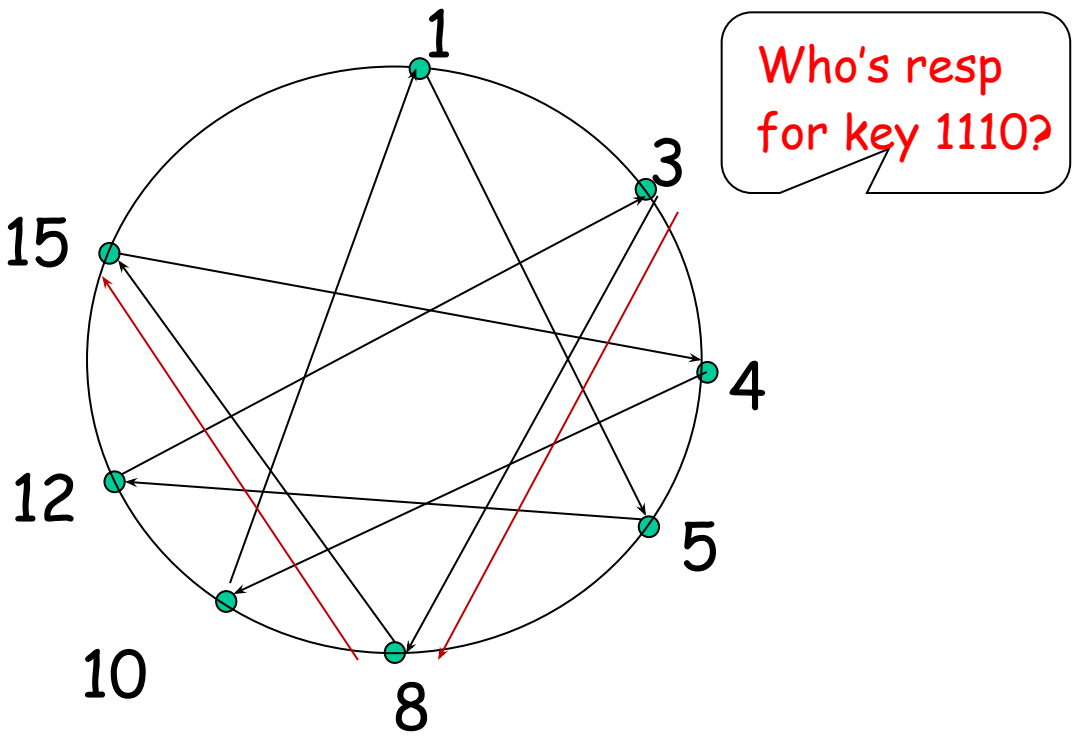
Circular DHT : simple routing

O(N) messages on avg to resolve query, when there are N peers



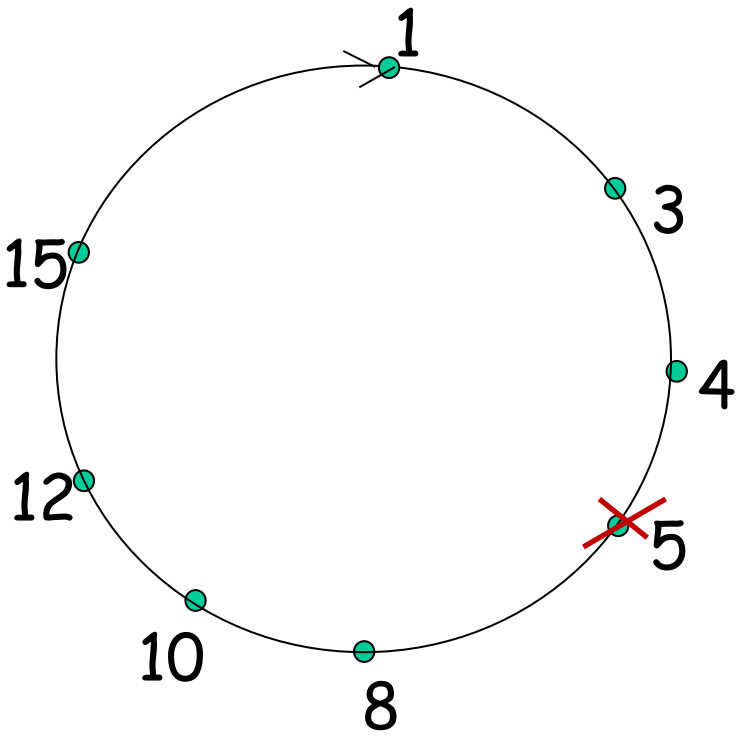
Define closest as closest successor

Circular DHT with Shortcuts



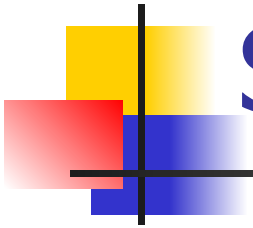
- each peer keeps track of IP addresses of predecessor, successor, short cuts.
- reduced from 6 to 2 messages.
- possible to design shortcuts so O(log N) neighbors, O(log N) messages in query

Peer Churn



- To handle peer churn, require each peer to know the IP address of its two successors.
- Each peer periodically pings its two successors to see if they are still alive.

- peer 5 abruptly leaves
- Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
- What if 5 and 8 leaves simultaneously?

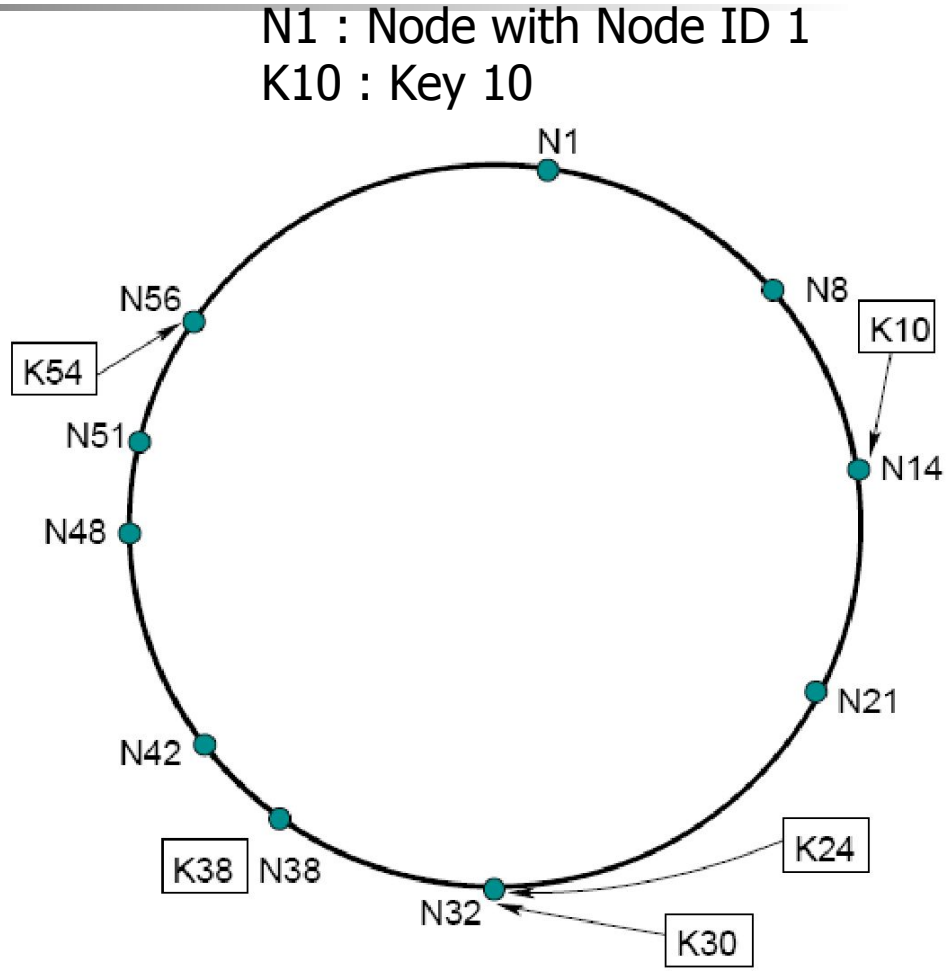


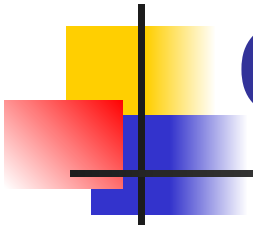
Structured P2P Systems

- Chord
  - Consistent hashing based ring structure
- Pastry
  - Uses ID space concept similar to Chord
  - Exploits concept of a nested group
- CAN
  - Nodes/objects are mapped into a d-dimensional Cartesian space
- Kademlia
  - Similar structure to Pastry, but the method to check the closeness is XOR function

Chord

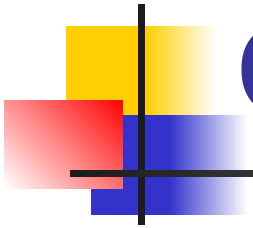
- Consistent hashing based on an ordered ring overlay
- Both keys and nodes are hashed to 160 bit IDs (SHA-1)
- Then keys are assigned to nodes using consistent hashing
  - Successor in ID space





Chord : hashing properties

- Uniformly Randomized
  - All nodes receive roughly equal share of load
  - As the number of nodes increases, the share of each node becomes more fair.
- Local
  - Adding or removing a node involves an O(1/N) fraction of the keys getting new locations

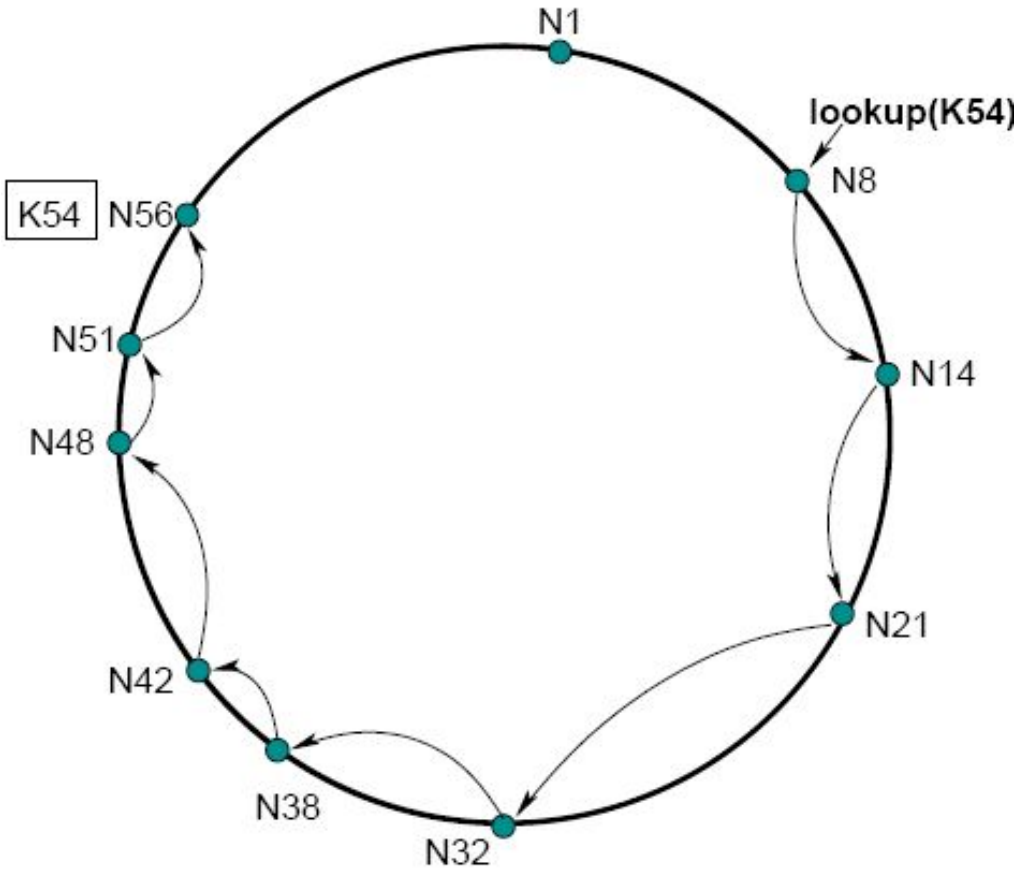


Chord : Lookup operation

- Searches the node that stores the key ({key, value} pair)
- Two protocols
  - Simple key lookup
    - Guaranteed way
  - Scalable key lookup
    - Efficient way

Chord : Simple Lookup

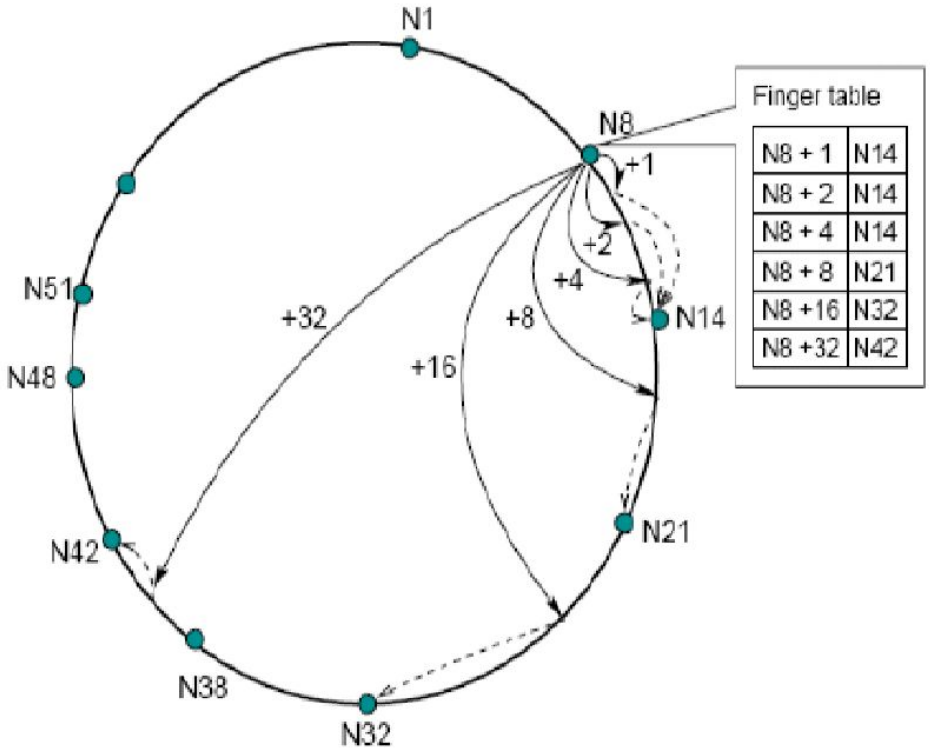
- Lookup query is forwarded to successor.
  - one way
- Forward the query around the circle
- In the worst case, O(N) forwarding is required
  - In two ways, O(N/2)

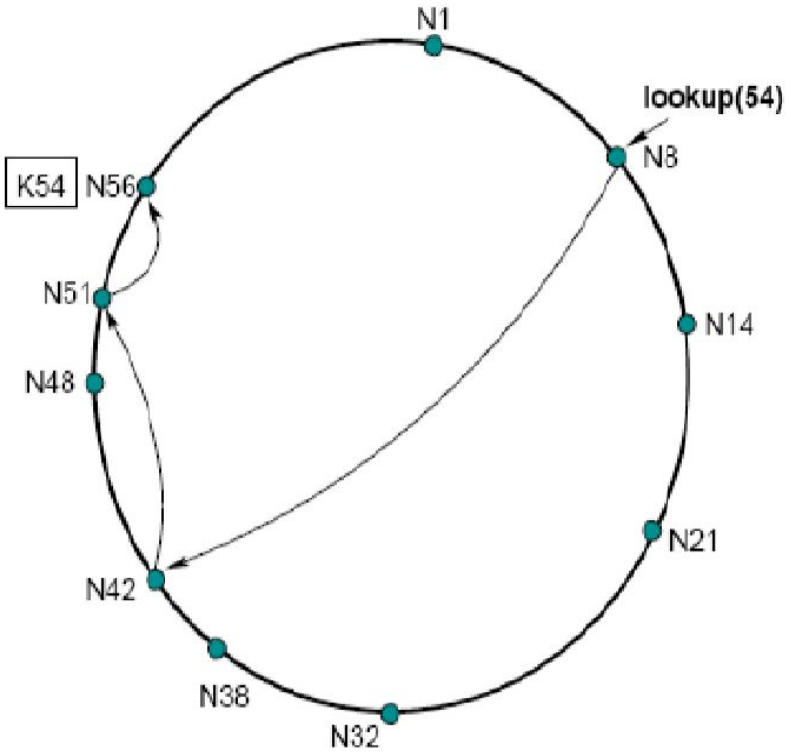


Chord : Scalable Lookup

- Each node **n** maintains a routing table with up to **m** entries (called the **finger table**)
- The **i^{th}** entry in the table is the location of the successor (**$n + 2^{i-1}$**)
- Query for a given identifier (**key**) is forwarded to the nearest node among **m** entries at each node. (node that most immediately precedes **key**)
- Search cost = O (log **N**) (m=O(log N))

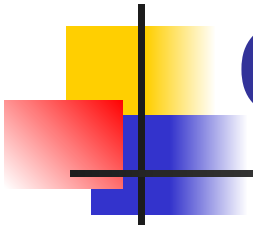
Chord : Scalable Lookup





i_{th} entry of a finger table points the successor of the key (nodeID + 2^{i-1})

