Introduction to Building the **IoT** with **P2P** & ☁️

Niels Olof Bouvin

Overview

- **Introduction to the course**
- Introduction to Peer-to-Peer networking
- Client/server compared to Peer-to-Peer
- P2P characteristics
- Gnutella
- k-Walkers
- Gia
- Summary

Course Overview

- **Blackboard (¯_(ツ)_/¯)**
  - (send me an email <bouvin@cs.au.dk>, if you are not on the list)
- **Course material**
  - papers & technical reports found in the BB system
  - *Building the Web of Things* by Dominique D. Guinard & Vlad M. Trifa, as well as some chapters from their free book *Using the Web to Build the IoT*
- **Group work (3-4 persons)**
  - First half of the course: Mandatory assignment
  - Second part of the course: Self-determined IoT/P2P project
- **Exam: Oral, 30 minutes, known questions & project**

Purpose of Course

- To familiarise you with decentralised sensing systems
- To introduce a number of design criteria for P2P as well as Web-based Internet of Things networks
- To teach you to assess the strengths and weaknesses of a given system, based on these criteria
- To establish practical knowledge of IoT/P2P networking by constructing a Web based sensing system with resilient decentralised storage from scratch, and, based on these gained skills, create your own project system

Topics

- Introduction to P2P
- Structured P2P systems
- MANET
- Security & Privacy
- P2P Applications
- BitTorrent
- IoT Applications
- Introduction to IoT
- Introduction to WoT
- Embedded systems
- Networks for IoT
- Web Things
- Discovery
- Security
- Cloud, IoT, and P2P
- P2P Streaming
- The Blockchain
- Distributed Web Platforms

Administrativia

the creation of groups

- **Divide yourself into groups (3-4 persons)**
  - create a matching group using the magic of Blackboard
- **Progress on mandatory assignment is to be presented to me during office hours (starting in week 37)**
  - Thursday + Friday, depending on number of groups
  - I'll create a Doodle next week for scheduling

Mandatory Assignment

- **You will create, from scratch, a system that**
  - creates a robust network of many peers using a structured P2P network topology
  - builds a resilient storage service on top of this network
  - integrates physical devices into a Web based network of Things
  - stores the physical measurements in your resilient network
  - and provides a rich interface to inspect state and history of sensed data and devices
- **Used technologies**
  - RESTful communication between peers (Node.js is used in the book)
  - Raspberry Pi (sensor kit being available for sale next week: 200,- ***only*** MobilePay)
- **Starting next week!**

Project Work

- **Starts as soon as you finish the mandatory assignment**
  - but ideally beginning after the autumn break (week 42)
- **You are free to choose any topic, provided that**
  - there is a strong element of IoT or P2P in your proposal (and that I approve it)
  - no restrictions on technology or choice of frameworks (as long as you make a Δ)
- **You will be expected to build a system, posit hypotheses, perform experiments, and reflect and conclude upon them**
  - in the form of a written report and an oral defence
- **Show'n'tell: Demonstration of your system before all**

Overview

- *Introduction to the course*
- **Introduction to Peer-to-Peer networking**
- Client/server compared to Peer-to-Peer
- P2P characteristics
- Gnutella
- k-Walkers
- Gia
- Summary

Defining Characteristics for P2P

- Resources are shared directly between peers
- Activities are (largely) coordinated between peers
- The peers are capable of handling contingencies

