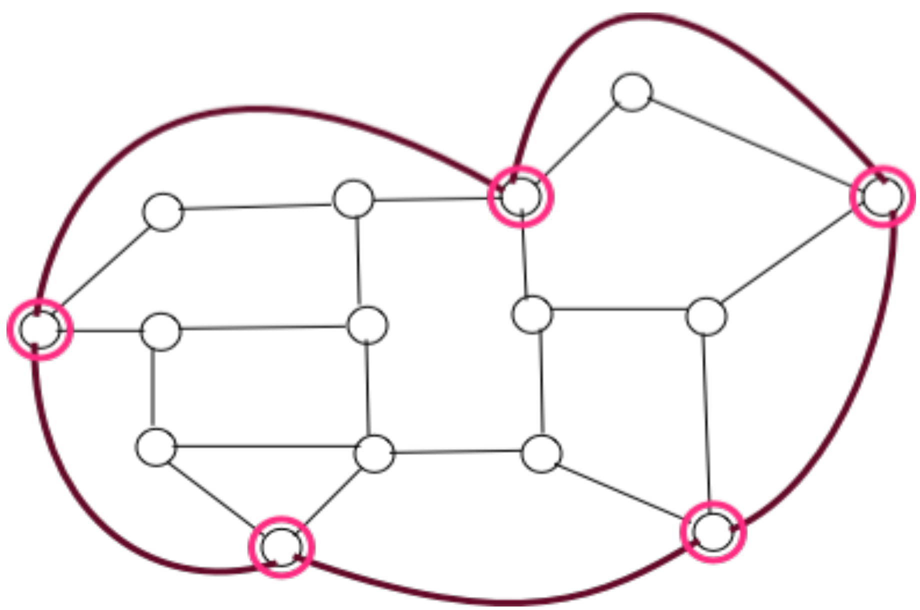
Peer-to-Peer and Social Networks

An overview of Gnutella

Overlay networks

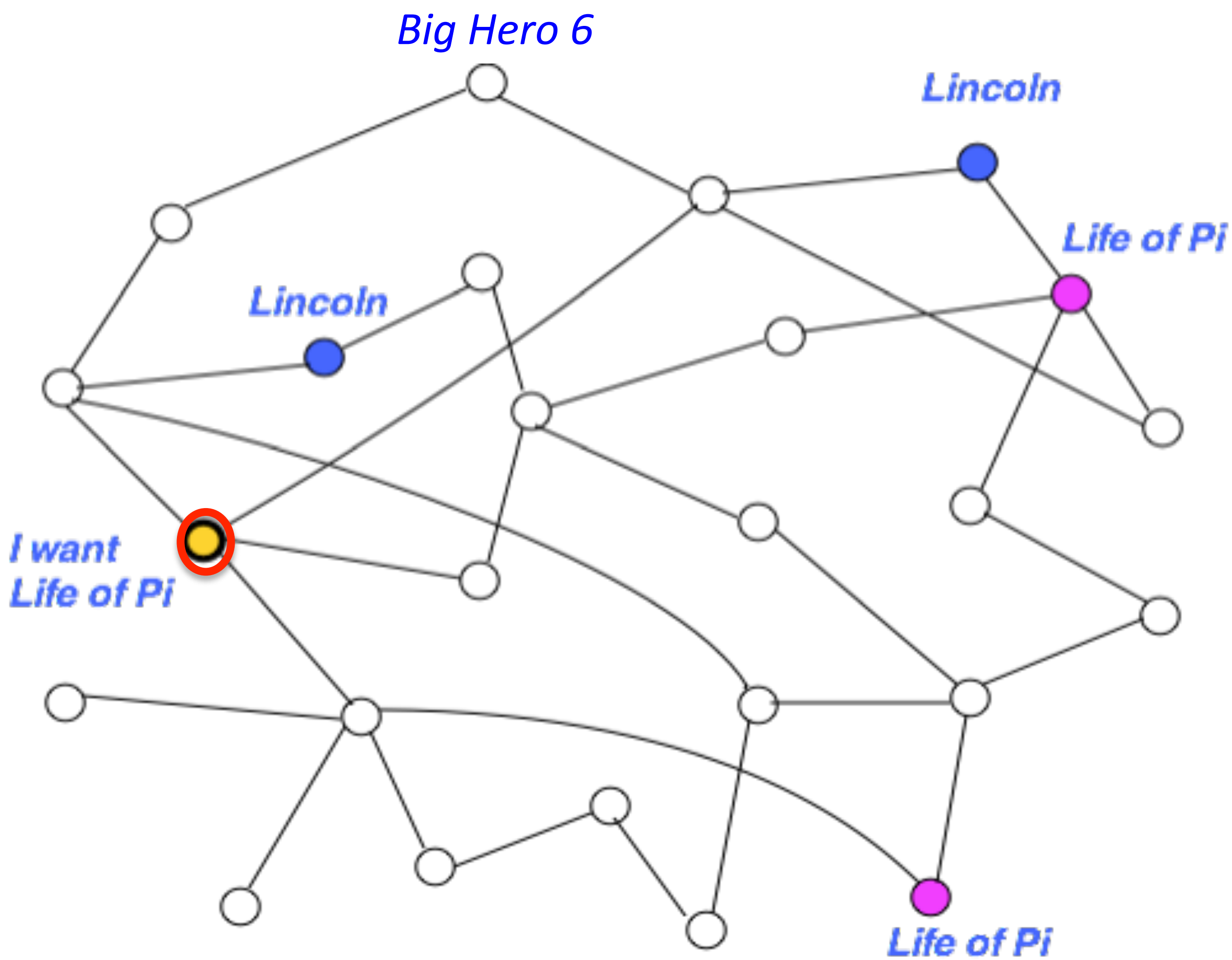
Overlay networks are logical networks defined on top of a physical network. The nodes (peers) are a subset of the real nodes at the edge of the physical network, but the links are logical links. The links can be modified by the peers if necessary. **No central server** is there to oversee this.