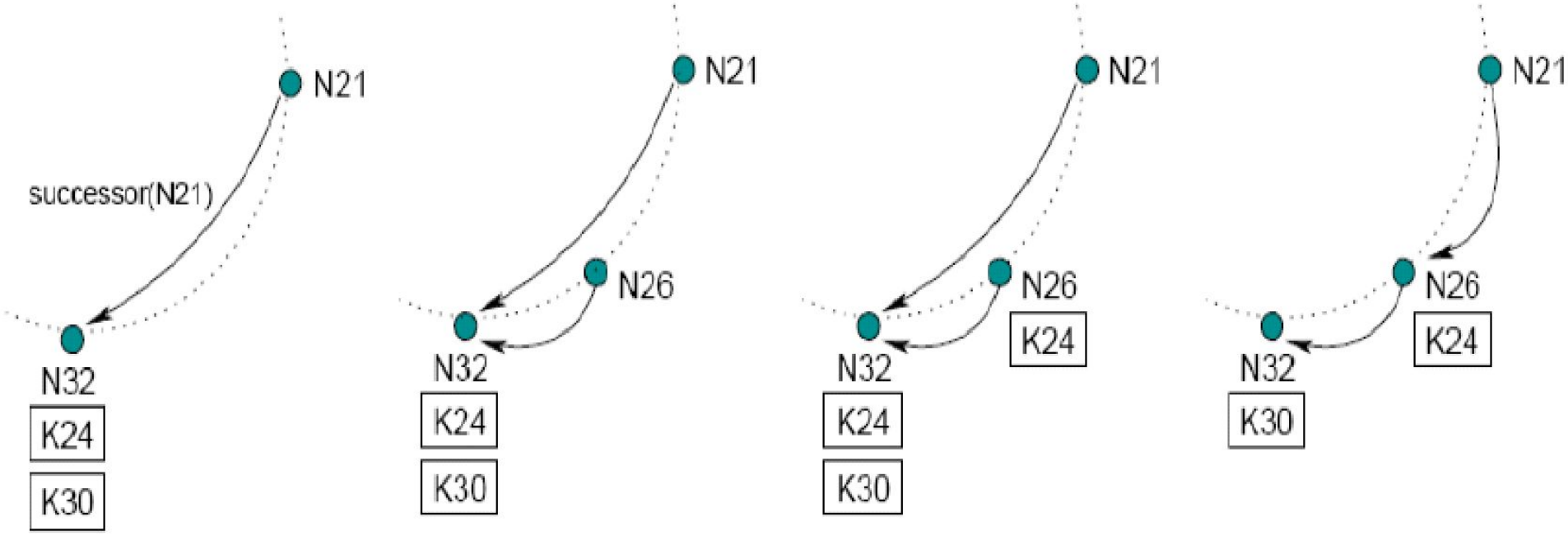
A finger table has O(log N) entries and the scalable lookup is bounded to O(log N)



Chord : Node Join

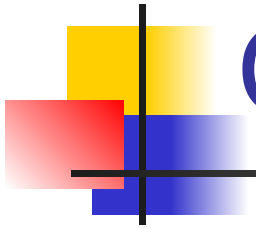
- New node **N** identifies its successor
  - Performs lookup (**N**)
- Takes over all successor's keys that the new node is responsible for
- Sets its predecessor to its successor's former predecessor
- Sets its successor's predecessor to itself
- Newly joining node builds a finger table
  - Performs lookup ($\mathbf{N + 2^{i-1}}$) (for i=0, 1, 2, …**I**)
  - **I**= number of finger print entries
- Update other nodes' finger tables

Chord : Node join example

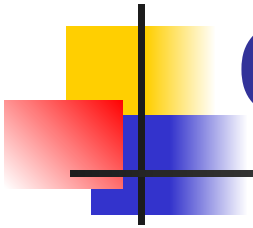


When a node joins/leaves the overlay, O(K/N) objects moves between nodes.

Chord : Node Leave



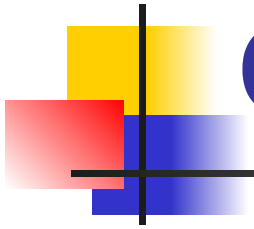
- Similar to Node Join
- Moves all keys that the node is responsible for to its successor
- Sets its successor's predecessor to its predecessor
- Sets its predecessor's successor to its successor
  - C.f. management of a linked list
- Finger Table??
  - There is no explicit way to update others' finger tables which point the leaving node



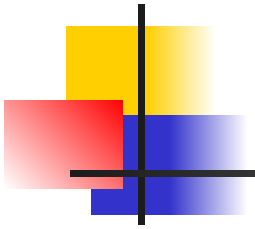
Chord : Stabilization

- If the ring is correct, then routing is correct, fingers are needed for the speed only
- Stabilization
  - Each node periodically runs the stabilization routine
  - Each node refreshes all fingers by periodically calling find_successor($n+2^i-1$) for a random i
  - Periodic cost is O(logN) per node due to finger refresh

Chord : Failure handling



- Failed nodes are handled by
  - Replication: instead of one successor, we keep r successors
    - More robust to node failure (we can find our new successor if the old one failed)
  - Alternate paths while routing
    - If a finger does not respond, take the previous finger, or the replicas, if close enough
- At the DHT level, we can replicate keys on the r successor nodes
  - The stored data becomes equally more robust

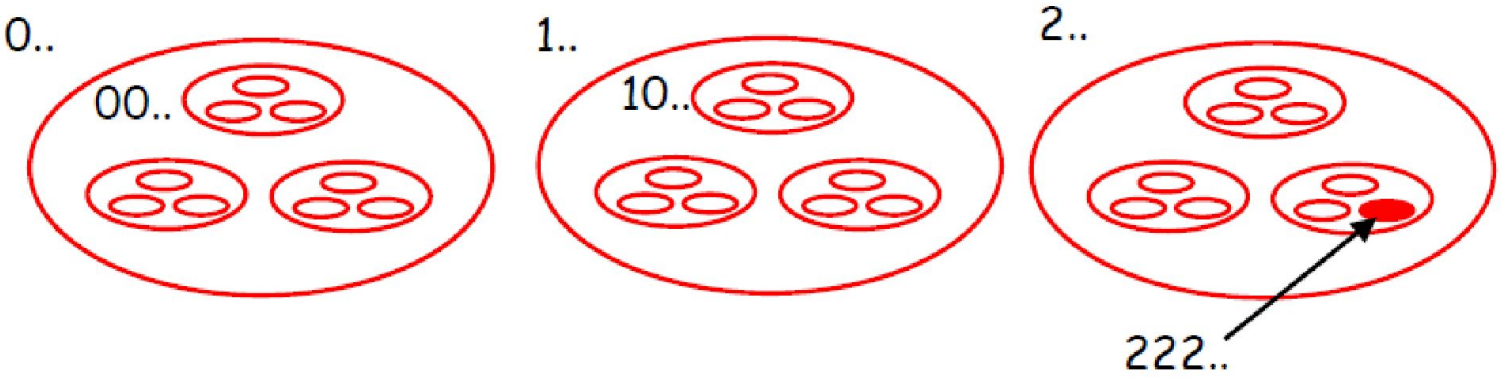


Pastry : Identifiers

- Applies a sorted ring in ID space like Chord
  - Nodes and objects are assigned a 128-bit identifier
- NodeID (and key) is interpreted as sequences of digit with base 2^b
  - In practice, the identifier is viewed in base 16 (b=4).
- The node that is responsible for a key is numerically closest (not the successor)
  - Bidirectional and using numerical distance

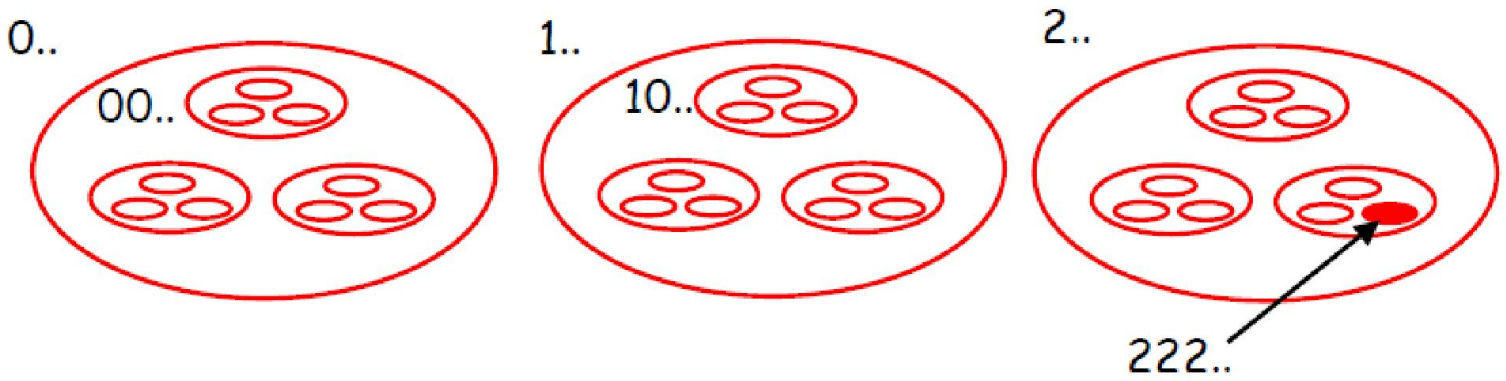
Pastry : ID space

- Simple example: nodes & keys have n-digit **base-3** ids, eg, 02112100101022
  - There are **3** nested groups for each group
- Each key is stored in a node with closest node ID
- Node addressing defines nested groups

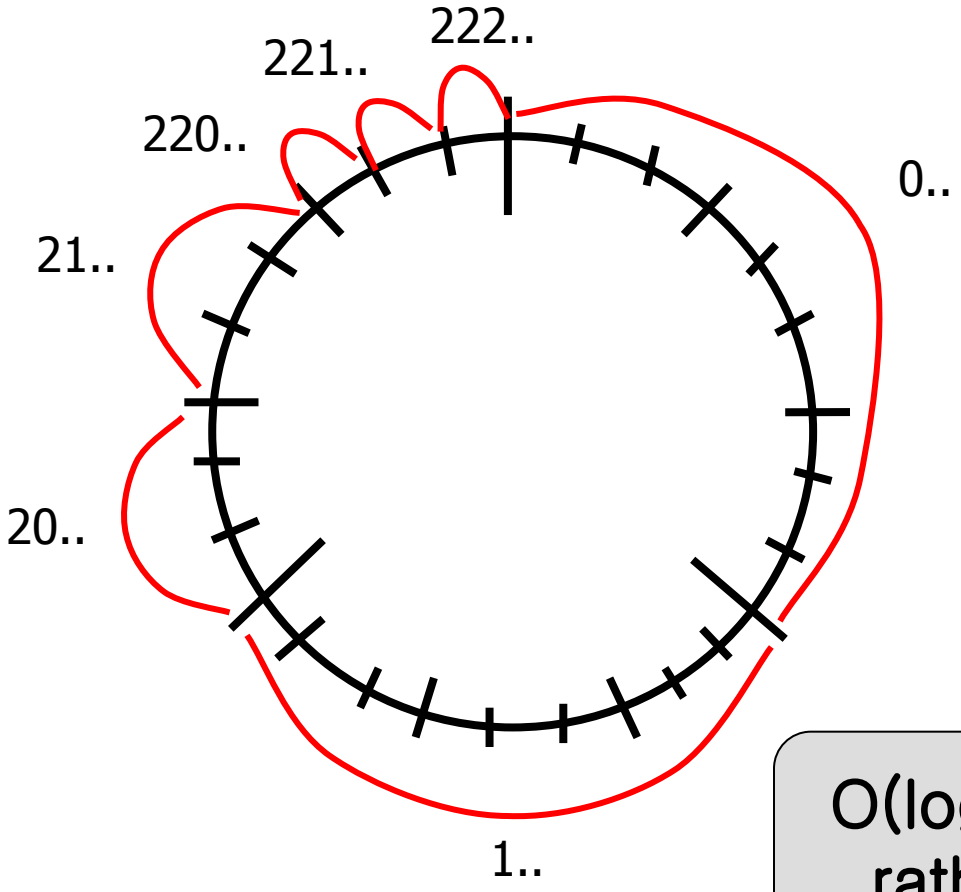


Pastry : Nested Group

- Nodes in same inner group know each other's IP address
- Each node knows IP address of one delegate node in some of the other groups
  - Which?
  - Node in 222…: 0…, 1…, 20…, 21…, 220…, 221…
  - 6 delegate nodes rather than 27



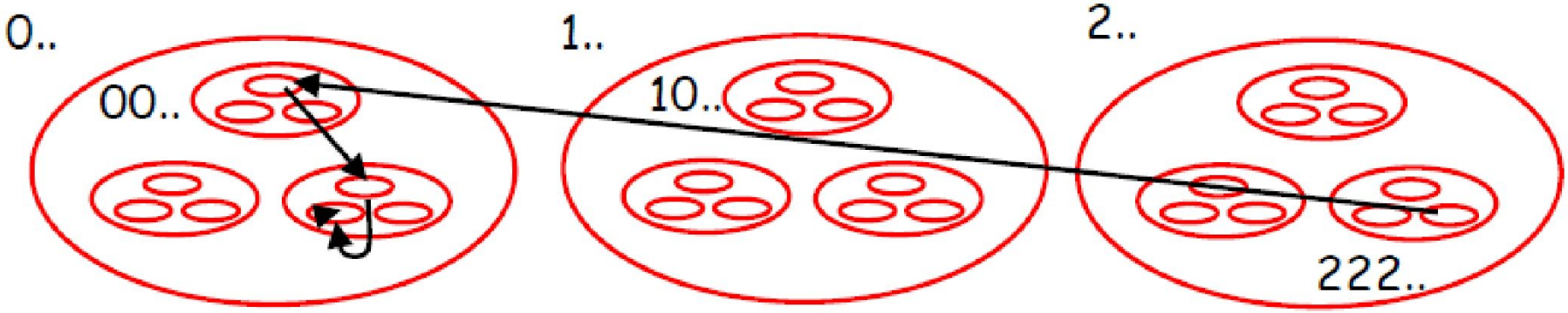
Pastry : Ring View



O(log N) delegates rather than O(N)

Pastry : Lookup in nested group

- Divide and conquer
- Suppose node in group 222… wants to lookup key k= 02112100210.
- Forward query to node node in 0…, then to node in 02…, then to node in 021…
- Node in 021… forwards to closest to key in 1 hop



Pastry : Routing table

Base-4 routing table

- Routing table
  - Provides delegate nodes in nested groups
  - Self-delegate for the nested group where the node is belong to
  - $O(\log_b N)$ rows
    - ☐ $O(\log_b N)$ lookup

NodeId 10233102

Leaf set	SMALLER	LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Pastry : Leaf set

- Leaf set
  - Set of nodes which is numerically closest to the node
    - L/2 smaller & L/2 higher
  - Periodically update
  - Support reliability and consistency
    - Cf) Successors in Chord
  - Replication boundary
  - Stop condition for lookup

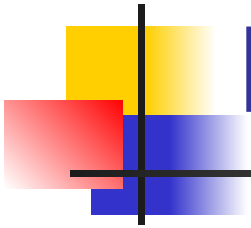
Base-4 routing table

NodeId 10233102

Leaf set	SMALLER	LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

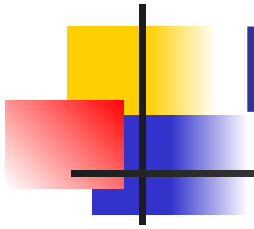
Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	



Pastry : Lookup Process

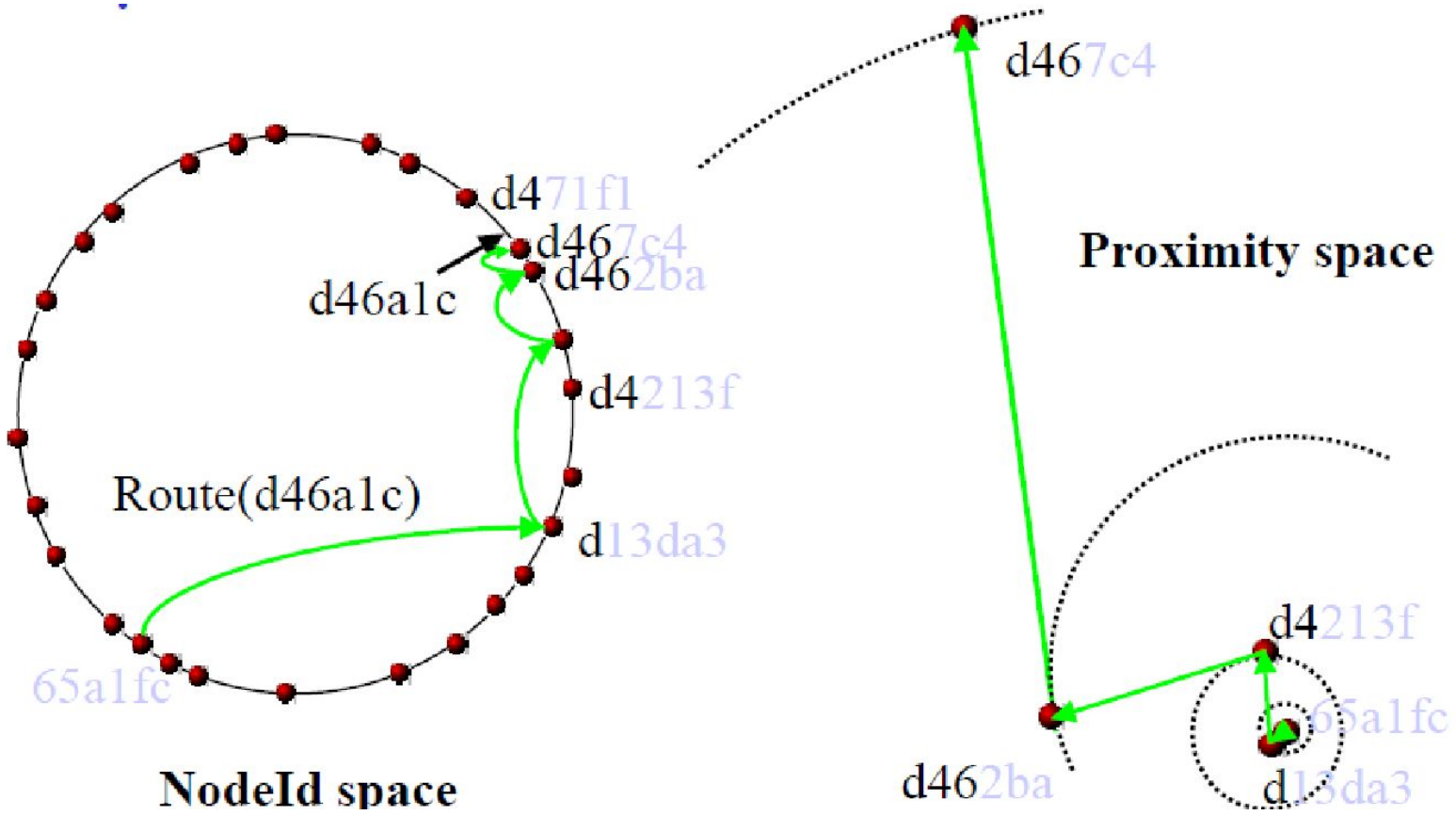
- **if** (destination is within range of our leaf set)
  - forward to numerically closest member
- **else**
  - **if** (there's a longer prefix match in table)
    - forward to node with longest match
  - **else**
    - forward to node in table
    - (a) shares at least as long a prefix
    - (b) is numerically closer than this node