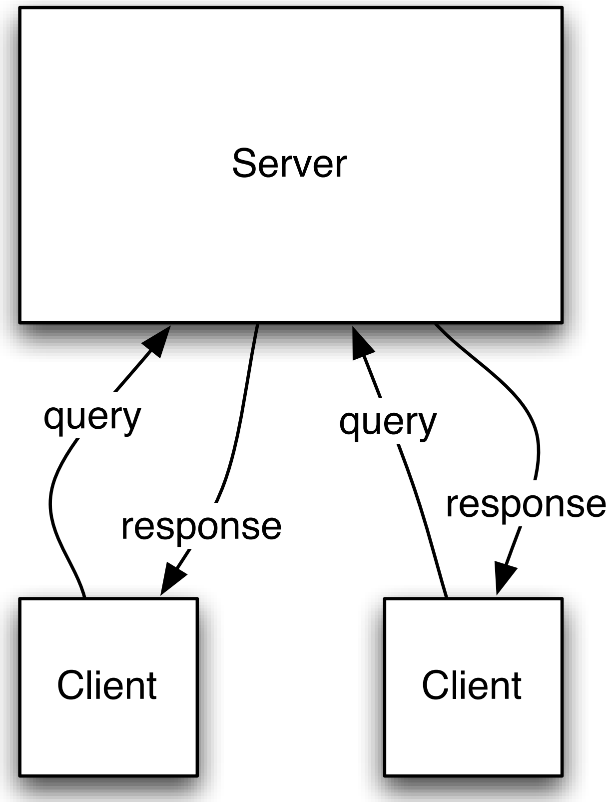
A Brief History of P2P Computing

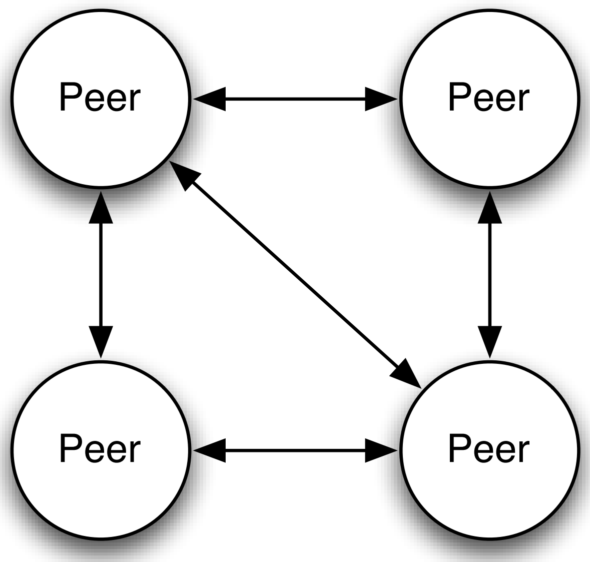
- **1969-1995: The original Peer-to-Peer Internet**
  - No firewalls, most services widely available
  - Usenet: based on Unix-to-Unix-CoPy. DNS: hosts are clients and servers, cache replies
- **1995-2001: The Internet explosion (and implosion)**
  - Movement away from P2P to client/server models
  - Web, firewalls, ADSL, asymmetric connections, NAT, ...
- **2001-...: New wave of peer-to-peer**
  - separating authoring from publishing; (Web) service oriented Internet; distributed media publishing; BitTorrent; P2P streaming
- **Now and onwards:**
  - The rise of the internet connected device/sensor: Will the edge overwhelm the center?

Overview

- *Introduction to the course*
- *Introduction to Peer-to-Peer networking*
- **Client/server compared to Peer-to-Peer**
- P2P characteristics
- Gnutella
- k-Walkers
- Gia
- Summary

Client/Server vs. P2P





Client/Server

Advantages

- Centralised
- Increased security
- Control
- Easy to maintain
- Simple
- Static topology
- State kept in one place
- Simple architecture
- Scalable (only few resources on client)
- Well known and well supported
- Loose coupling between client/client

Client/Server
Disadvantages

- Single point of failure
- Scalability is costly
- Large bandwidth requirements at server
- Can be far away from clients (latency)
- State kept in one place
- Central control
- Does not take advantage of the resources of the clients
- Collaboration between clients involves the server

Peer-to-Peer
Advantages

- Robust
- Scalability
- More clients = more available resources
- Dynamic (self configuring)
- Replication
- Decentralised (autonomy)
- Peers can collaborate directly
  - if designed well with low latency due to closeness

Peer-to-Peer
Disadvantages

- Architectural complexity
- Churn: Peers joining and leaving
- Resources are distributed and not always available
- More demanding of peers
- New technology: abstractions, techniques, etc., are not as mature

Client/Server vs. P2P

In Practice

- **No need to pick only one, when you can use both**
- **Most successful P2P systems incorporate client/server elements**
  - often for bootstrapping purposes
- **Cloud-based servers alleviates scalability concerns (though you still have to pay, so maximising work at the edges makes sense)**

Overview

- *Introduction to the course*
- *Introduction to Peer-to-Peer networking*
- *Client/server compared to Peer-to-Peer*
- **P2P characteristics**
- Gnutella
- k-Walkers
- Gia
- Summary

Classes of P2P Architectures

- **Purely decentralised architectures**
  - All peers have the same basic capabilities and offer similar services
- **Partially centralised architectures**
  - Some, usually more powerful and/or well connected, peers will accept more demanding roles on an ad hoc basis
- **Hybrid decentralised architectures**
  - Some central servers facilitate coordination

Degrees of P2P Structure

- **Unstructured networks**
  - Peers connect in a more or less haphazardly way – resulting in a network graph either power-law or random. Routing/searching is ad-hoc or based on heuristic
- **Semi-structured networks**
  - While the network is still relatively random, resources are placed so that efficient routing works
- **Structured networks**
  - Peers and resources are placed according to a rigidly defined schema, which is maintained over the lifetime of the network

P2P Characteristics
Scalability

- **The ability of a system to support an increasing use**
- Pro: Network, storage, computational power of peers may be leveraged
- Con: Routing, location, synchronising may not scale; “fat” clients needed; peers must contribute

P2P Characteristics
Performance

- **The time it takes for a system to react to a stimulus**
- Pro: Data and computation may be close to peers, high degree of distribution
- Con: Replicated, distributed state and computation; complex architectures

P2P Characteristics
Availability

- **The part of the deployment period during which a system can deliver the services it implements**
- Pro: No single point of failure/robustness; system may be self-configuring, replicated, autonomous
- Con: Ensuring consistent availability; having knowledge of network state

P2P Characteristics
Fairness

- **Distributing work equally across the peers according to their needs and abilities**
- Pro: Necessary in order to maintain the good performance of P2P
- Con: Difficult to ensure

P2P Characteristics

Integrity and authenticity

- **The ability of a system to maintain correct state**
- Pro: State is distributed, so it can not *all* be corrupted
- Con: Cryptographically authenticated security more difficult to establish without central authority

P2P Characteristics
Security

- **The degree to which a system can withstand attacks**
- Pro: Robustness against Denial of Service attacks; anonymity
- Con: Complex, decentralised security architecture

P2P Characteristics

Anonymity, deniability, censorship resistance

- **Being able to retrieve or publish information without risk of discovery**
- Pro: Adds security, difficult to suppress information
- Con: Not easy to ensure, what if running the system becomes a crime? Should *all* information be freely and anonymously available?