History

The Gnutella network is a fully distributed alternative of the centralized Napster. Initial popularity of the network received a boost after Napster's legal demise in early 2001.

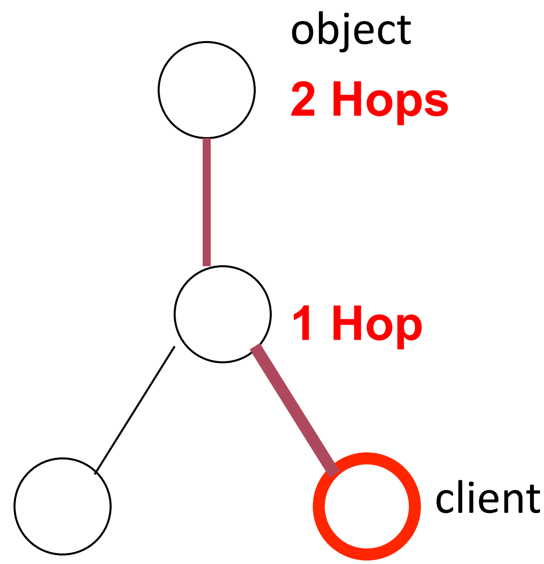
What is Gnutella



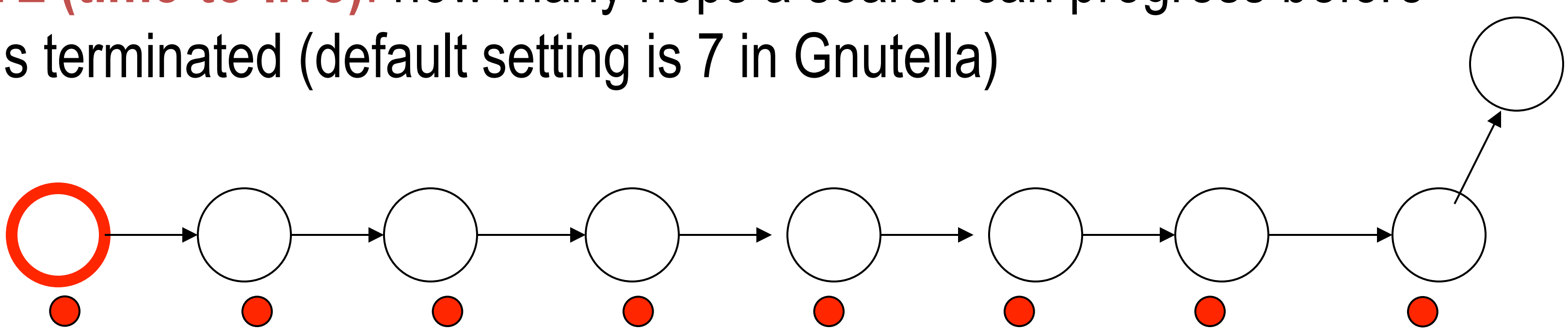
Gnutella is a ***search protocol*** with no central authority.