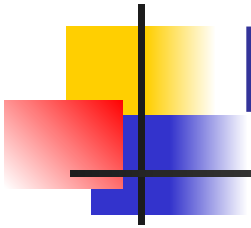


Pastry : Proximity routing

- **Assumption:** scalar proximity metric
  - e.g. ping delay, # IP hops
  - a node can probe distance to any other node
- **Proximity invariant:**
  - **Each routing table entry refers to a node close to the local node (in the proximity space), among all nodes with the appropriate nodeId prefix.**

Pastry : Routing in Proximity Space

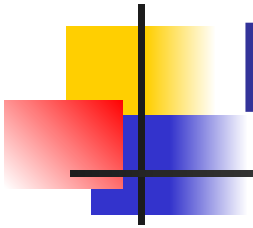




Pastry : Join and Failure

- Join
  - Finds numerically closest node already in network
  - Ask state from all nodes on the route and initialize own state
    - LeafSet and Routing Table
- Failure Handling
  - Failed leaf node: contact a leaf node on the side of the failed node and add appropriate new neighbor
  - Failed table entry: contact a live entry with same prefix as failed entry until new live entry found, if none found, keep trying with longer prefix table entries

Summary : Structured DHT based P2P



- Design issues
  - ID (node, key) mapping
  - Routing (Lookup) method
  - Maintenance (Join/Leave) method
  - All functionality should be fully distributed

Summary : Unstructured vs Structured

	Query Lookup	Overlay Network Management
Unstructured	Flood-based (heavy overhead)	Simple
Structured	Bounded and effective, O(log N)	Complex (heavy overhead)

Summary of Consistent Hashing

- Consistent hashing
  - Elegant way to divide a workload across machines
  - Very useful in clusters:
- Replication for high availability, efficient recovery after node failure
- Incremental scalability: “add nodes, capacity increases”
- Self-management: minimal configuration
- Unique trait: no single server to shut down/monitor

Couchbase automated data partitioning [5]

OpenStack's Object Storage Service - Swift[6]

Partitioning in Amazon's storage system Dynamo[7]

Data partitioning in Apache Cassandra[8]

Data partitioning in Voldemort[9]

Akka's consistent hashing router[10]

Riak, a distributed key-value database[11]

Gluster, a network-attached storage file system[12]

Akamai content delivery network[13]

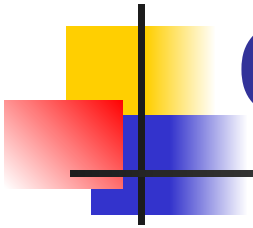
Discord chat application[14]

Maglev network load balancer[15]

Data partitioning in Azure Cosmos DB

Actively used today in many key-value stores

P2P Content Dissemination

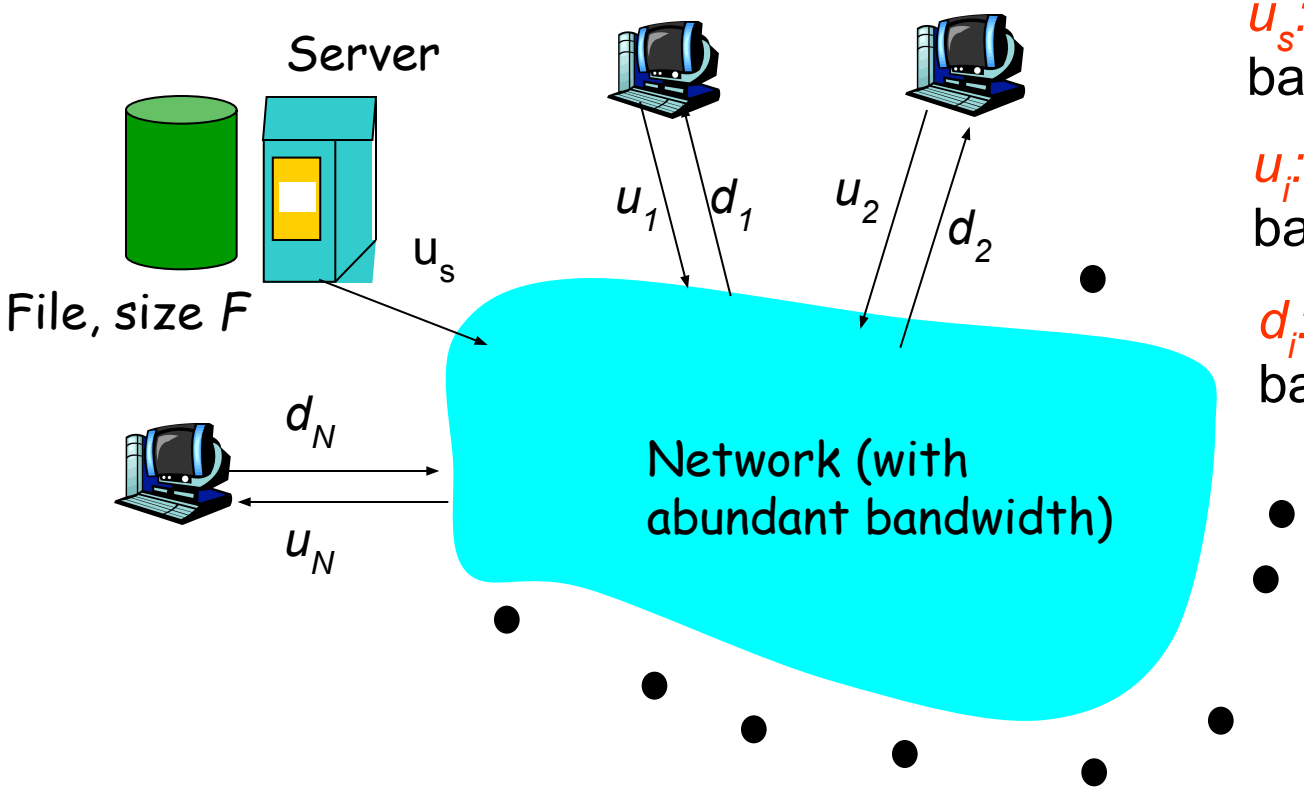


Content dissemination

- Content dissemination is about allowing clients to actually get a file or other data after it has been located
- Important parameters
  - Throughput
  - Latency
  - Reliability

File Distribution: Server-Client vs P2P

Question : How much time to distribute a file from one server to *N peers*?



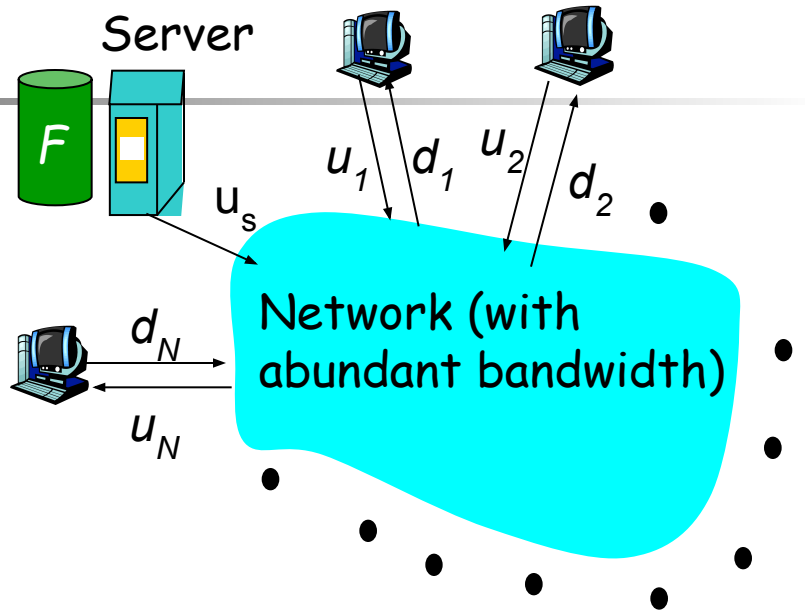
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download

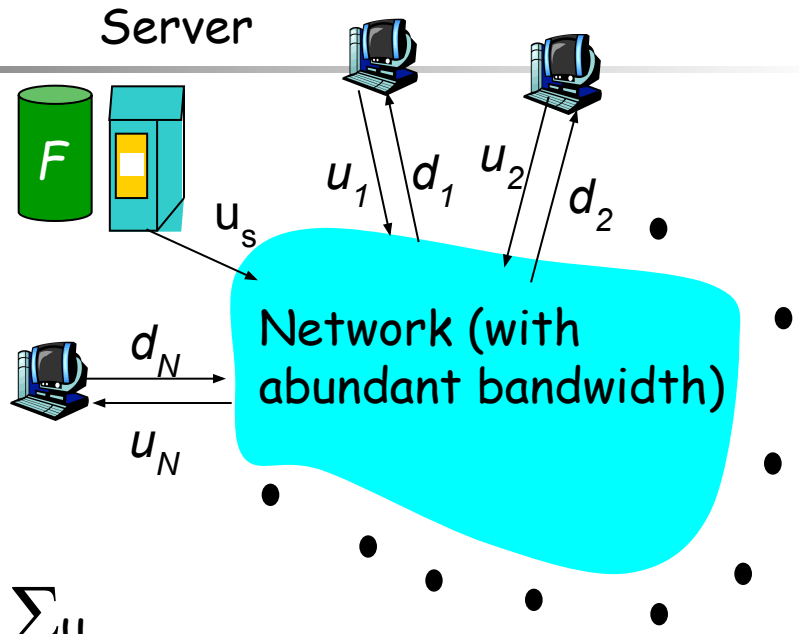


Time to distribute F to N clients using client/server approach $= d_{cs} = \max_i \{ NF/u_s, F/\min(d_i) \}$

increases linearly in N (for large N)

File distribution time: P2P

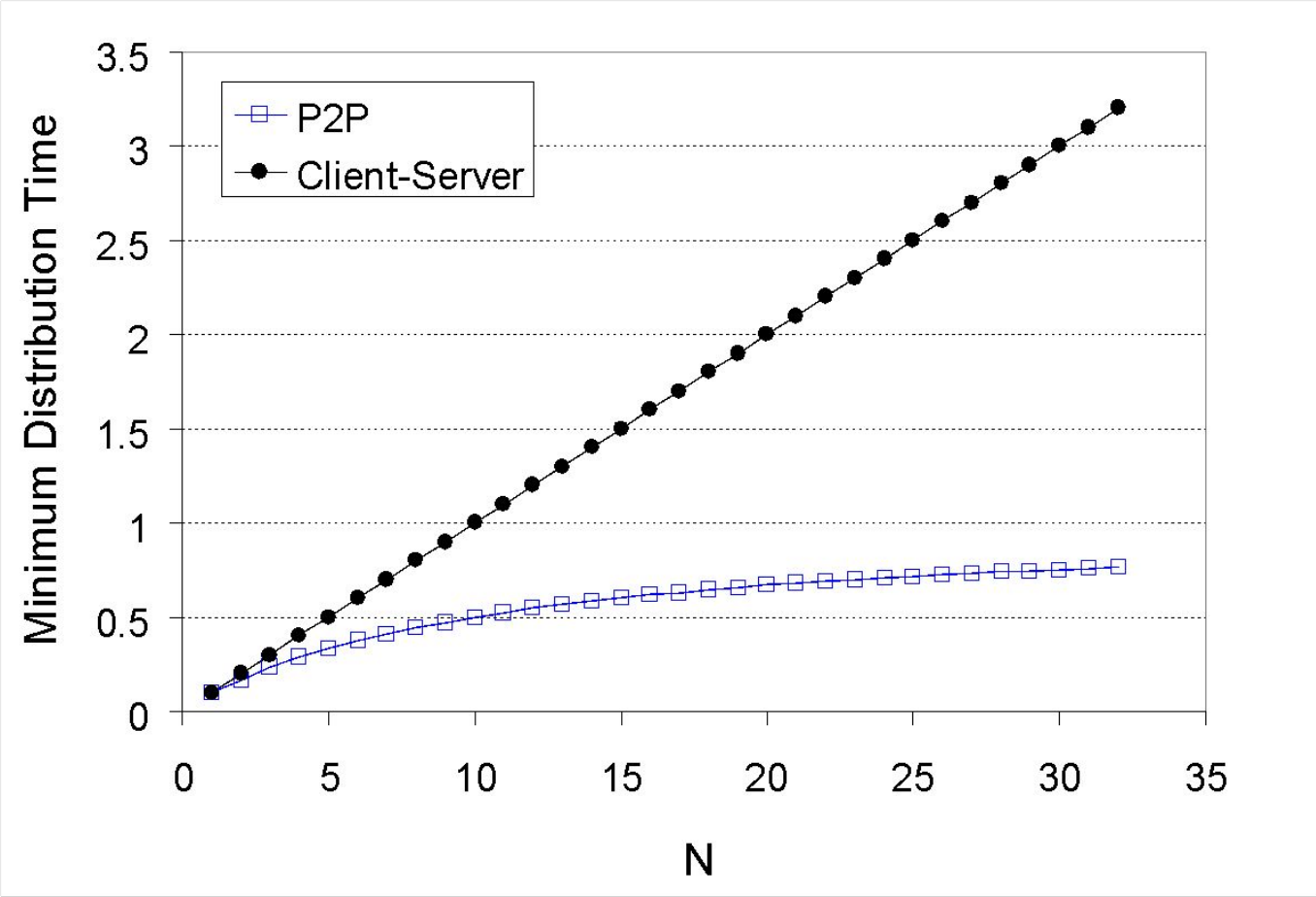
- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \sum u_i$