Overview

- *Introduction to the course*
- *Introduction to Peer-to-Peer networking*
- *Client/server compared to Peer-to-Peer*
- *P2P characteristics*
- **Gnutella**
- k-Walkers
- Gia
- Summary

Gnutella

- **The first major truly distributed P2P file sharing system – a counterpoint to the SPoF of Napster**
  - Gnutella is fully distributed and cannot be easily be taken out by an attack (legal or otherwise)
- **Invented by Justin Frankel & Tom Pepper of Nullsoft**
  - most famous for creating WinAmp
- ***Very* quickly pulled by AOL/Time Warner**
  - at that point the source was "in the wild", and a number of Gnutella variants have since developed
- **Quite primitive system, yet hugely successful**

Gnutella protocol:
5 commands are all you need

- **Ping**
  - used for discovery
- **Pong**
  - the response to a Ping
- **Query**
  - used for searching
- **QueryHit**
  - the response to a successful query
- **Push**
  - used to get fire-walled servents to reach outside the firewall

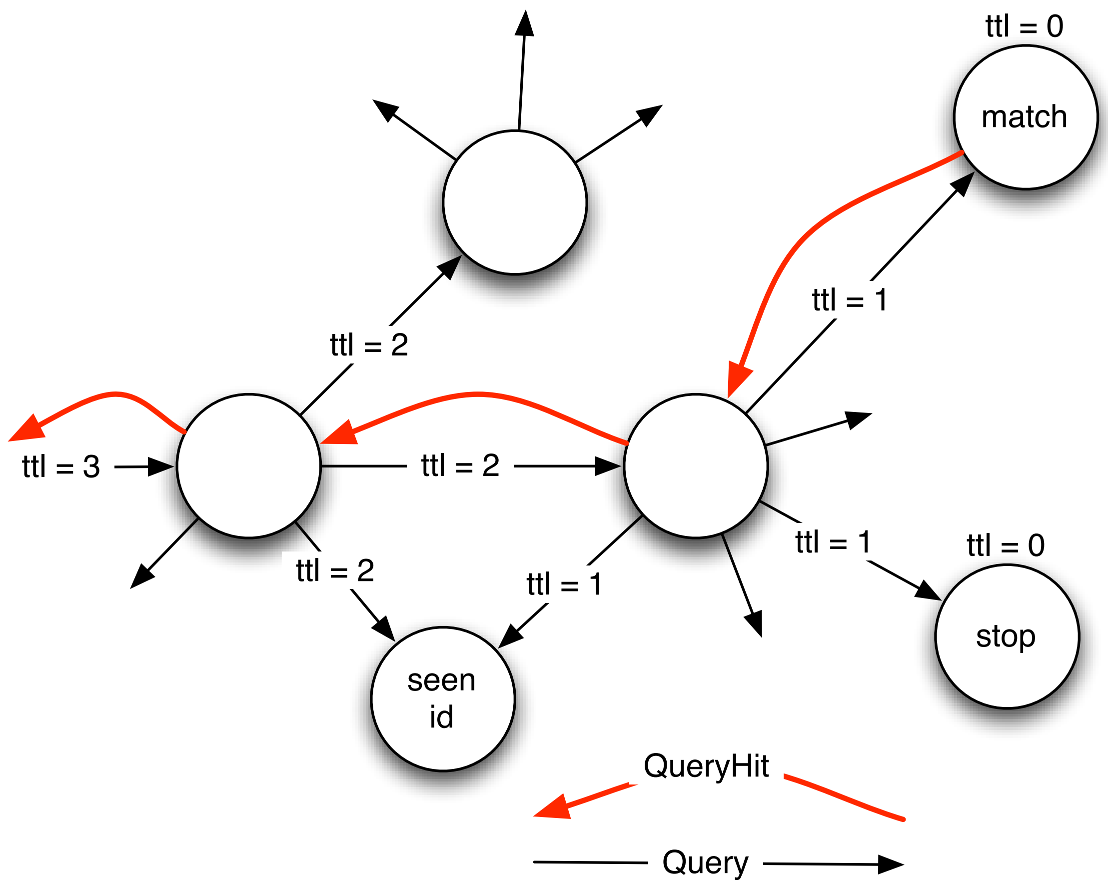
How does it work?

- A Gnutella peer starts out with a number of peers from “somewhere” (perhaps found on a Web page)
- It can ping these peers to receive information about them, and thus build a list of potential peers to contact in the future
- Pings and queries are sent to all known peers who in turn call all their peers and so on (flooding)
- Queries has a unique ID (128 bit) and a TTL (Time To Live). This ensures that peers do not retransmit the same query twice and that queries eventually die out

How does it work?