Gnutella Jargon

Each node is both a server and a client ("servent").



TTL (time-to-live): how many hops a search can progress before it is terminated (default setting is 7 in Gnutella)



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 forward your Ping to other connected peers with open connections
- A Pong packet also contains:
  - an IP address
  - port number
  - Pong packets come back via same route

Step 2: Searching

- Gnutella "Query" ask other peers if they have the file you desire. A Query packet might ask, ***"Do you have any song whose name matches "Once upon a time"?***
- Peers check to see if they have matches & respond (if they have any matches) & send packet to connected peers. Otherwise the query is forwarded
- Continues for TTL

Step 3: Downloading

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

Remarks

- Very simple idea , but **lacks scalability**, since query flooding wastes bandwidth.
- Sometimes, existing objects may not be located due to limited TTL.

Various improved search strategies have been proposed. These have been used by newer client of the Gnutella protocol, like Limewire. Improvements use Ultrapeer, pong caching etc.

Searching in Gnutella

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.

Measurements provide useful information about the topology. Candidate topologies are

-- Random graph

-- Power law graph

-- Small world graphs

Gnutella topology

Gnutella topology is actually a power-law graph

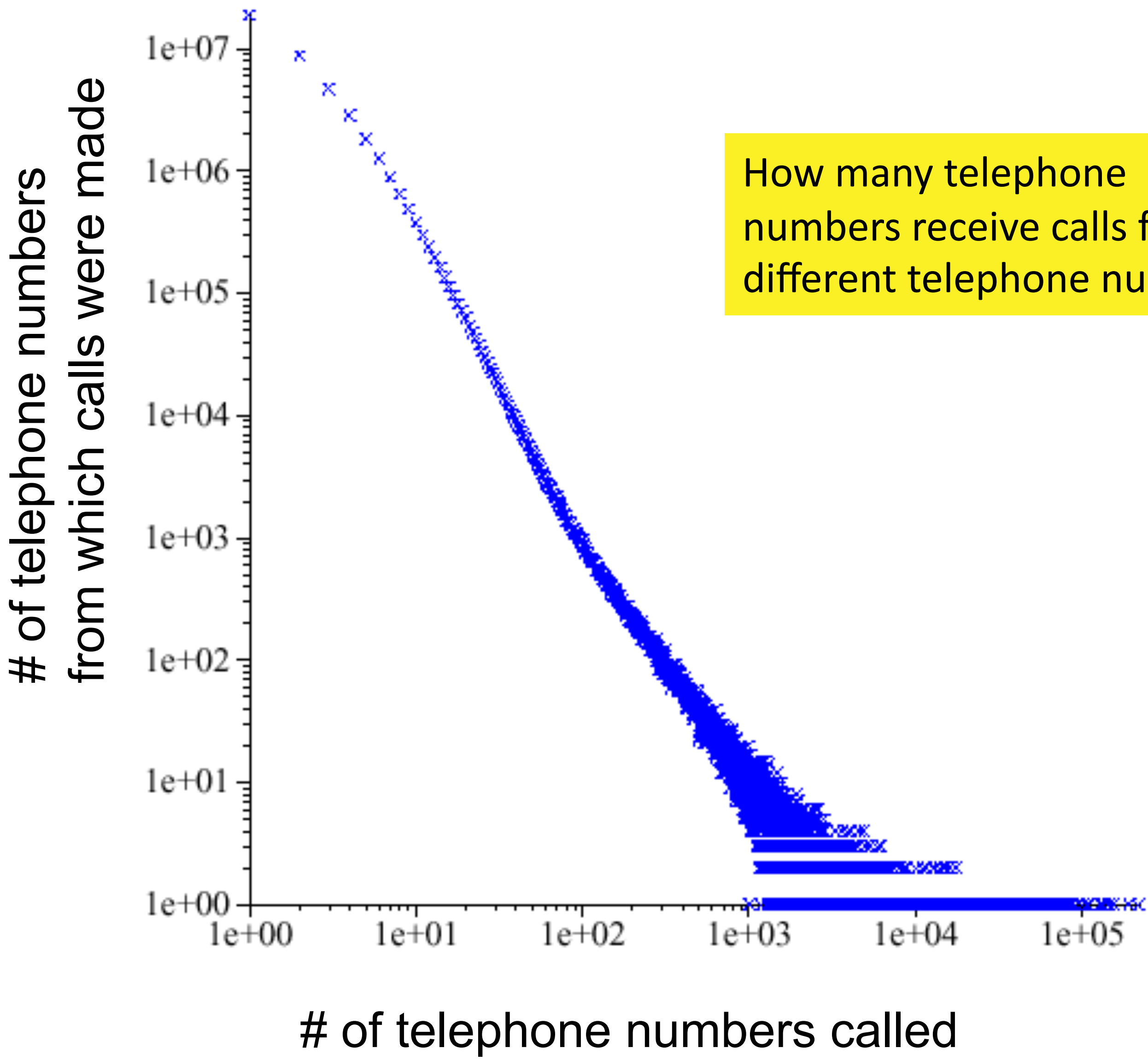
The number of nodes $N(k)$ with degree k obeys $N(k) = C.\, k^{-r}$