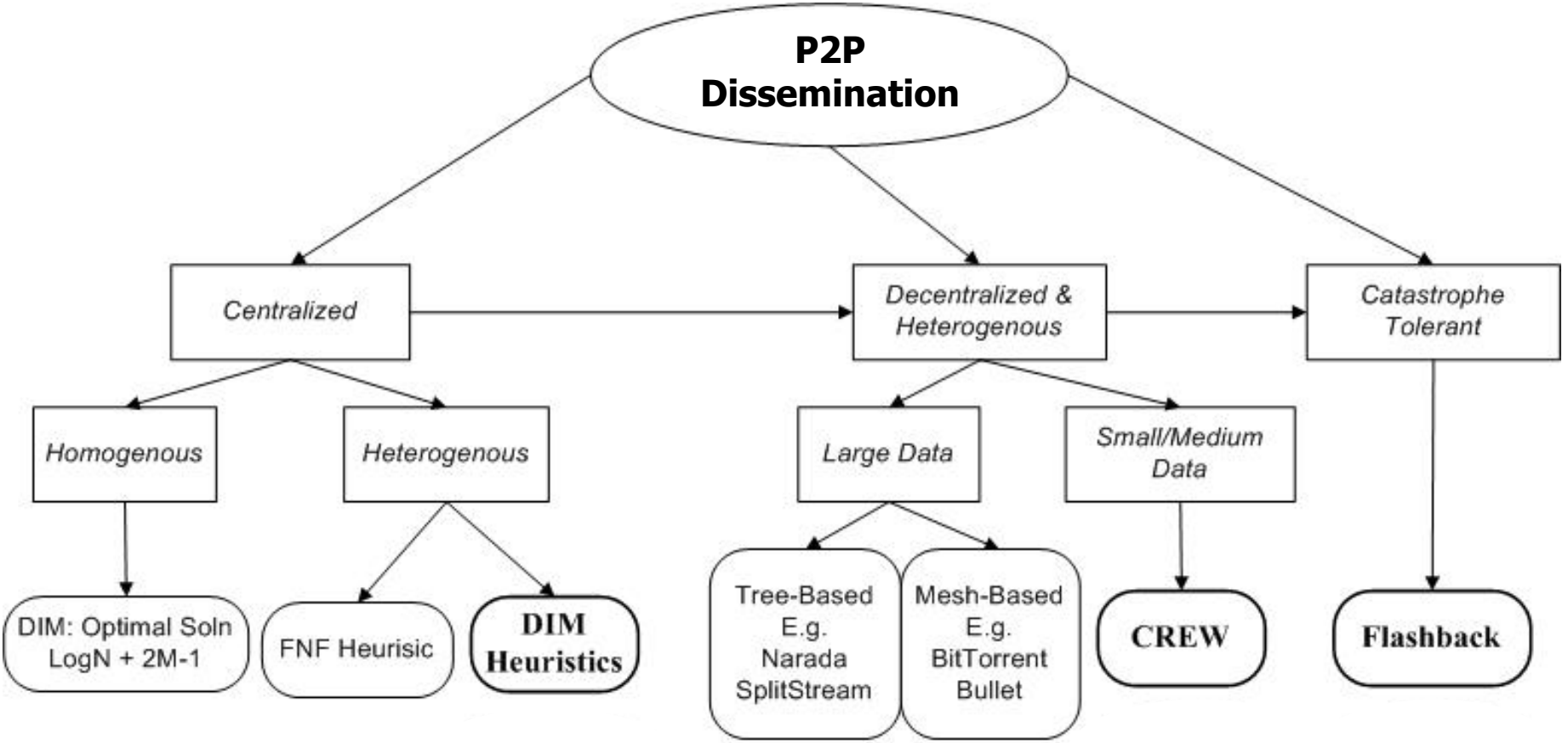


$$d_{P2P} = \max \left\{ F/u_s, F/\min(d_i) , NF/(u_s + \sum u_i) \right\}$$

Server-client vs. P2P: example

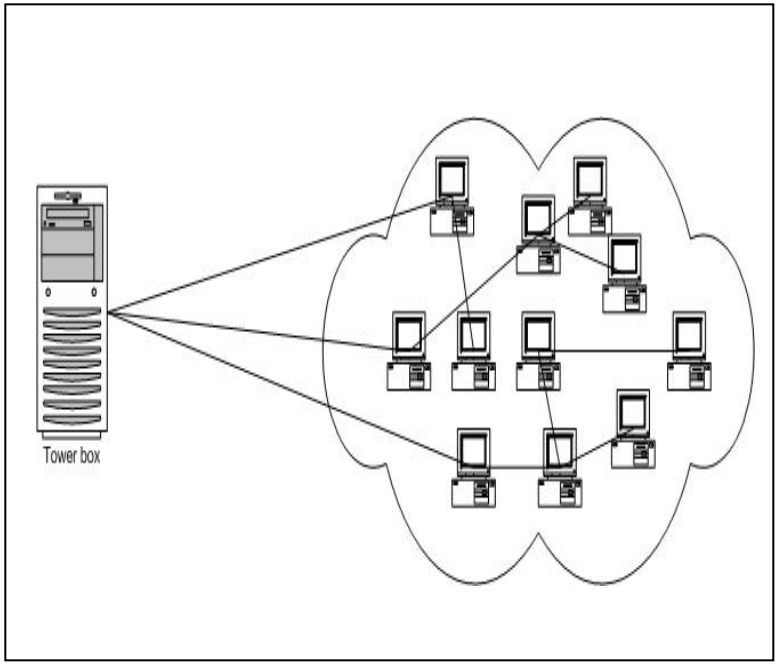
Client upload rate = u, F/u = 1 hour, u_s = 10u, $d_{min} \geq u_s$




P2P
Dissemination
Centralized
Decentralized &
Heterogenous
Catastrophe
Tolerant
Homogenous
Heterogenous
Large Data
Small/Medium
Data
DIM: Optimal Soln
LogN + 2M-1
FNF Heurisic
DIM
Heuristics
Tree-Based
E.g.
Narada
SplitStream
Mesh-Based
E.g.
BitTorrent
Bullet
CREW
Flashback

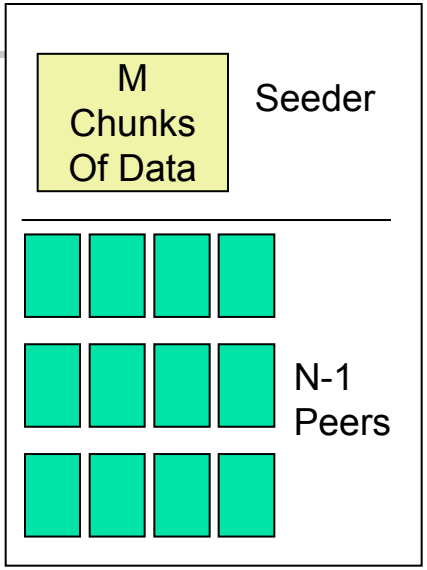
Problem Formulation

- Least time to disseminate:
  - Fixed data *D* from one seeder to *N* nodes
- Insights / Axioms
  - Involving end-nodes speeds up the process (Peer-to-Peer)
  - Chunking the data also speeds up the process
- Raises many questions
  - How do nodes find other nodes for exchange of chunks?
  - Which chunks should be transferred?
  - Is there an optimal way to do this?



Optimal Solution in Homogeneous Network

- Least time to disseminate:
  - All M chunks to N-1 peers
- Constraining the problem
  - Homogeneous network
  - All Links have same throughput & delay
  - Underlying network fully connected (Internet)

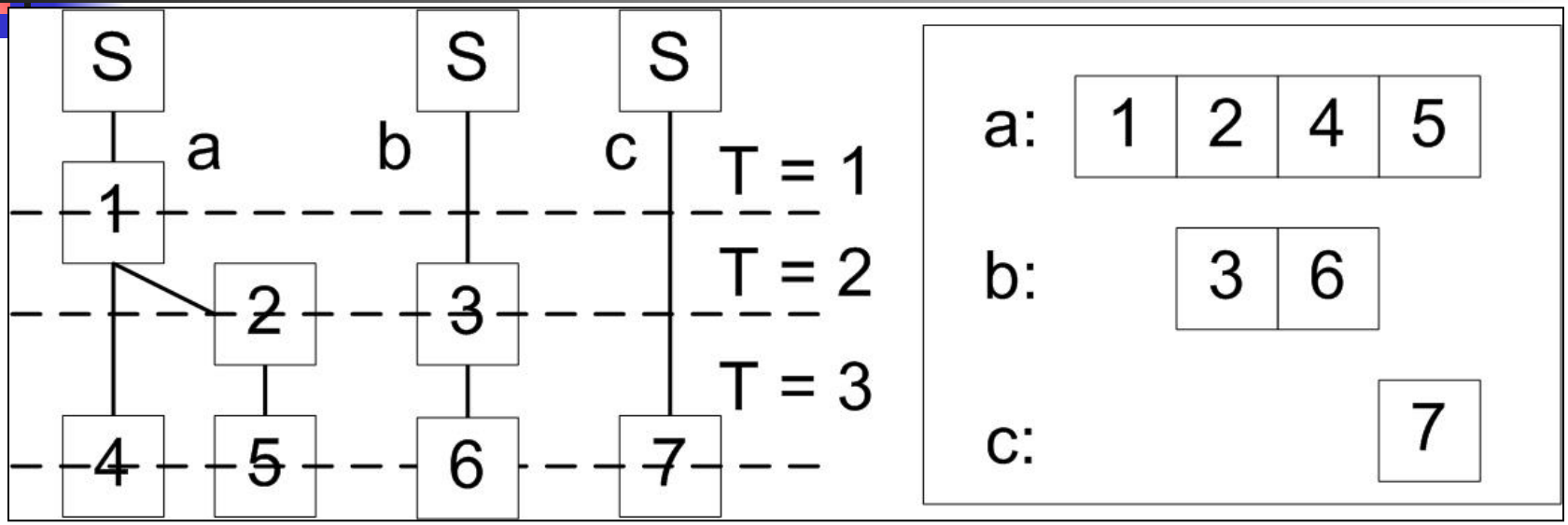


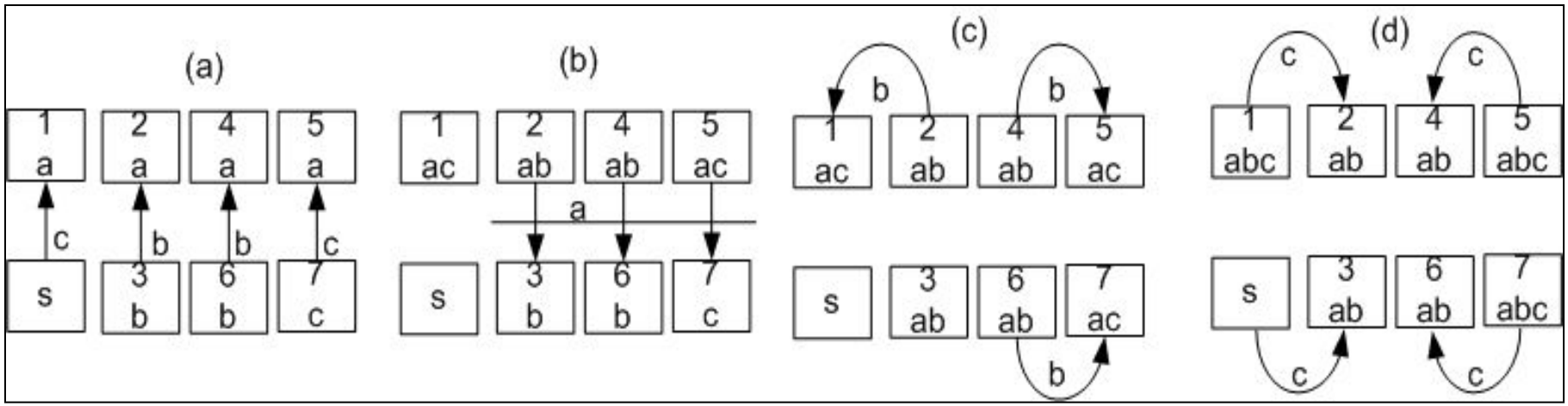
- Optimal Solution (DIM): $Log_2N + 2(M-1)$
  - *Ramp-Up: Until each node has at least 1 chunk*
  - *Sustained-Throughput: Until all nodes have all chunks*
- There is also an optimal chunk size

FARLEY, A. M. *Broadcast time in communication networks*. In SIAM Journal Applied Mathematics (1980)

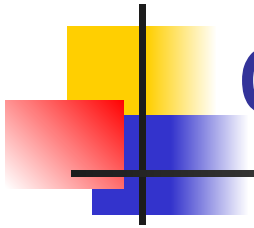
Ganesan, P. On Cooperative Content Distribution and the Price of Barter. ICDCS 2005

Example Working of Optimal Solution

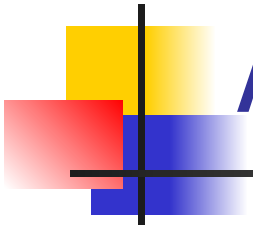




Practical Content dissemination systems



- Centralized
  - Server farms behind single domain name, load balancing
- Dedicated CDN
  - CDN is independent system for typically many providers, that clients only download from (use it as a service), typically http
  - Akamai, FastReplica
- End-to-End (P2P)
  - Special client is needed and clients self-organize to form the system themselves
  - BitTorrent(Mesh-swarm), SplitStream(forest), Bullet(tree+mesh), CREW(mesh)



Akamai

- Provider (eg CNN, BBC, etc) allows Akamai to handle a subset of its domains (authoritive DNS)
- Http requests for these domains are redirected to nearby proxies using DNS
  - Akamai DNS servers use extensive monitoring info to specify best proxy: adaptive to actual load, outages, etc
- 20,000+ servers worldwide, claimed 10-20% of overall Internet traffic is Akamai
- Wide area of services based on this architecture
  - availability, load balancing, web based applications, etc

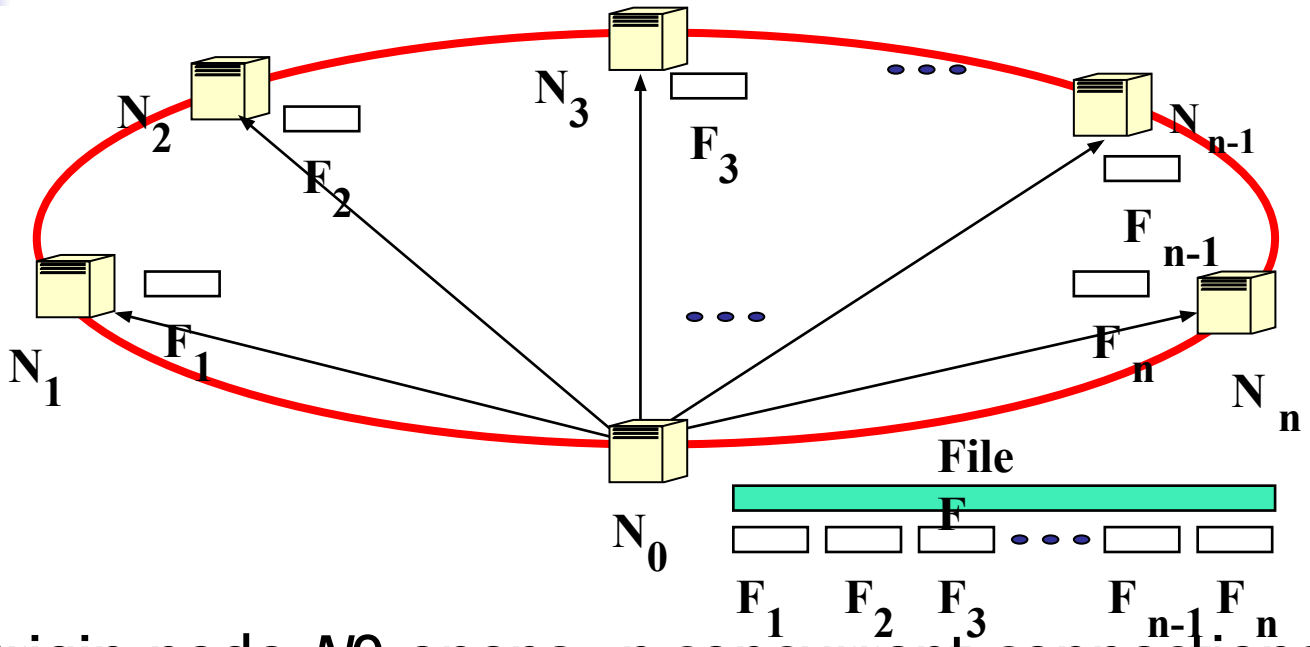
Distributed CDN : Fast Replica

- Disseminate large file to large set of edge servers or distributed CDN servers
- Minimization of the overall replication time for replicating a file *F* across *n* nodes $N_1, \ldots, N_n$.
- File *F* is divides in *n* equal subsequent files:

$F_1, \ldots, F_n$, where $Size(F_i) = Size(F) / n$ bytes for each $i = 1, \ldots, n$.

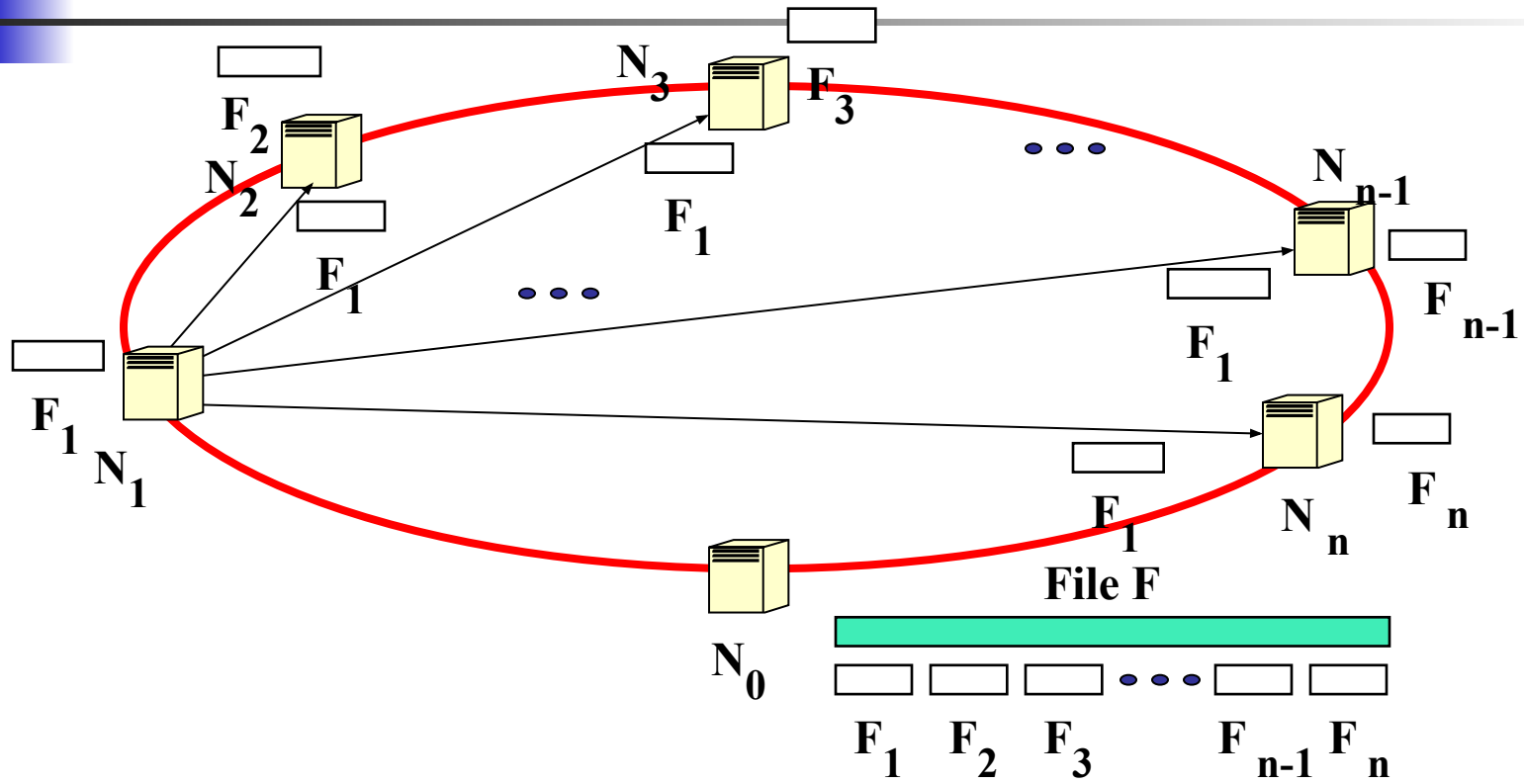
- Two steps of dissemination
  - Distribution and Collection