- Peers remember (for a limited time) received and transmitted queries and whence they came
- If a query match is found, the response (containing the query and the host address) is returned following the query route back to the originator
- The originator receives (presumably) a number of hits and can then contact a host directly for downloading (usually through HTTP)

A Gnutella Example



Ranking of Gnutella peers

- **Peers report**
  - amount of shared data
  - available bandwidth
- **Self-reporting is problematic**
  - claim your bandwidth is low, and you will be left alone

Gnutella is inefficient

- Flooding ensures that all peers within TTL horizon are contacted
- However, flooding generates a tremendous amount of (duplicate) network traffic
- Gnutella is so inefficient, that swamping the network becomes quite likely, *even without any data traffic*

Gnutella calculations

	TTL=1	*TTL=2*	*TTL=3*	*TTL=4*	*TTL=5*	*TTL=6*	*TTL=7*	*TTL=8*
N=2	332	664	996	1328	1660	1992	2324	2656
N=3	498	1494	3486	7470	15438	31374	63246	126990
N=4	664	2656	8632	26560	80344	241696	725752	2177920
N=5	830	4150	17430	70550	283030	1132950	4532630	18131350
N=6	996	5976	30876	155376	777876	3890376	19452876	97265376
N=7	1162	8134	49966	300958	1806910	10842622	65056894	390342526
N=8	1328	10624	75696	531200	3719728	26039424	182277296	1275942400

Traffic (in bytes) generated by search for the string ‘Grateful Dead Live’ in a perfectly balanced Gnutella graph with variable TTL and #Neighbours per peer

Gnutella experiences

- Flooding hardly the most efficient use of network resources
- Downloads the whole file from a single peer
  - So if that peer goes missing in the middle of your download… so does your data
- Advantage of Gnutella: So abysmal performance, it spurred the development of a lot of improvements

Overview

- *Introduction to the course*
- *Introduction to Peer-to-Peer networking*
- *Client/server compared to Peer-to-Peer*
- *P2P characteristics*
- *Gnutella*
- **k-Walkers**
- Gia
- Summary

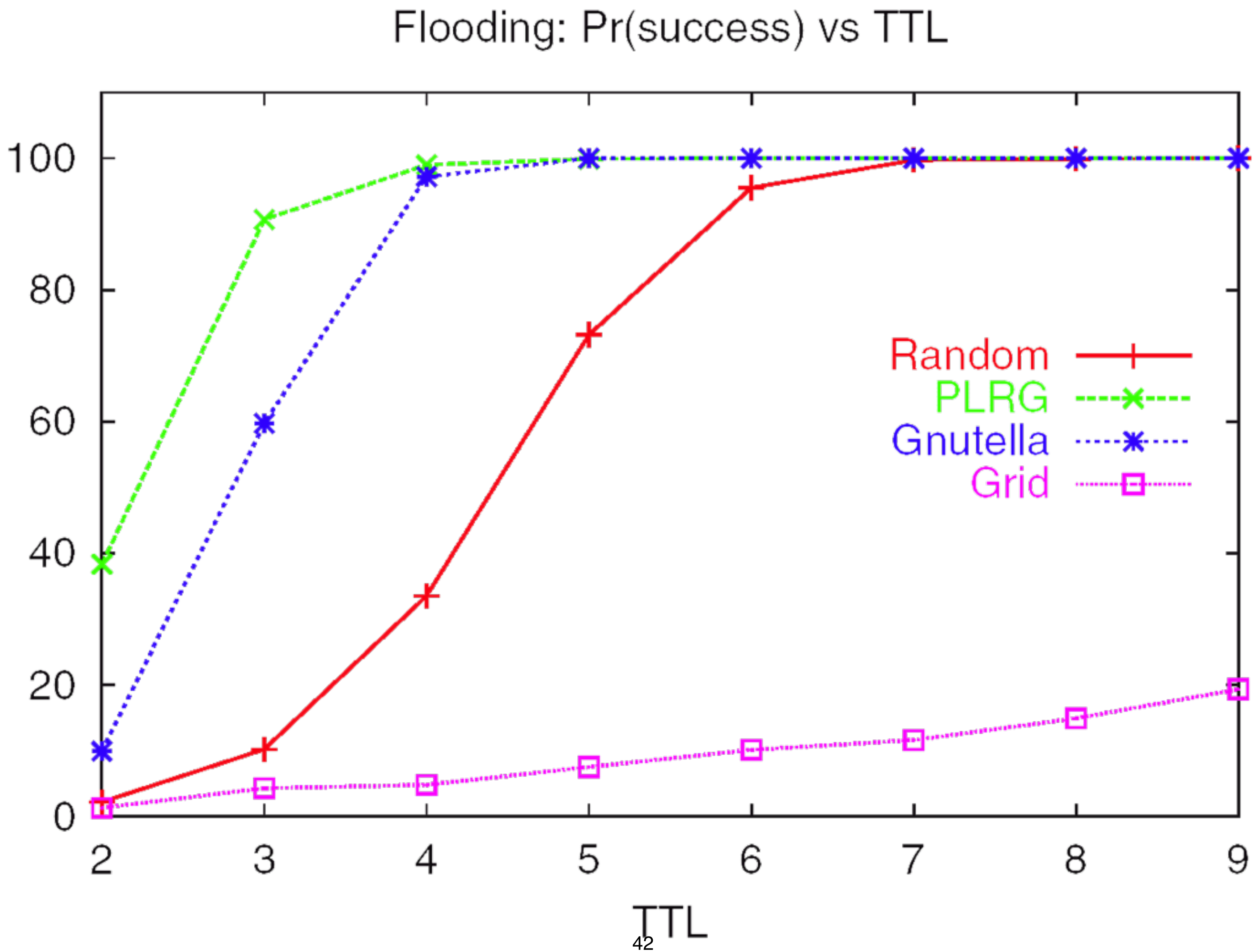
k-Walker Search in Unstructured P2P Networks