The primary reason appears to be the idea of "rich gets richer"

- popular web pages attract more peers

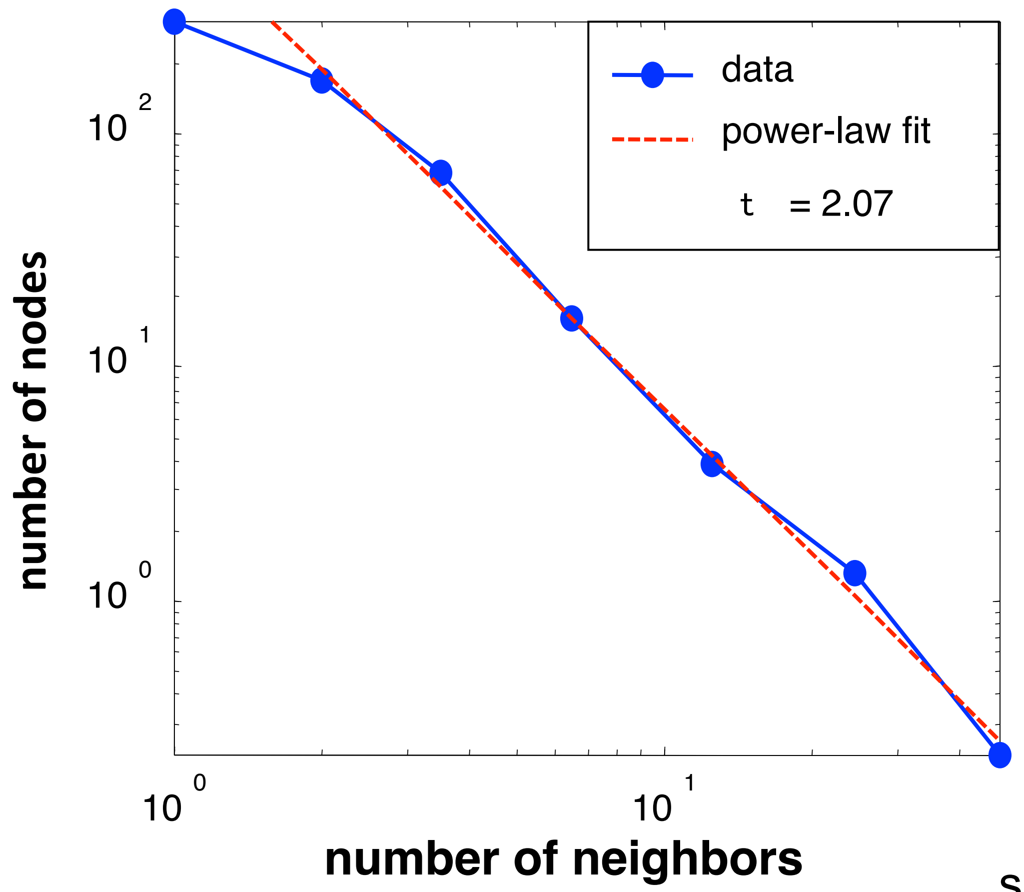
- peers prefer to connect to the well-connected nodes

AT&T Call Graph



How many telephone numbers receive calls from k different telephone numbers?