FastReplica : Distribution



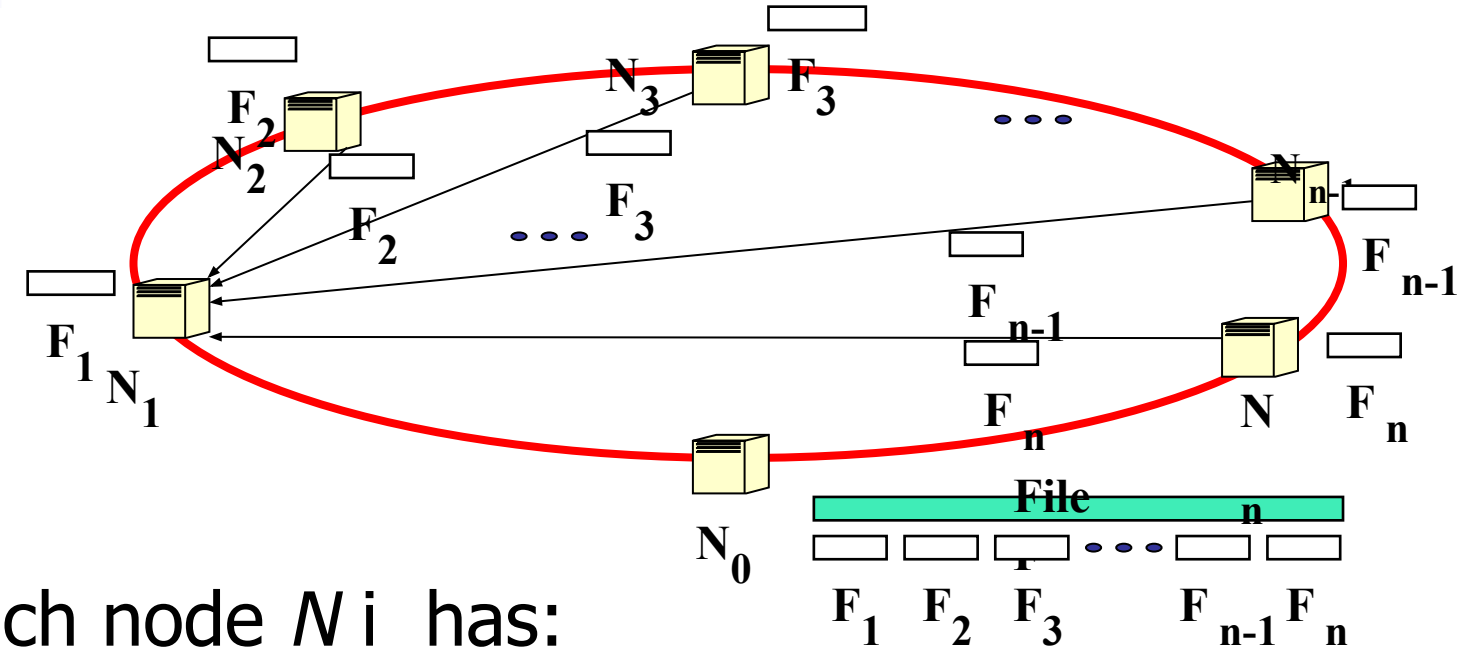
- Origin node *N*0 opens *n* concurrent connections to nodes *N*1, … , *N*n and sends to each node the following items:
  - a distribution list of nodes *R* = {*N*1, … , *N*n} to which subfile *F*i has to be sent on the next step;
  - subfile *F*i .

FastReplica : Collection



- After receiving *Fi* , node *Ni* opens (*n*-1) concurrent network connections to remaining nodes in the group and sends subfile *Fi* to them

FastReplica : Collection (overall)



- Each node *N* i has:
  - (*n* - 1) outgoing connections for sending subfile *F* i ,
  - (*n* - 1) incoming connections from the remaining nodes in the group for sending complementary subfiles *F* 1, … , *F* i-1 ,*F* i+1 , … , *F* n.

FastReplica : Benefits

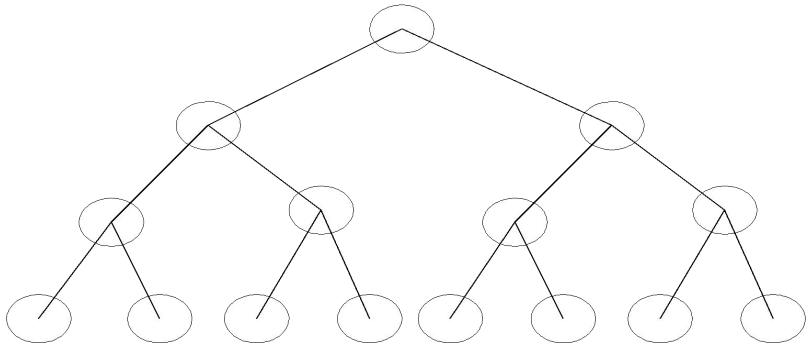
- Instead of typical replication of the entire file *F* to *n* nodes using *n* Internet paths *FastReplica* exploits *(n x n)* different Internet paths within the replication group, where each path is used for transferring *1/n*-th of file *F*.

- *Benefits:*
  - The impact of congestion along the involved paths is limited for a transfer of *1/n*-th of the file,
  - *FastReplica* takes advantage of the upload and download bandwidth of recipient nodes.

Decentralized Dissemination

Tree:

- Intuitive way to implement a decentralized solution
- Logic is built into the structure of the overlay



However:

- Sophisticated mechanisms for heterogeneous networks (SplitStream)
- Fault-tolerance Issues

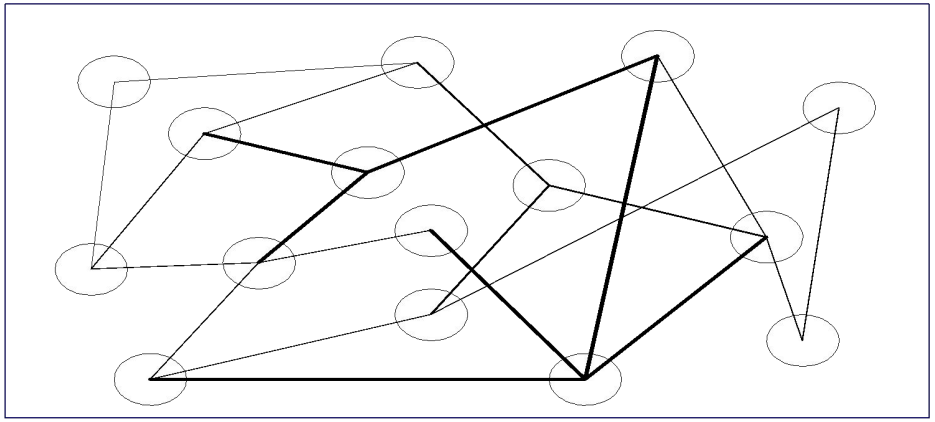
Mesh-Based (Bittorrent, Bullet):

- Multiple overlay links
- High-BW peers: more connections
- Neighbors exchange chunks

Robust to failures

- Find new neighbors when links are broken
- Chunks can be received via multiple paths

Simpler to implement



BitTorrent

- 20-50% of p2p internet traffic was BitTorrent (a decade ago)
- Special client software is needed
  - BitTorrent, BitTyrant, µTorrent, LimeWire …
- Basic idea
  - Clients that download a file at the same time help each other (ie, also upload chunks to each other)
  - BitTorrent clients form a swarm : a random overlay network

BitTorrent Traffic Is Suddenly Increasing — Possibly Due to 'Streaming Fragmentation'

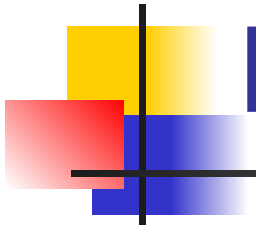
Ashley King October 1, 2018

BitTorrent is making a bit of a comeback. But why?

BitTorrent's traffic surges as the number of streaming services explode

By **Pavan Ramchandani** - October 2, 2018 - 7:00 am

BitTorrent : Publish/download



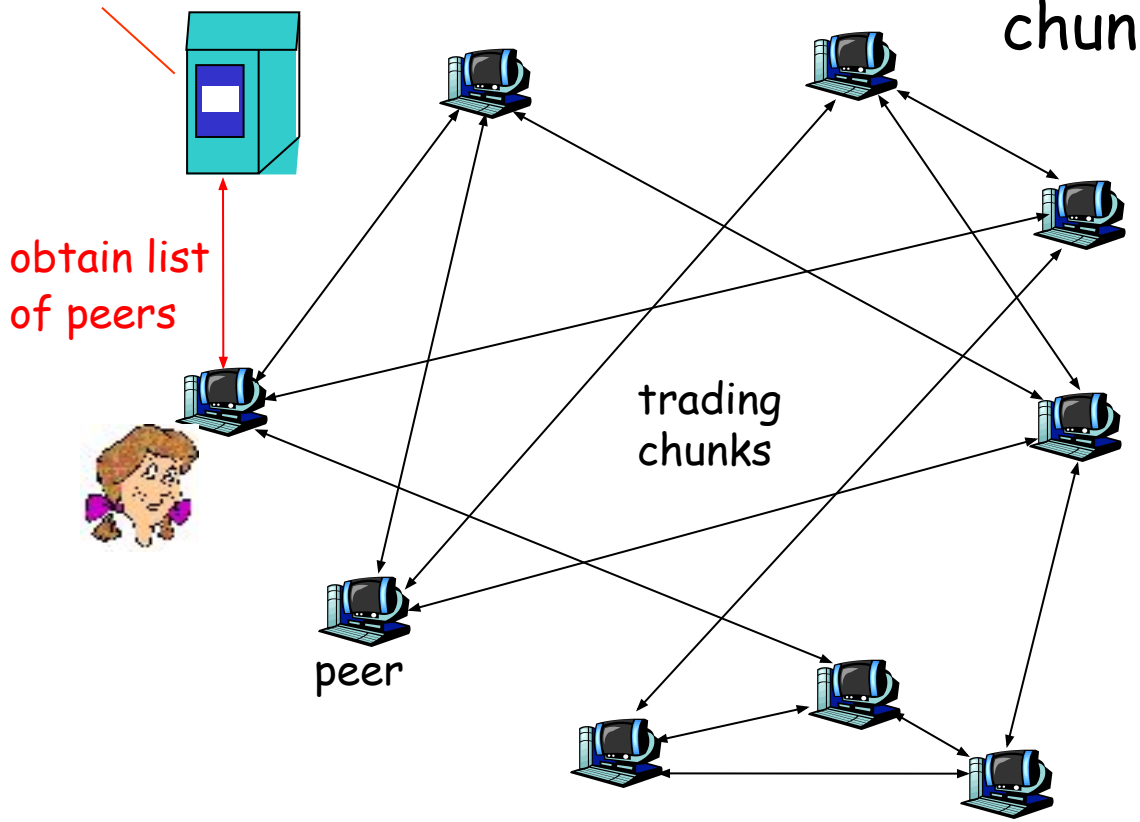
- Publishing a file
  - Put a ".torrent" file on the web: it contains the address of the tracker, and information about the published file
  - Start a tracker, a server that
    - Gives joining downloaders random peers to download from and to
    - Collects statistics about the swarm
  - There are "trackerless" implementations by using Kademlia DHT (e.g. Azureus)
- Download a file
  - Install a bittorrent client and click on a ".torrent" file

File distribution: BitTorrent

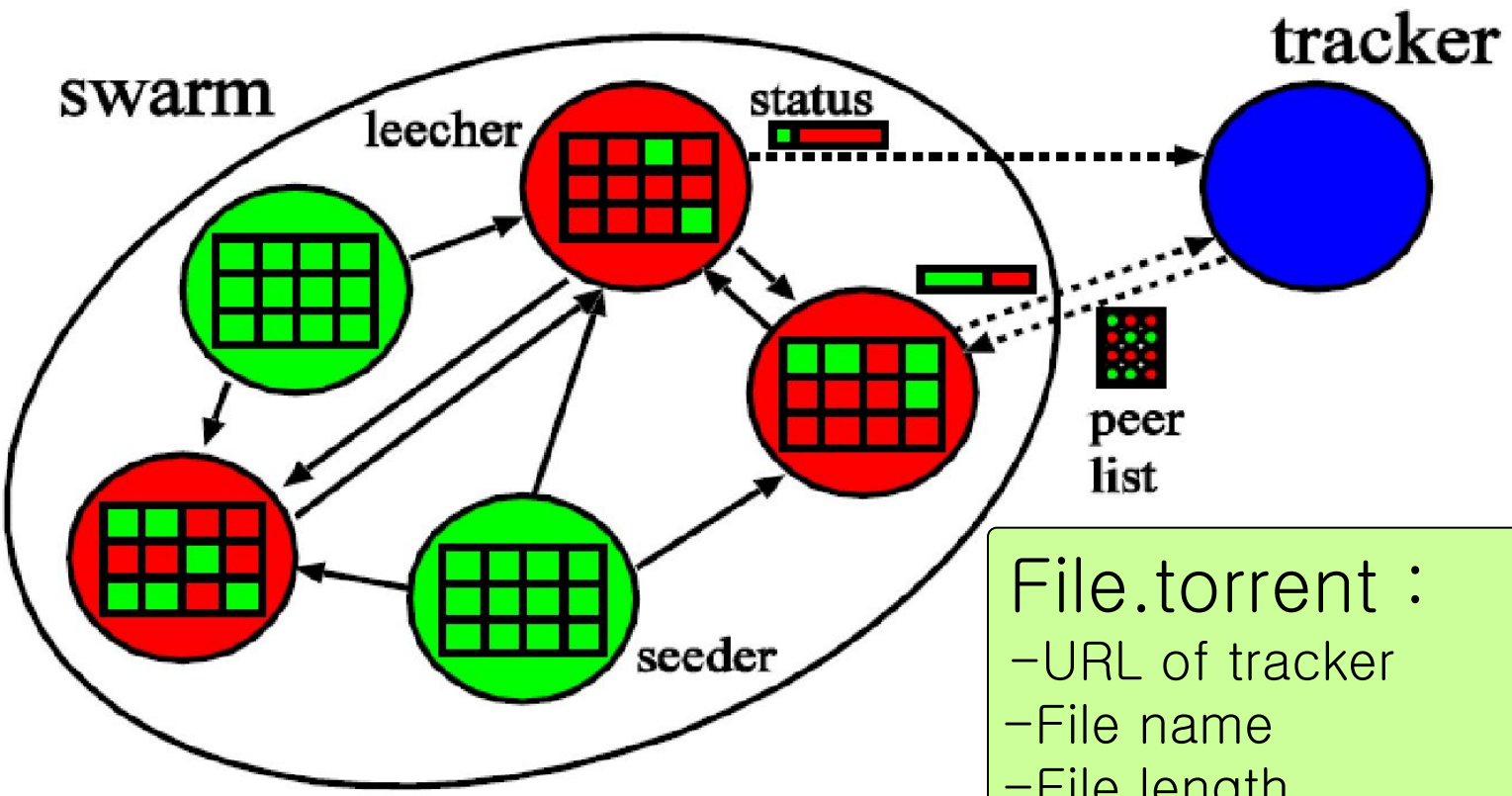
P2P file distribution

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file

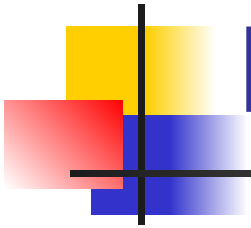


BitTorrent : Overview



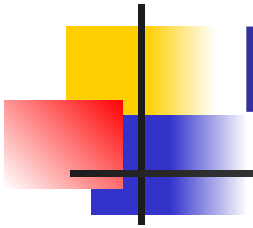
Seeder – peer having entire file
Leecher – peer downloading file

File.torrent :
- URL of tracker
- File name
- File length
- Chunk length
- Checksum for each chunk (SHA1 hash)



BitTorrent : Client

- Client first asks 50 random peers from tracker
  - Also learns about what chunks (256K) they have
- Pick a chunk and tries to download its pieces (16K) from the neighbors that have them
  - Download does not work if neighbor is disconnected or denies download (choking)
  - Only a complete chunk can be uploaded to others
- Allow only 4 neighbors to download (unchoking)
  - Periodically (30s) optimistic unchoking : allows download to random peer
    - important for bootstrapping and optimization
  - Otherwise unchokes peer that allows the most download (each 10s)