- **Aim**
  - Investigate how bad flooding is and demonstrate superior searching methods
  - Investigate how the distribution of replicates affects searching (but we will not go into that)

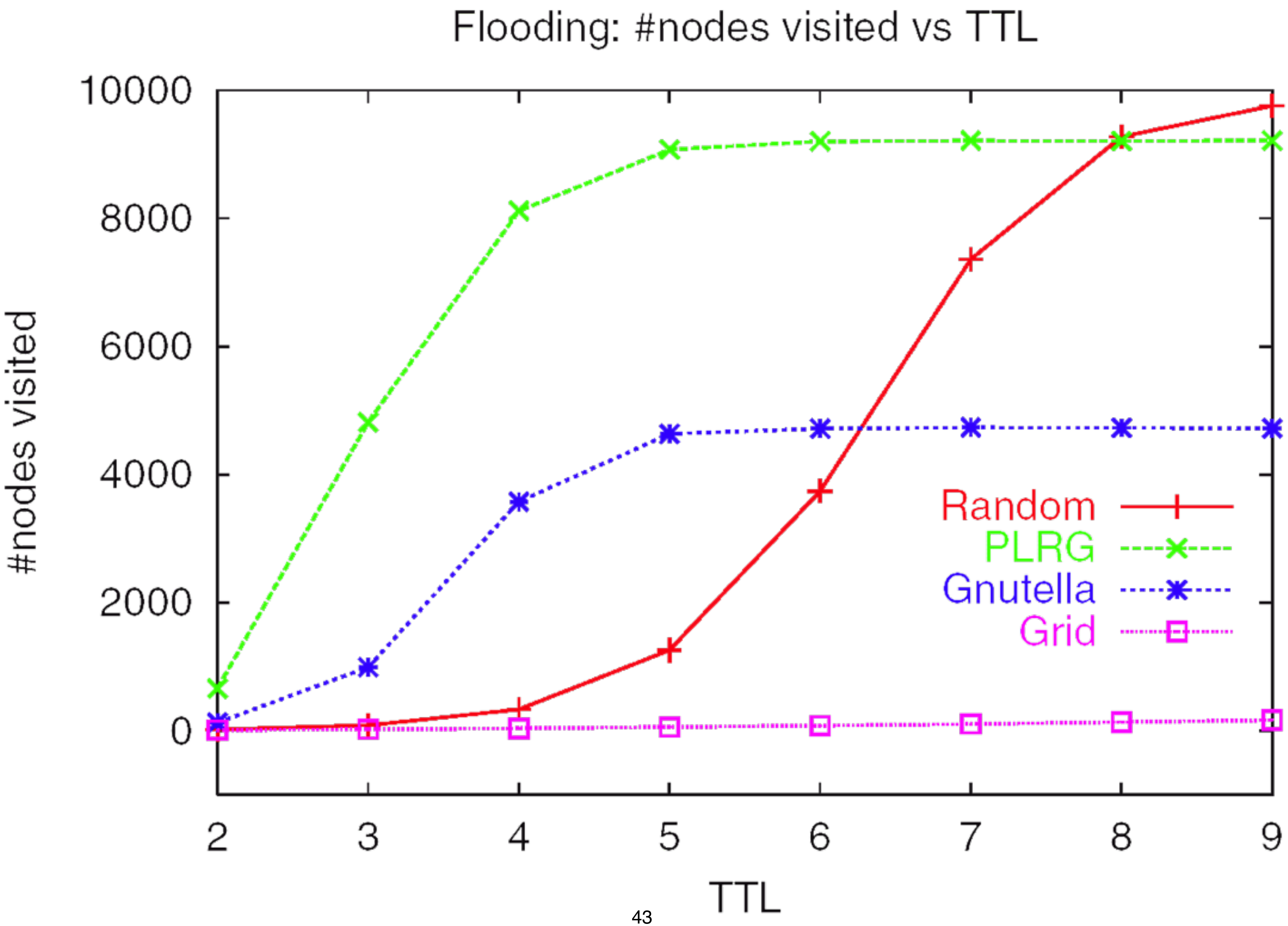
Time to Live Considered Harmful

- **Naïve Gnutella searching consists of spreading queries by flooding until TTL is exhausted**
  - if a hit is found, the flooding continues regardless everywhere else
  - if a hit is not found, a tremendous amount of messages have been sent for no good reason
  - high TTL generates a lot of traffic
  - low TTL may not locate the desired resource
- **But, as a query quickly covers a large portion of the network neighbourhood, the delay between issuing a query and receiving results is quite low**