Gnutella network



summer 2000,
data provided by Clip2

power-law distribution

Search strategies

- Flooding
- Random walk /
  - Biased random walk/
  - Multiple walker random walk

    (Combined with)

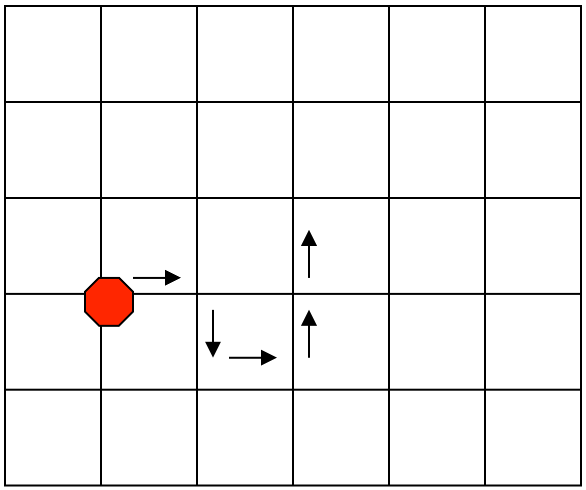
- One-hop replication /
- Two-hop replication
- k-hop replication

On Random walk

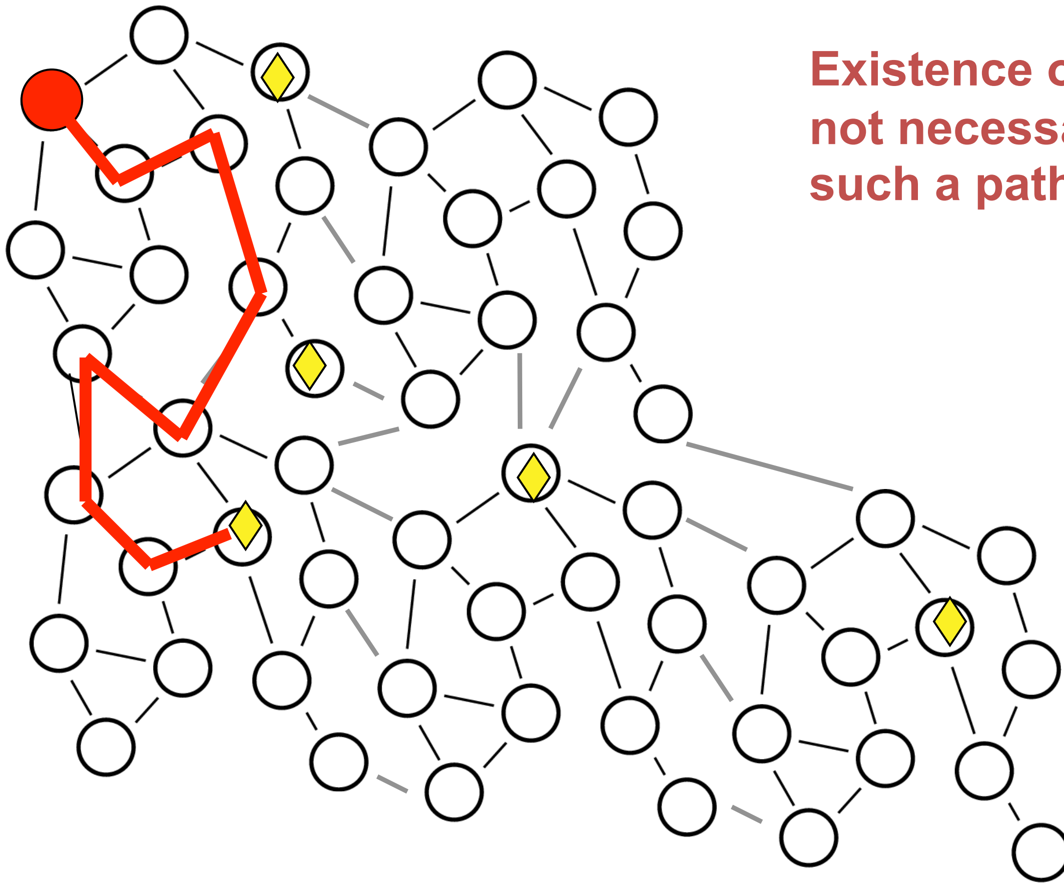
Rich history. Let p(d) be the probability that a random walk on a d-dimensional lattice returns to the origin. In 1921, Pólya proved that,

(1) p(1)=p(2)=1, but
(2) p(d) < 1 for d > 2

There are similar results on two walkers meeting each other via random walk



Search via random walk



Existence of a path does not necessarily mean that such a path can be discovered

Search via Random Walk

Search metrics

Delay = discovery time in hops

Overhead = total distance covered (i.e. total nodes visited by the walker)

(Both should be as small as possible).