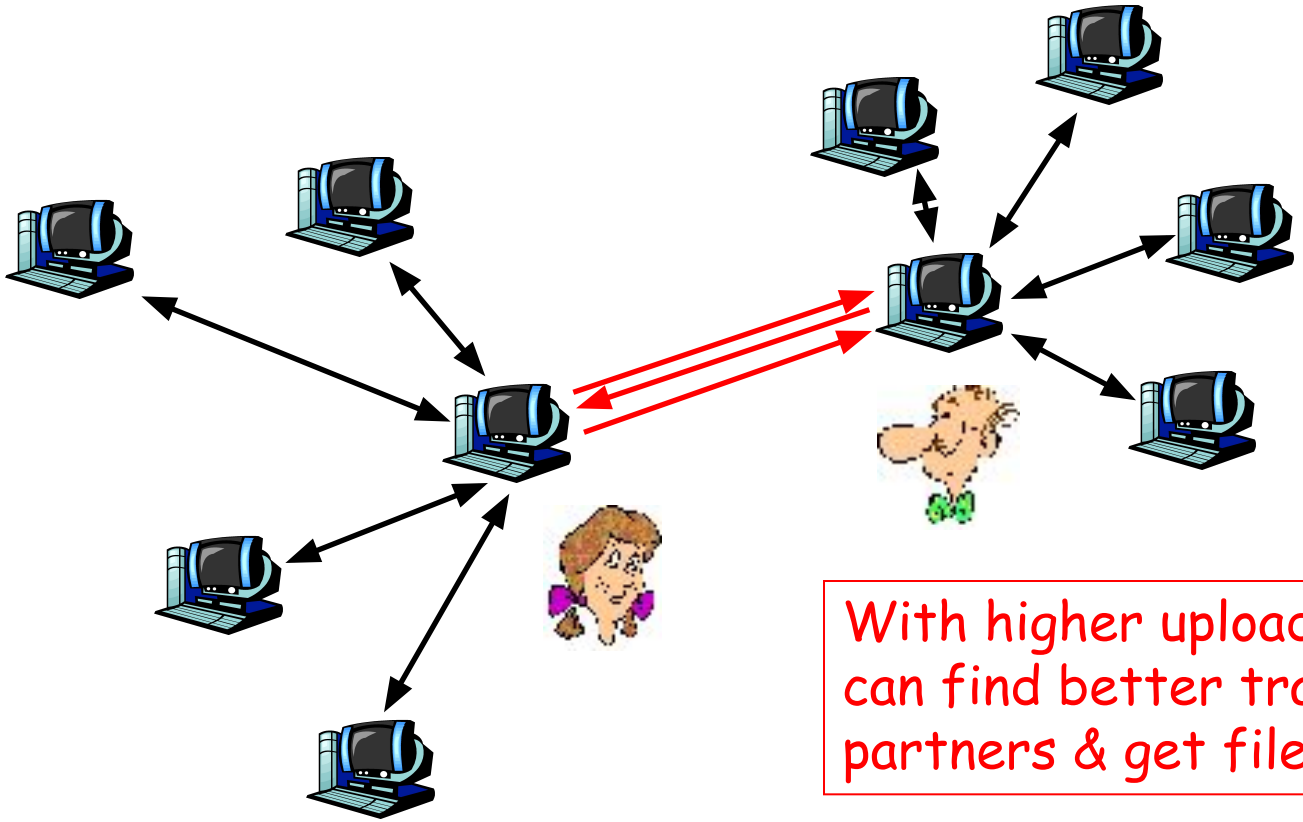
BitTorrent : Tit-for-Tat

- Tit-for-tat
  - Cooperate first, then do what the opponent did in the previous game
- BitTorrent enables tit-for-tat
  - A client unchokes other peers (allow them to download) that allowed it to download from them
  - Optimistic unchocking is the initial cooperation step to bootstrapping

BitTorrent: Tit-for-tat

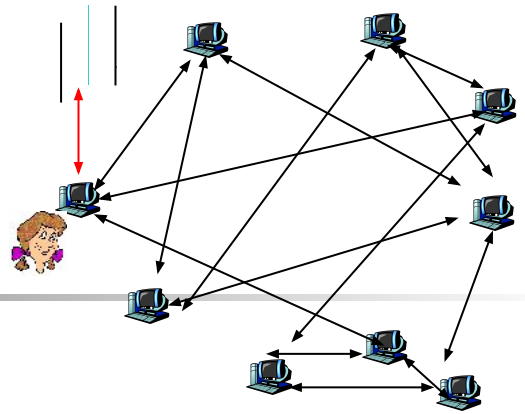
(1) Alice "optimistically unchokes" Bob
(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

(3) Bob becomes one of Alice's top-four providers



With higher upload rate, can find better trading partners & get file faster!

BitTorrent (1)



- file divided into 256KB *chunks*.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers.
- peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain

BitTorrent : Chunk selection

- What chunk to select to download?
- Clients select the chunk that is rarest among the neighbors (Local decision)
  - Increases diversity in the pieces downloaded; Increase throughput
  - Increases likelihood all pieces still available even if original seed leaves before any one node has downloaded entire file
- Except the first chunk
  - Select a random one (to make it fast: many neighbors must have it)

BitTorrent (2)

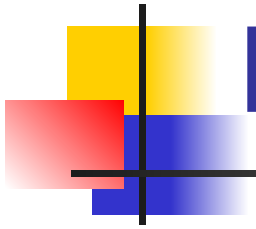
Pulling Chunks

- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

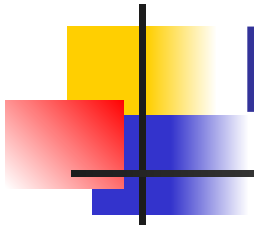
- Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4
  - "optimistically unchoke"

BitTorrent : Pros/Cons



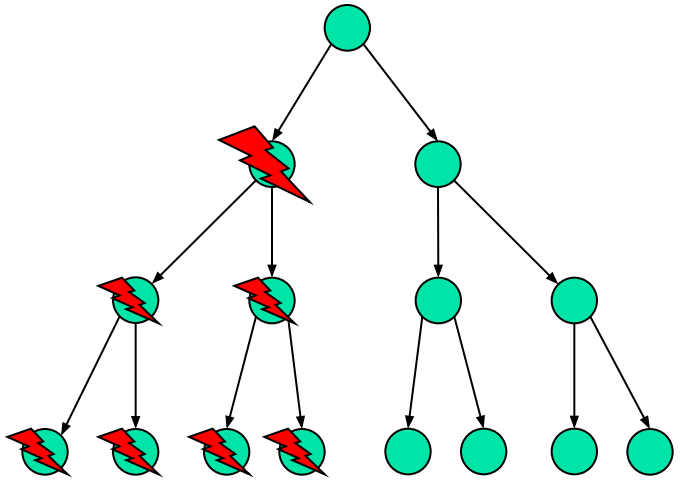
- Pros
  - Proficient in utilizing partially downloaded files
  - Encourages diversity through “rarest-first”
    - Extends lifetime of swarm
  - Works well for “hot content”
- Cons
  - Assumes all interested peers active at same time; performance deteriorates if swarm “cools off”
  - Even worse: no trackers for obscure content

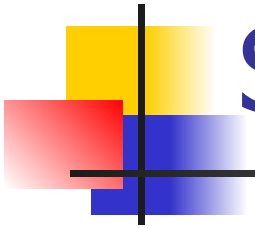
More P2P Content Dissemination



Overcome tree structure – SplitStream, Bullet

- Tree
  - Simple, Efficient, Scalable
  - But, vulnerable to failures, load-unbalanced, no bandwidth constraint
- SplitStream
  - Forest (Multiple Trees)
- Bullet
  - Tree(Metadata) + Mesh(Data)
- CREW
  - Mesh(Data,Metadata)





SplitStream

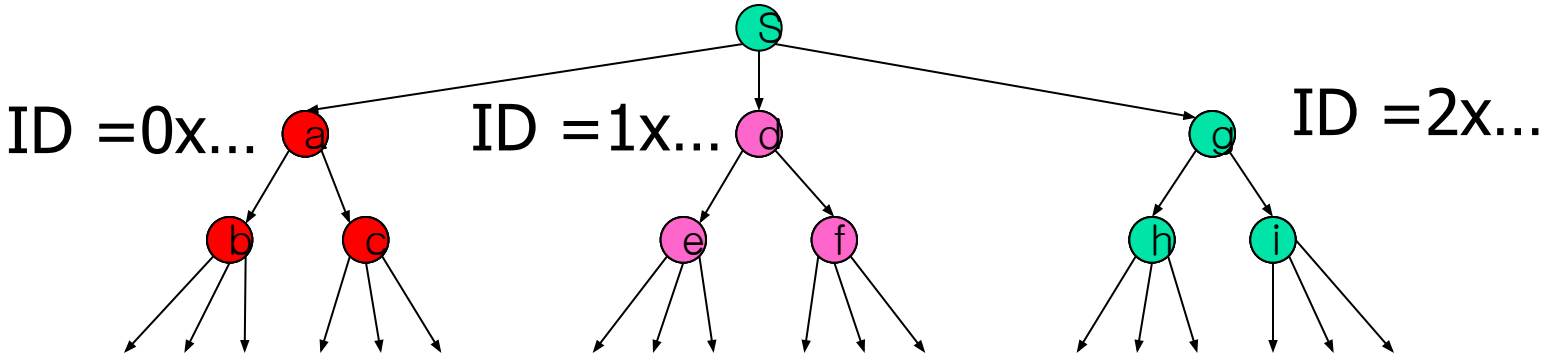
- Forest based dissemination
- Basic idea
  - Split the stream into K stripes (with MDC coding)
  - For each stripe create a multicast tree such that the forest
    - Contains interior-node-disjoint trees
    - Respects nodes' individual bandwidth constraints

SplitStream : MDC coding

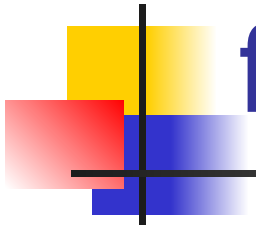
- Multiple Description coding
  - Fragments a single media stream into M substreams ($M \geq 2$)
  - K packets are enough for decoding ($K < M$)
  - Less than K packets can be used to approximate content
    - Useful for multimedia (video, audio) but not for other data
    - Cf) erasure coding for large data file

SplitStream : Interior-node-disjoint tree

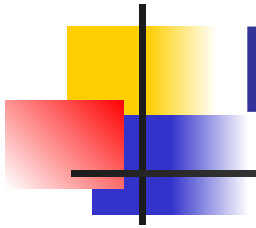
- Each node in a set of trees is interior node in at most one tree and leaf node in the other trees.
- Each substream is disseminated over subtrees



SplitStream : Constructing the forest



- Each stream has its groupID
  - Each groupID starts with a different digit
- A subtree is formed by the routes from all members to the groupId
  - The nodeIds of all interior nodes share some number of starting digits with the subtree's groupId.
- All nodes have incoming capacity requirements (number of stripes they need) and outgoing capacity limits

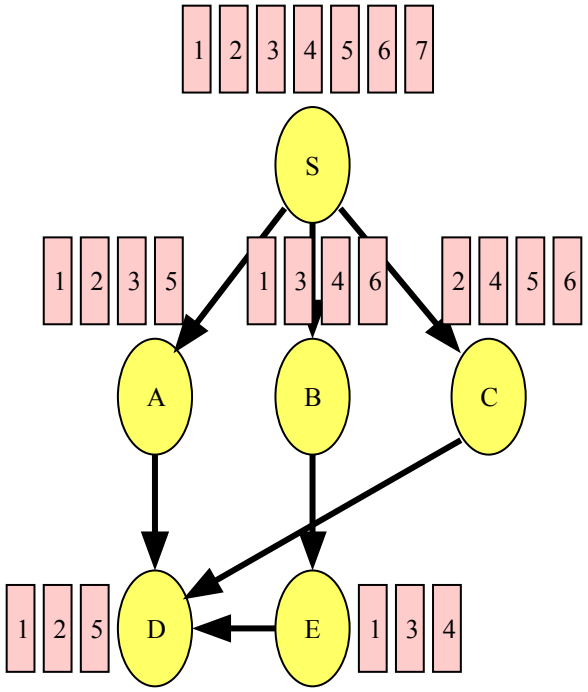


Bullet

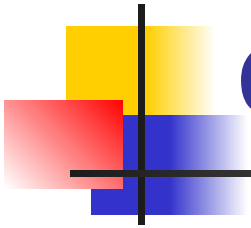
- Layers a mesh on top of an overlay tree to increase overall bandwidth
- Basic Idea
  - Use a tree as a basis
  - In addition, each node continuously looks for peers to download from
  - In effect, the overlay is a tree combined with a random network (mesh)

Bullet : RanSub

- Two phases
  - **Collect phase** : using the tree, membership info is propagated upward (random sample and subtree size)
  - **Distribution phase** : moving down the tree, all nodes are provided with a random sample from the entire tree, or from the non-descendant part of the tree

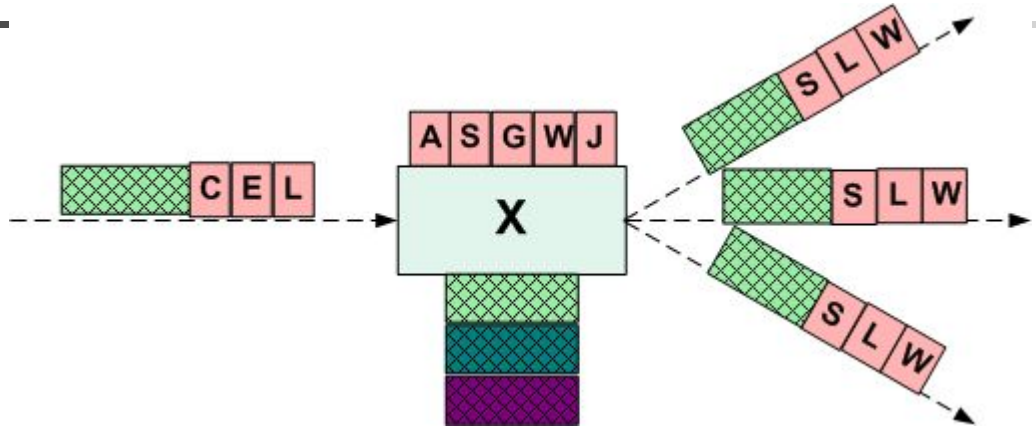


Bullet : Informed content delivery



- When selecting a peer, first a similarity measure is calculated
  - Based on summary-sketches
- Before exchange missing packets need to be identified
  - Bloom filter of available packets is exchanged
  - Old packets are removed from the filter
    - To keep the size of the set constant
- Periodically re-evaluate senders
  - If needed, senders are dropped and new ones are requested

Gossip-based Broadcast



Probabilistic Approach with Good Fault Tolerant Properties

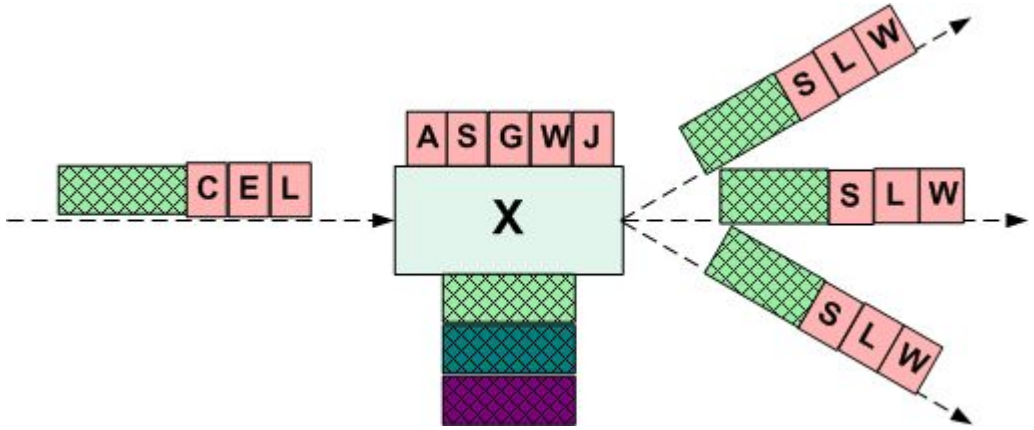
- Choose a destination node, uniformly at random, and send it the message
- After Log(N) rounds, all nodes will have the message w.h.p.
- Requires N*Log(N) messages in total
- Needs a 'random sampling' service

Usually implemented as

- Rebroadcast 'fanout' times
- Using UDP: Fire and Forget

BiModal Multicast (99), Lpbcast (DSN 01), Rodrigues'04 (DSN), Brahami '04, Verma'06 (ICDCS), Eugster'04 (Computer), Koldehofe'04, Periera'03

Gossip-based Broadcast: Drawbacks



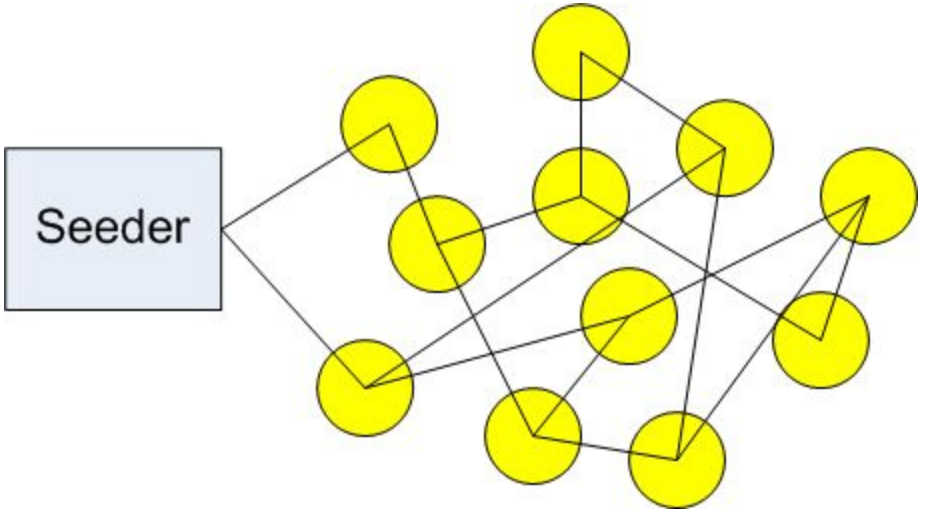
Problems

- More faults, higher fanout needed (not dynamically adjustable)
- Higher redundancy ☐ lower system throughput ☐ slower dissemination
- Scalable view & buffer management
- Adapting to nodes' heterogeneity
- Adapting to congestion in underlying network