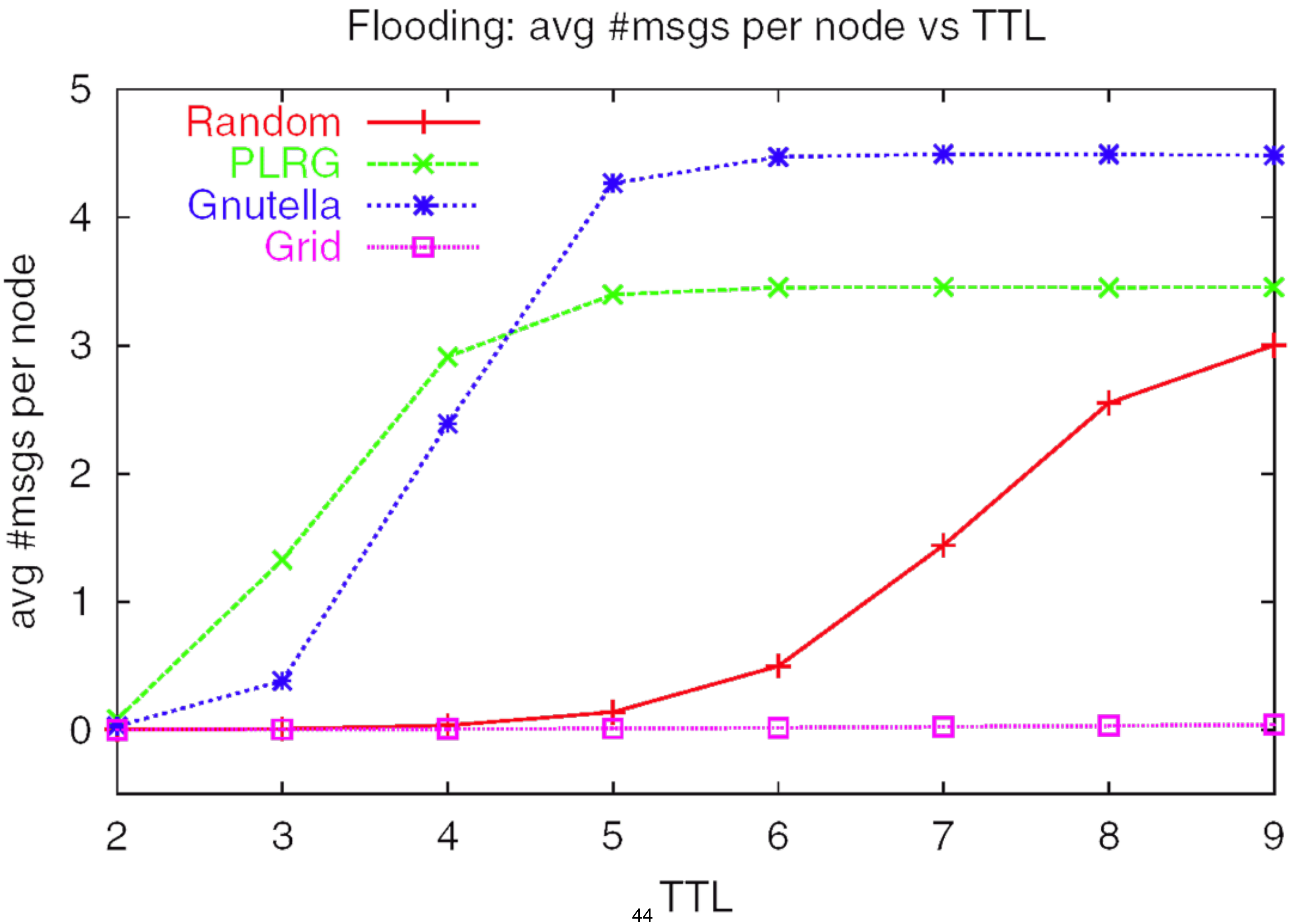
Successful Search/TTL



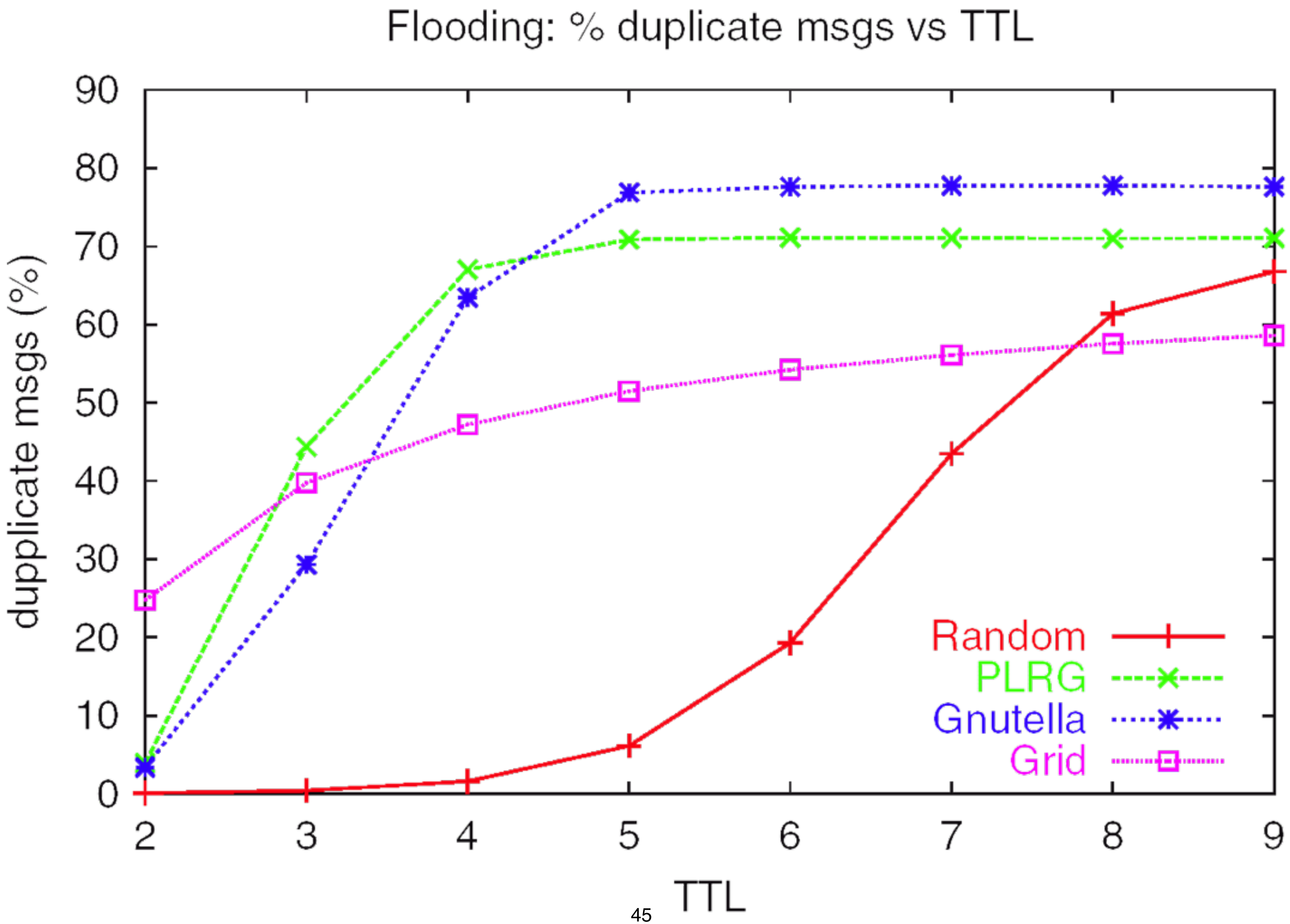
#Nodes Visited/TTL