For a single random walker, these are equal.

K random walkers (K>1) leads to a smaller delay

For search by **flooding**, if delay = h then

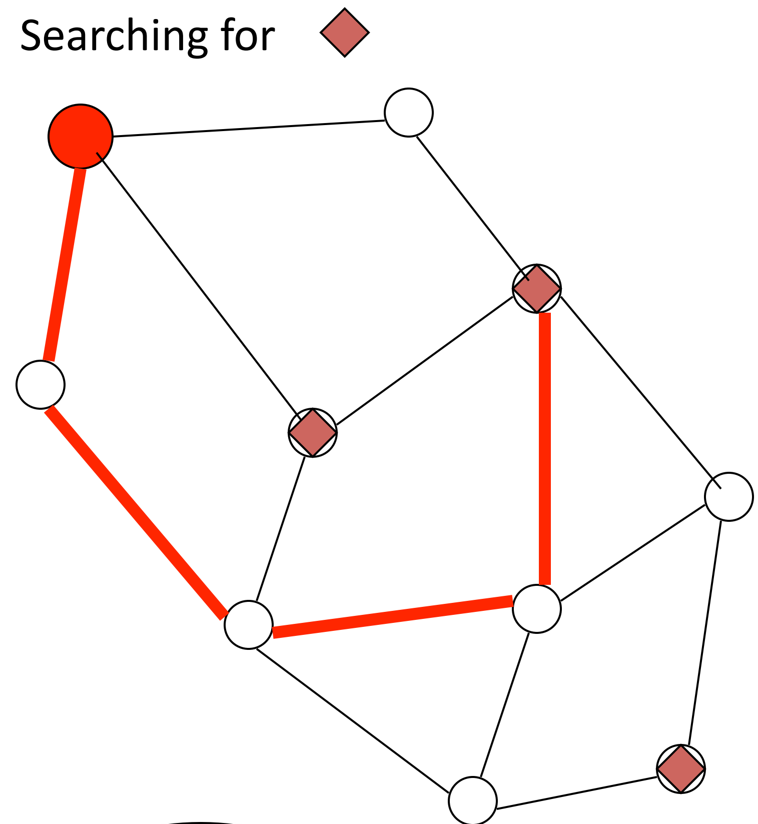
overhead $\leq d + d^2 + \ldots + d^h$ where d = max degree of a node.

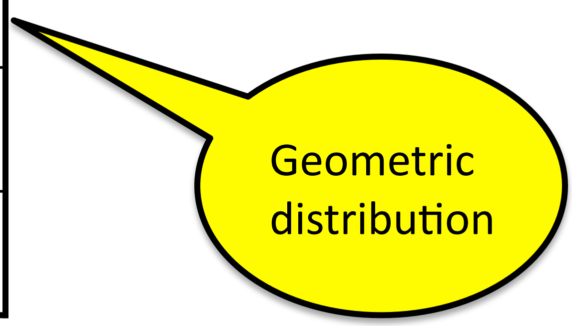
A simple analysis of random walk

Let p = Population of the object.
i.e. the fraction of nodes hosting the object

T = TTL (time to live)

Hop count **h**	Probability of success
1	p
2	$(1-p).p$
3	$(1-p)^2.p$
T	$(1-p)^{T-1}.p$





A simple analysis of random walk

Expected hop count E(h) =

$$1.p + 2.(1-p).p + 3(1-p)^2.p + \ldots + T.(1-p)^{T-1}.p$$

$$= 1/p. (1-(1-p)^T) - T(1-p)^T$$

With a large TTL, **E(h) = 1/p**, which is intuitive.

With a small TTL, there is a risk that search will time out before an existing object is located.

K random 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.