CREW: Preliminaries

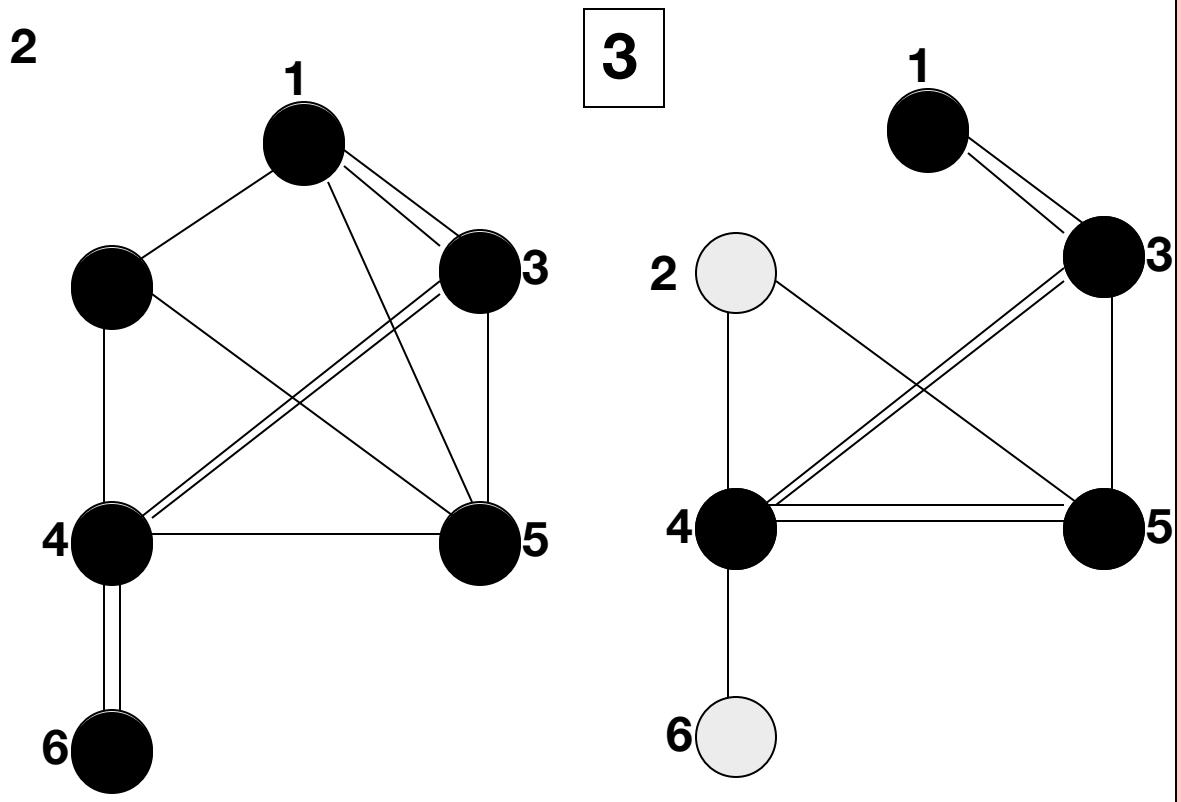
MetaData

File Attributes Name, MimeType, Size, etc.	
Chunk-1	Checksum
Chunk-2	Checksum
Chunk-i	Checksum
Chunk-M	Checksum



Deshpande, M., et al. CREW: A Gossip-based Flash-Dissemination System IEEE International Conference on Distributed Computing Systems (ICDCS). 2006.

CREW (Concurrent Random Expanding Walkers) Protocol



- **Basic Idea:** Servers 'serve' data to only a few clients
  - Who In turn become servers and 'recruit' more servers
- Split data into chunks
  - Chunks are concurrently disseminated through random-walks
- Self-scaling and self-tuning to heterogeneity

What is new about CREW

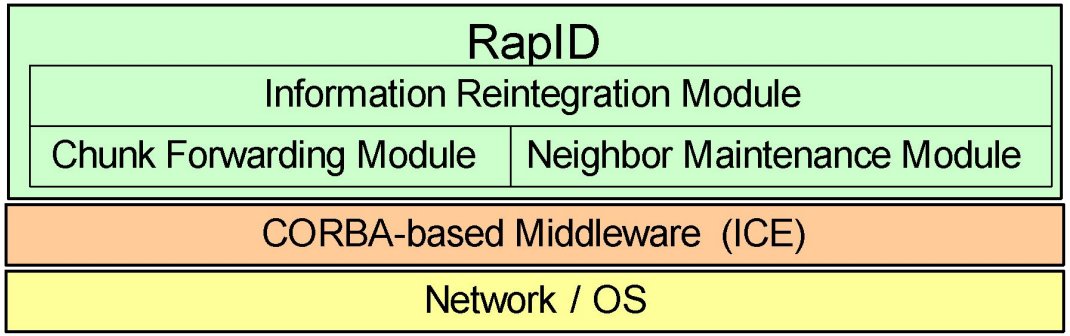
- No need to pre-decide fanout or complex protocol to adjust it
  - Deterministic termination
  - Autonomic adaptation to fault level (More faults ☐more pulls)
- Scalable, real-time and low-overhead view management
  - Number of neighbors as low as Log(N) (expander overlay)
  - Neighbors detect and remove dead node ☐ disappears from all nodes' views instantly
  - List of node addresses not transmitted in each gossip message
- Use of metadata plus handshake to reduce data overhead
  - No transmission of redundant chunks

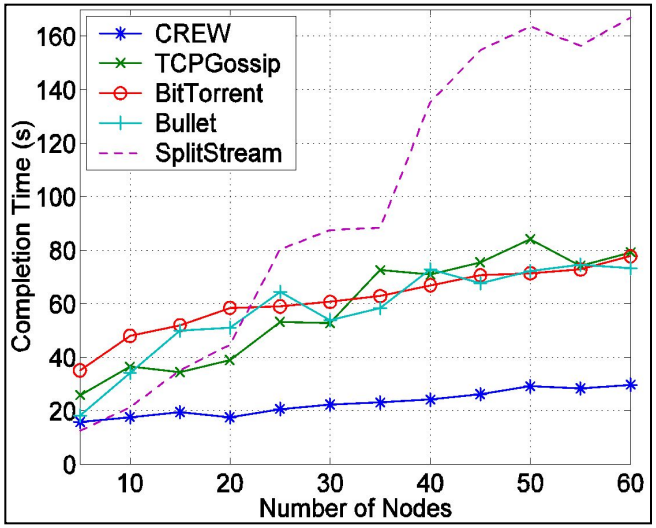
- Handshake overloading
  - For 'random sampling' of the overlay
  - Quick feedback about system-wide properties
  - Quick adaptation

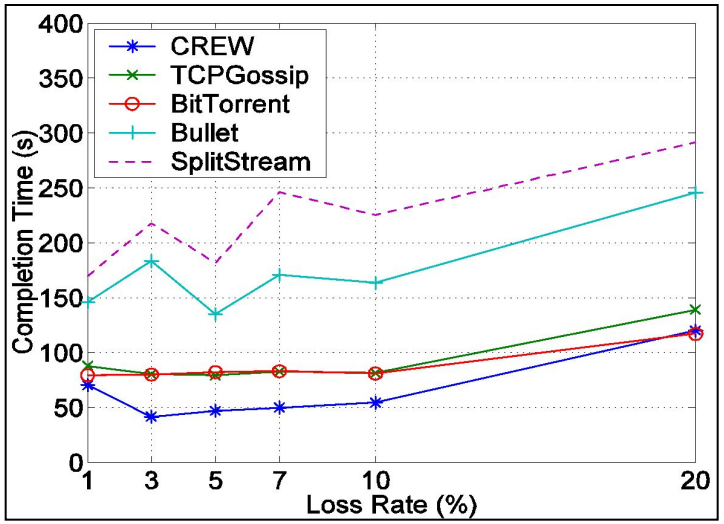
- Use of TCP as underlying transport
  - Automatic flow and congestion control at network level
  - Less complexity in application layer

- Implemented using RPC middleware

CREW Protocol: Latency, Reliability







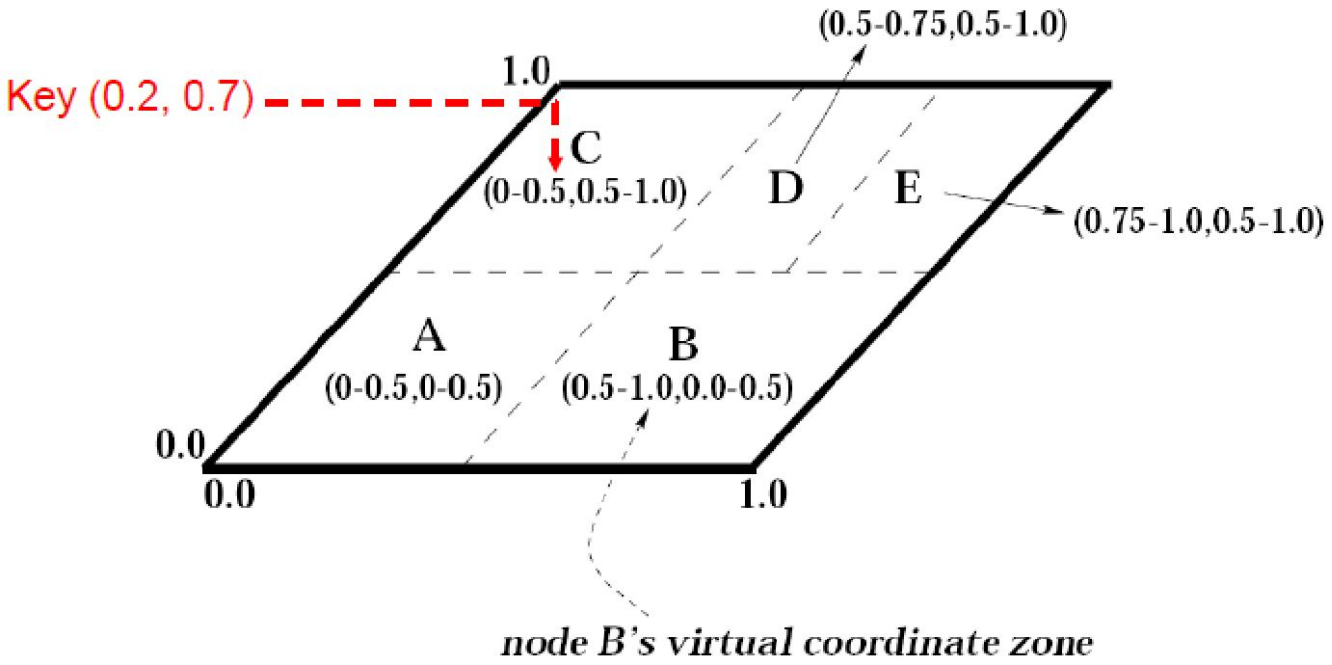

BitTorrent (BTT)
Potential??
COINBUREAU
COINBUREAU

EXTRA SLIDES

More on P2P Search/Lookup

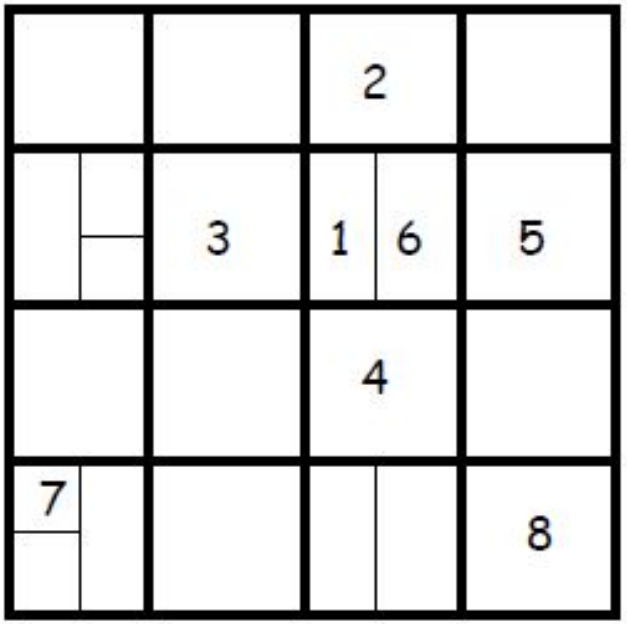
CAN : Content Addressable Network

- Hash value is viewed as a point in a D-dimensional Cartesian space
  - Hash value points <n1, n2, …, nD> as a key.
  - D-dimensional requires D distinct hash functions.
- Each node responsible for a D-dimensional "cube" in the space



CAN : Neighbors

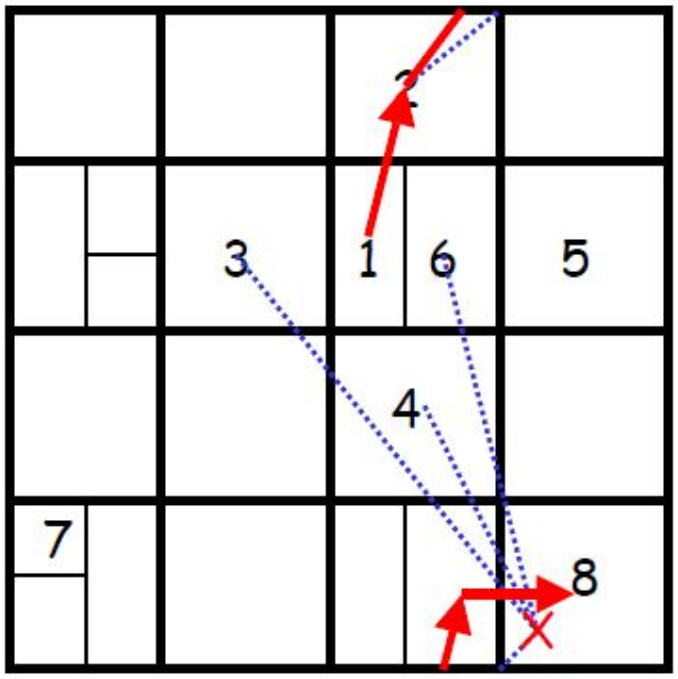
- Nodes are neighbors if their cubes "touch" at more than just a point
  - Neighbor information : Responsible space and node IP Address



- Example: D=2
- 1's neighbors: 2,3,4,6
- 6's neighbors: 1,2,4,5
- Squares "wrap around", e.g., 7 and 8 are neighbors
- Expected # neighbors: O(D)

CAN : Routing

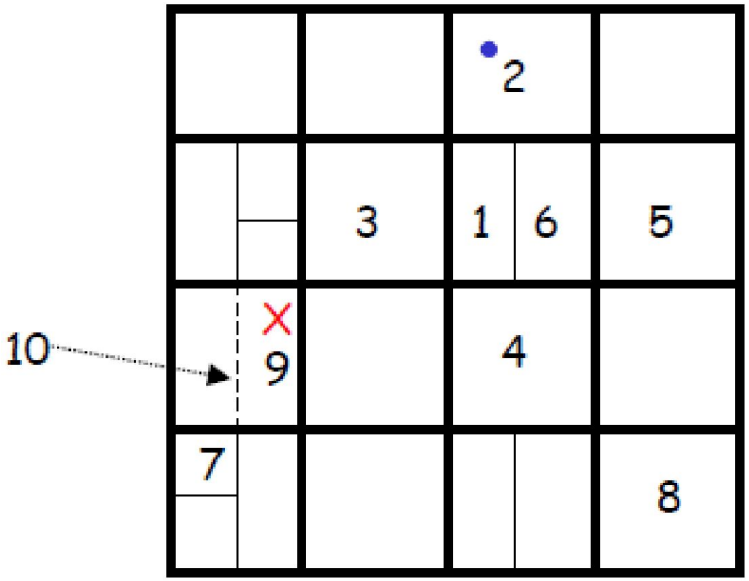
- To get to <n1, n2, …, nD> from <m1, m2, …, mD>
  - choose a neighbor with smallest Cartesian distance from <n1, n2, …, nD> (e.g., measured from neighbor's center)



- e.g., region 1 needs to send to node covering X
- Checks all neighbors, node 2 is closest
- Forwards message to node 2
- Cartesian distance monotonically decreases with each transmission
- Expected # overlay hops: $(DN^{1/D})/4$