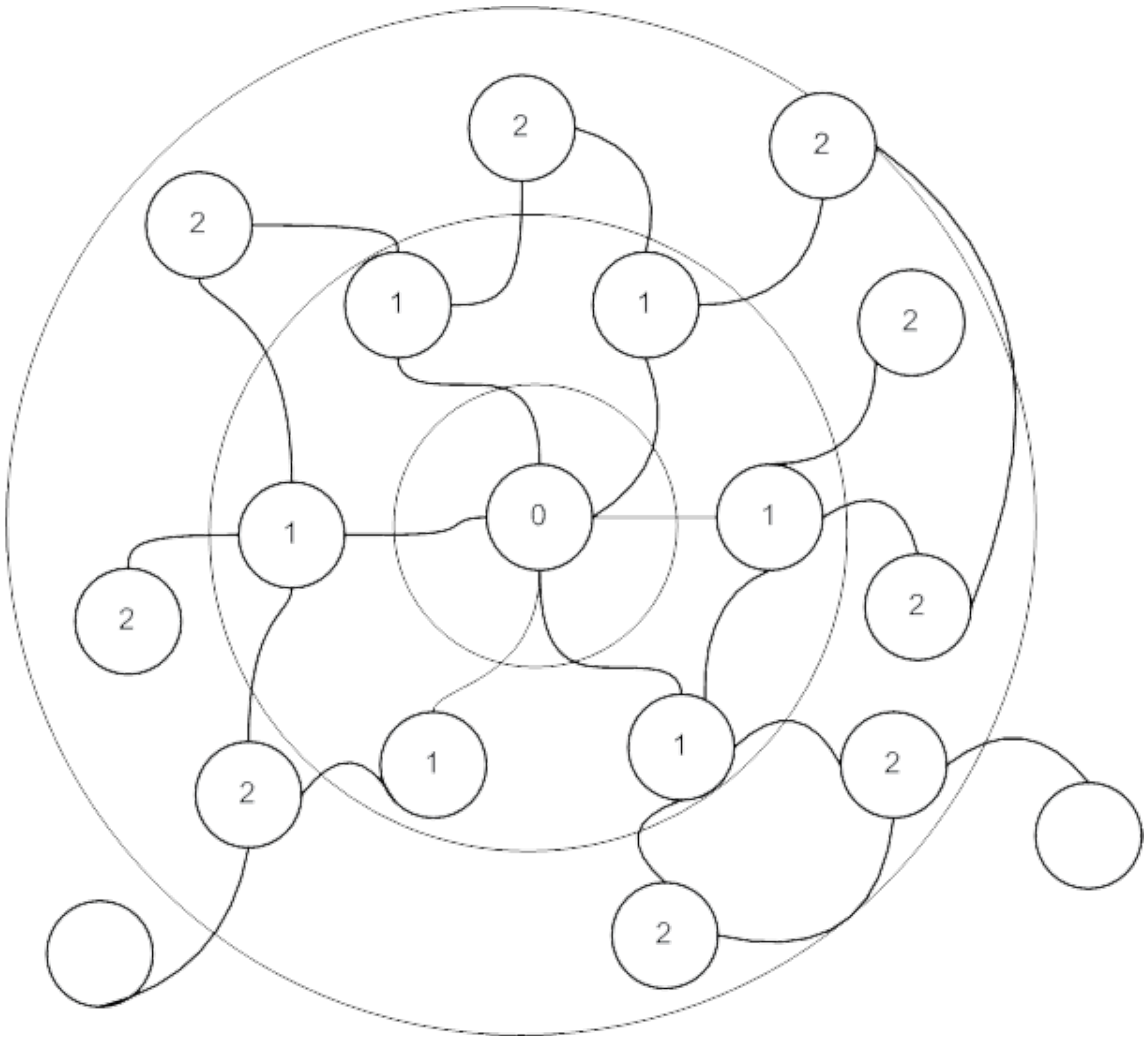
Average #Messages per Node/TTL



%Duplicate Messages/TTL



Flooding Alternative: Expanding Ring