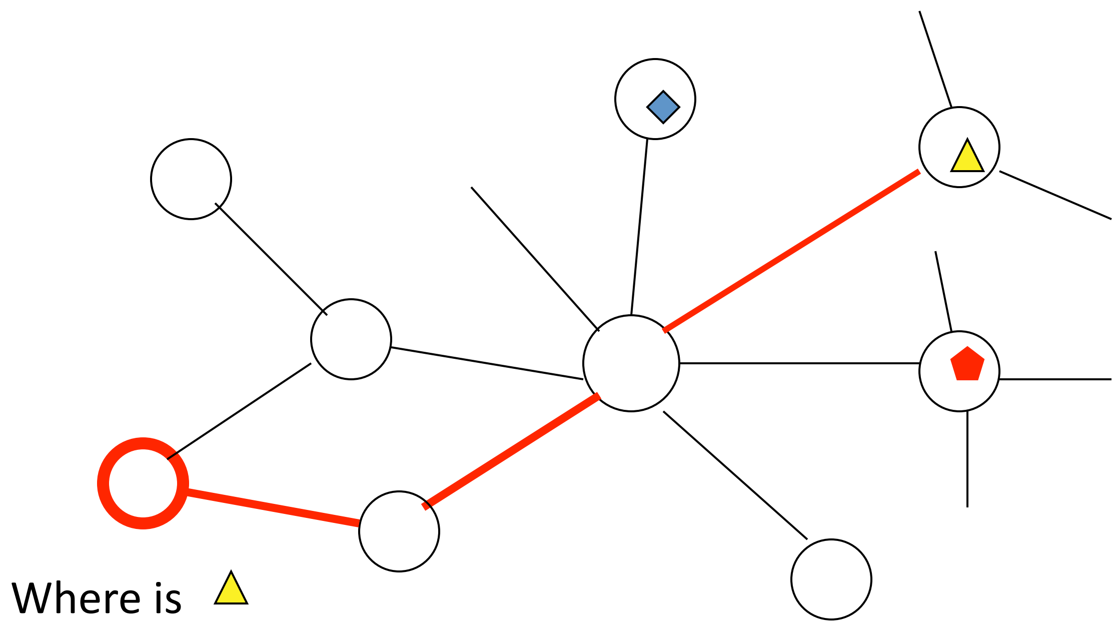
Increasing search efficiency

Major strategies

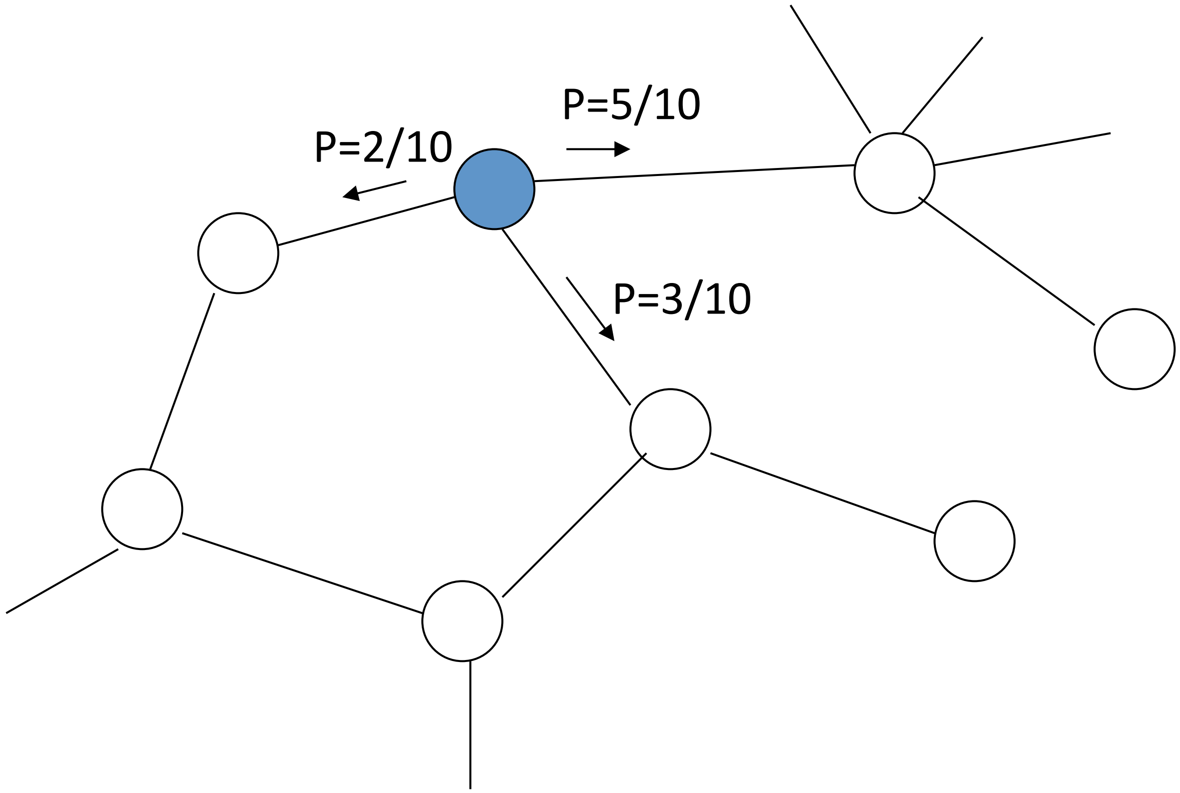
1. Biased walk utilizing node degree heterogeneity.
2. Utilizing structural properties (power-law property)
3. Topology adaptation for faster search
4. Introducing ultrapeers (i.e supernodes) in the graph