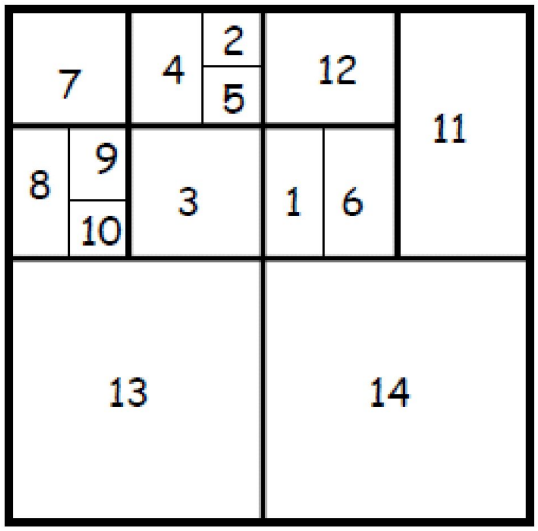
CAN : Join

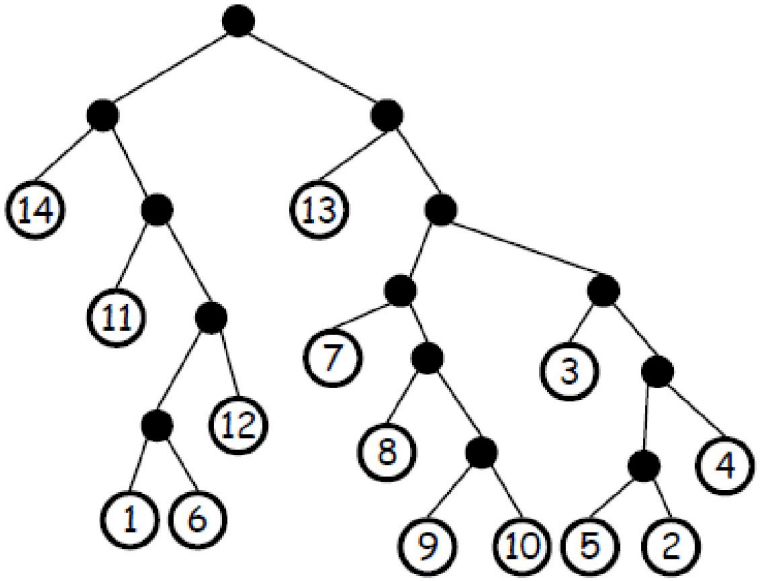
- To join the CAN overlay:
  - find some node in the CAN (via bootstrap process)
  - choose a point in the space uniformly at random
  - using CAN, inform the node that currently covers the space that node splits its space in half
    - 1st split along 1st dimension
    - if last split along dimension i < D, next split along i+1st dimension
    - e.g., for 2-d case, split on x-axis, then y-axis
  - keeps half the space and gives other half to joining node



The likelihood of a rectangle being selected is proportional to it's size, i.e., big rectangles chosen more frequently

CAN Failure recovery

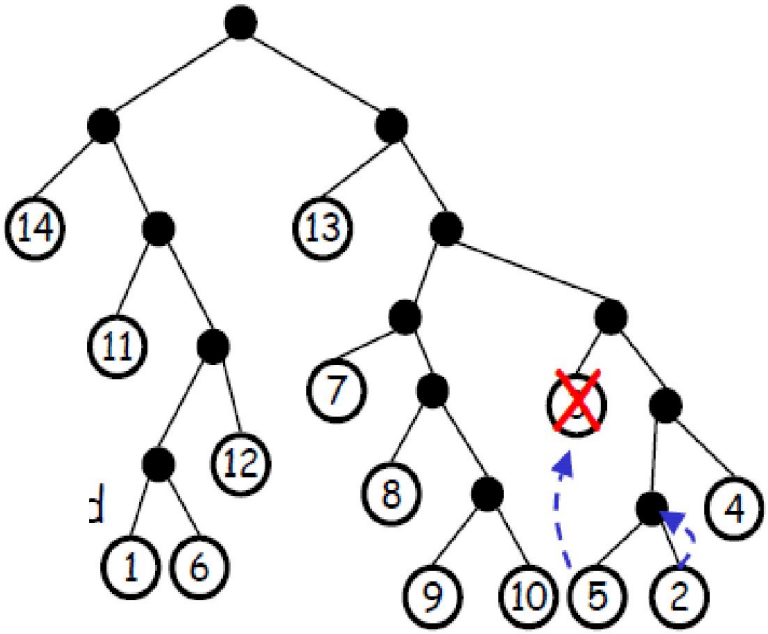


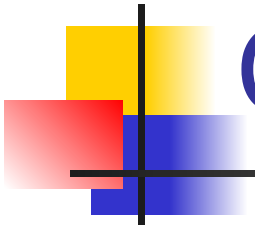


- View partitioning as a binary tree
  - Leaves represent regions covered by overlay nodes
  - Intermediate nodes represents "split" regions that could be "reformed"
  - Siblings are regions that can be merged together (forming the region that is covered by their parent)

CAN Failure Recovery

- Failure recovery when leaf S is removed
  - Find a leaf node T that is either
    - S's sibling
    - Descendant of S's sibling where T's sibling is also a leaf node
  - T takes over S's region (move to S's position on the tree)
  - T's sibling takes over T's previous region

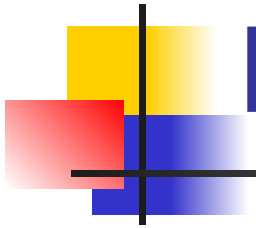




CAN : speed up routing

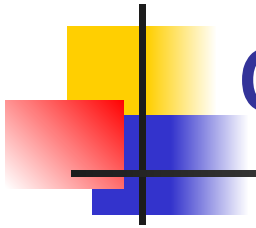
- Basic CAN routing is slower than Chord or Pastry
- Manage long ranged links
  - Probabilistically maintain multi-hop away links (2 hop away, 3 hop away ..)
  - Exploit the nested group routing

Kademlia : BitTorrent DHT



- Developed in 2002
- For Distributed Tracker
  - trackerless torrent
  - Torrent files are maintained by all users using BitTorrent.
- For each nodes, files, keywords, deploy SHA-1 hash into a 160 bits space.
- Every node maintains information about files, keywords “close to itself”.

Kademlia : XOR based closeness

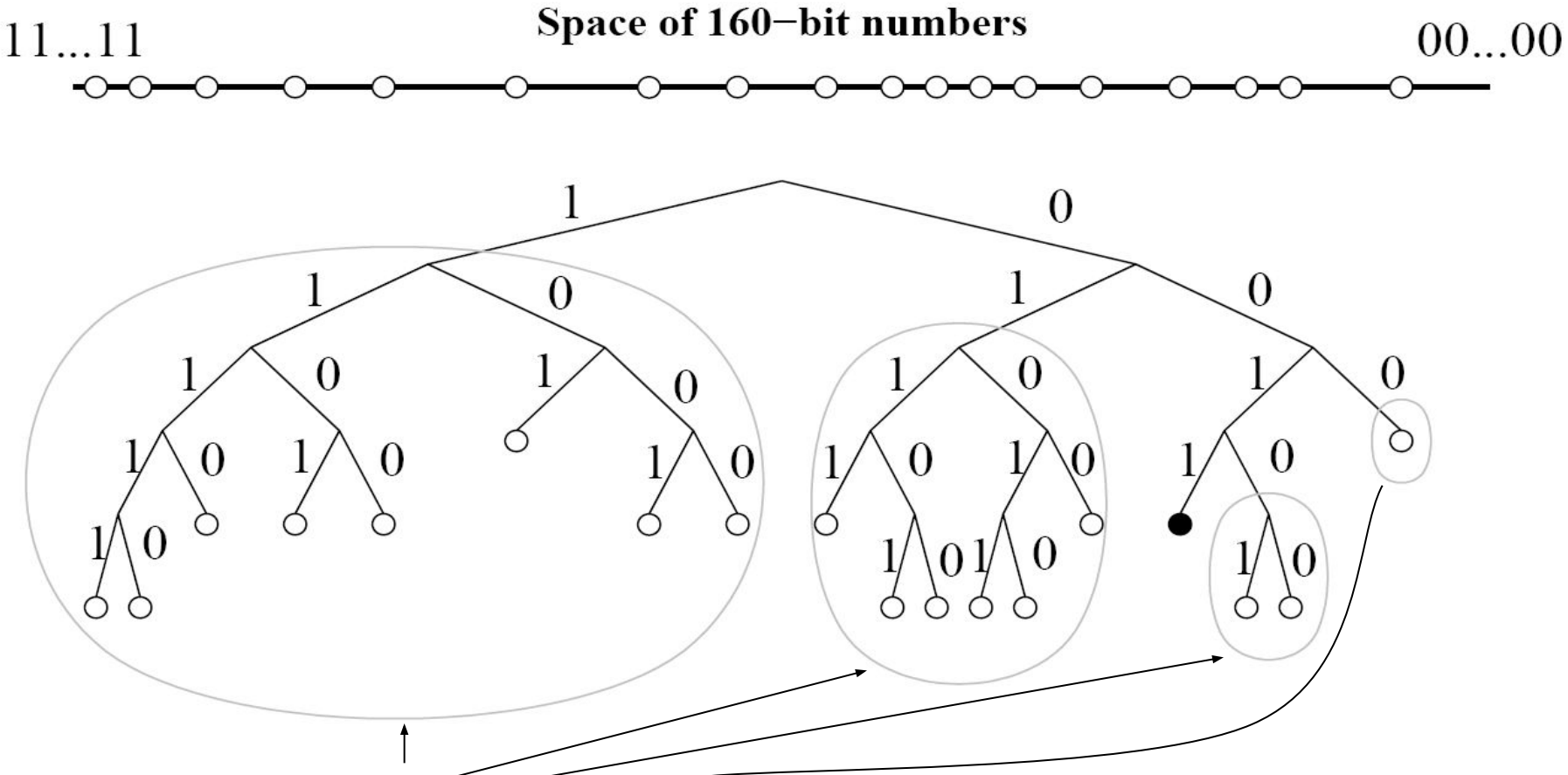


- The closeness between two objects measure as their bitwise XOR interpreted as an integer.
- D(a, b) = a XOR b
  - d (x,x) = 0
  - d (x,y) > 0 if x ≠ y
  - d (x,y) = d (y,x)
  - d (x,y) + d (y,z) ≥ d (x, z)
  - For each x and t, there is exactly one node y for which d (x,y) = t

Kademlia : Binary Tree of ID Space

- Treat node as leaves in a binary tree.
- For any given node, dividing the binary tree into a series of successively lower subtree that don't contain the node.
- For any given node, it keeps touch at least one node (up to k) of its subtrees. (if there is a node in that tree.) Each subtree possesses a k-bucket.

Kademlia : Binary Tree of ID Space

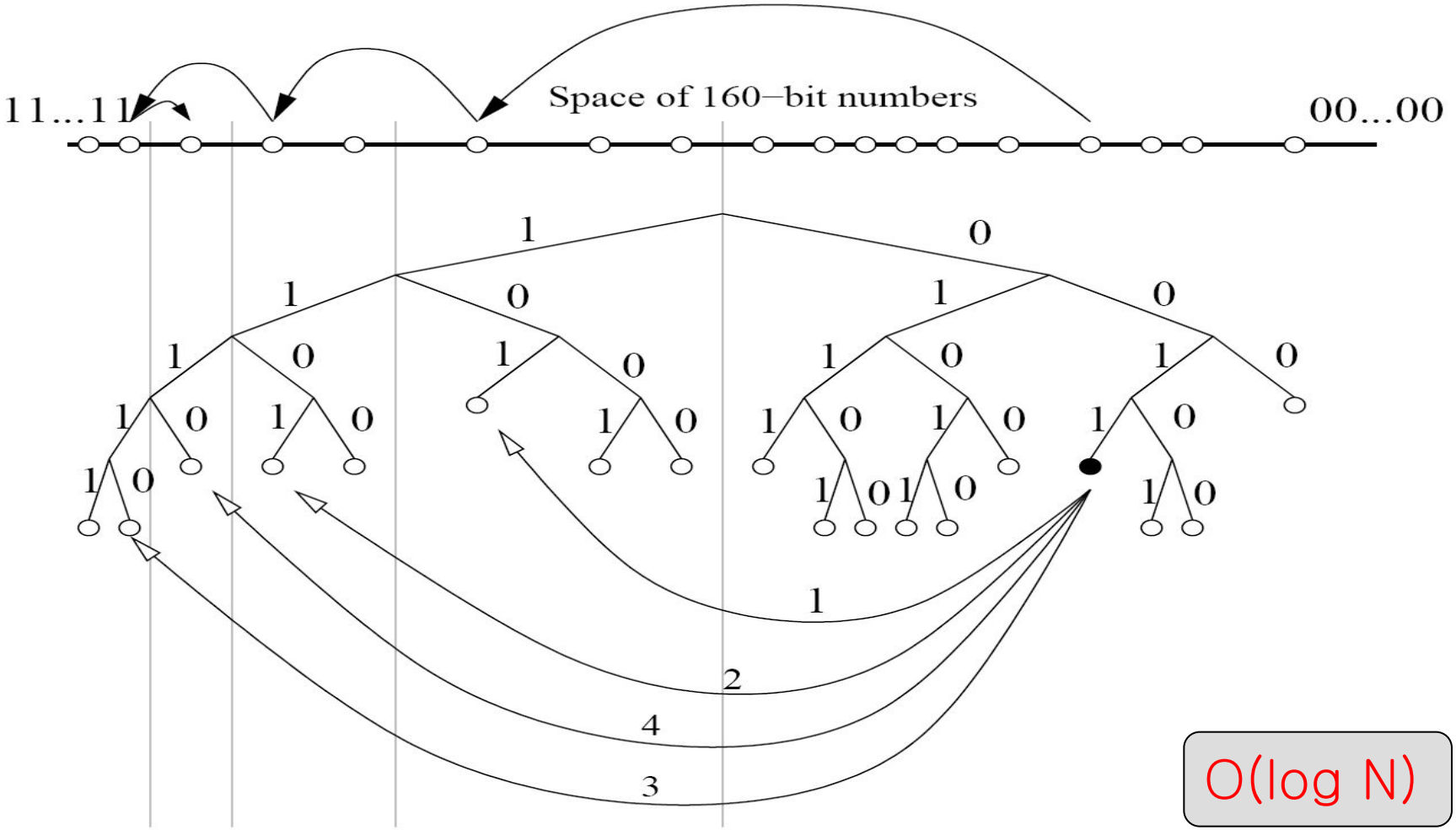


Subtrees for node 0011....
c.f. nested group

Each subtree has k buckets (delegate nodes), K = 20 in general

Kademlia : Lookup

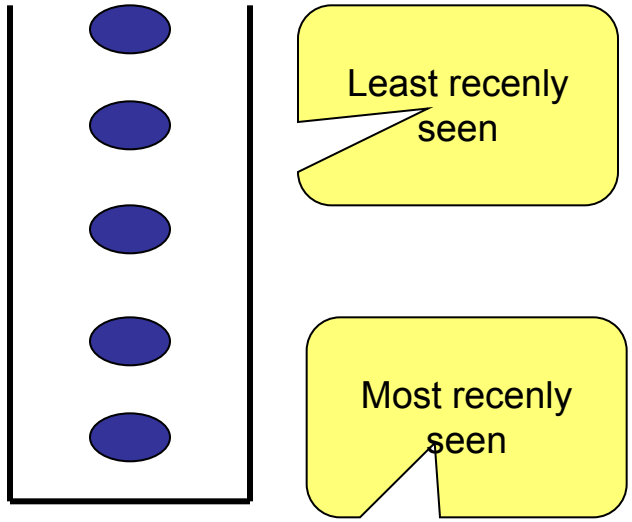
When node 0011..... wants search 1110.....



O(log N)

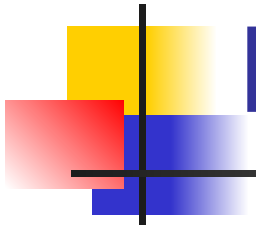
Kademlia : K-bucket

- K-bucket for each subtree
  - A list of nodes of a subtree
  - The list is sorted by time last seen.
  - The value of K is chosen so that any give set of K nodes is unlikely to fail within an hour.
    - So, K : Reliability parameter
  - The list is updated whenever a node receives a message.



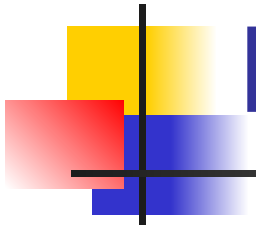
Gnutella showed that the longer a node Is up, the more likely it is to remain up for one more hour

Kademlia : K-bucket



- By relying on the oldest nodes, k-buckets promise the probability that they will remain online.
- Dos attack is prevented since the new nodes find it difficult to get into the k-bucket
- If malicious users live long and dominate all the K-bucket, what happens?
  - Eclipse attack
  - Sybil attack

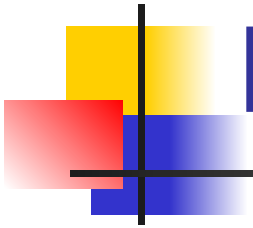
Kademlia : RPC



- PING: to test whether a node is online
- STORE: instruct a node to store a key
- FIND_NODE: takes an ID as an argument, a recipient returns (IP address, UDP port, node id) of k nodes it knows from closest to ID (node lookup)
- FIND_VALUE: behaves like FIND_NODE, unless the recipient received a STORE for that key, it just returns the stored value.

Kademlia : Lookup

- The most important task is to locate the k closest nodes to some given node ID.
- Kademlia employs a recursive algorithm for node lookups. The lookup initiator starts by picking a nodes from its closest non-empty k-bucket.
- The initiator then sends parallel, asynchronous FIND_NODE to the α nodes it has chosen.
- α is a system-wide concurrency parameter, such as 3.
  - Flexibility of choosing online nodes from k-buckets
  - Reducing latency



Kademlia : Lookup

- The initiator resends the FIND_NODE to nodes it has learned about from previous RPCs.
- If a round of FIND_NODES fails to return a node any closer than the closest already seen, the initiator resends the FIND_NODE to all of the k closest nodes it has not already queried.
- The lookup terminates when the initiator has queried and gotten responses from the k closest nodes it has seen.