One hop replication

Each node keeps track of the indices of the files belonging to its immediate neighbors. As a result, high capacity / high degree nodes can provide useful clues for a large number of search queries.



Biased random walk



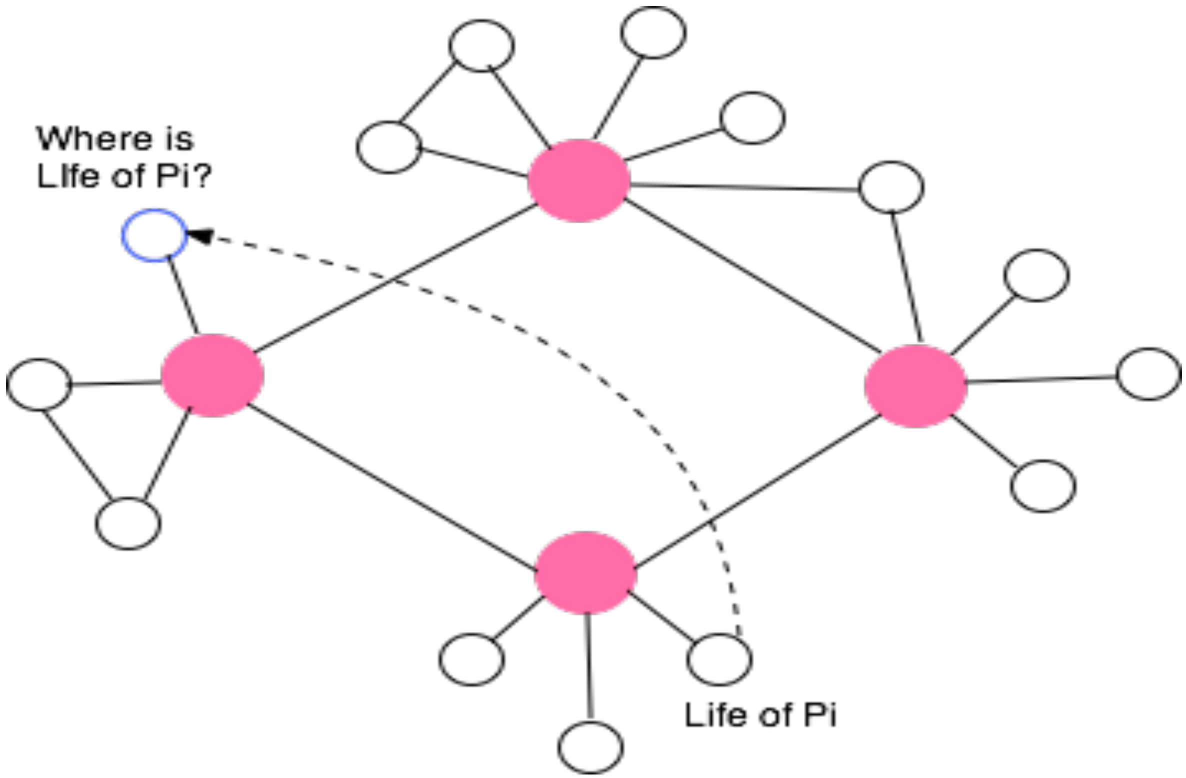
Each node records the degree of the neighboring nodes. Search **gravitates towards high degree nodes** that hold more clues.

Ultrapeers or supernodes

To overcome the scalability problem, some resource-rich nodes were given the status of as ultrapeers or supernodes. Search requests (and responses) by edge nodes are handled by the closest ultrapeer, which served as local index servers. This scaled down the decentralization.

Used by KaZaA, Limewire and many subsequent clients.

Ultrapeers or supernodes



Two-layered architecture reduces search time