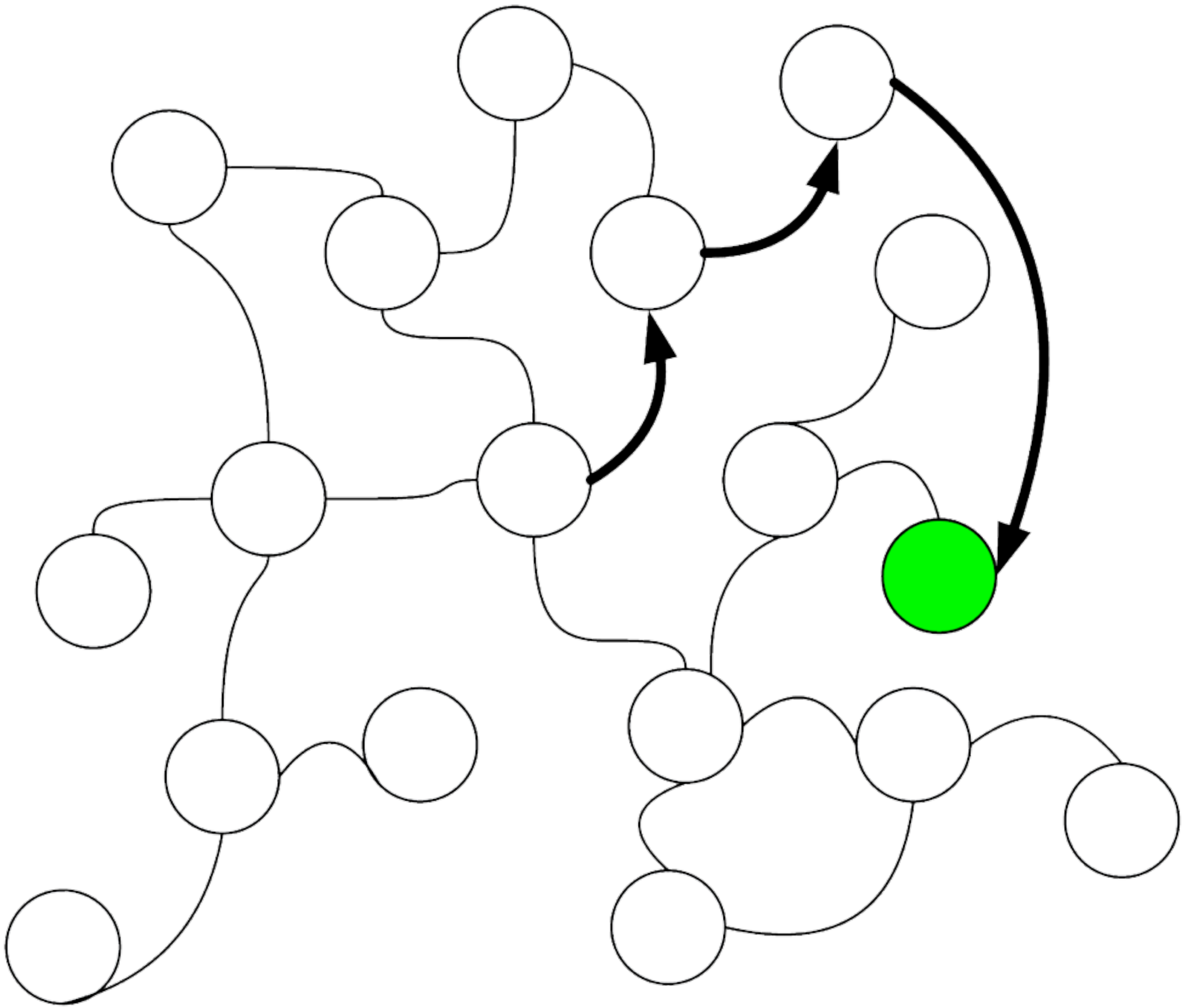


Start with small values of TTL, and increase TTL until sufficient number of hits are found

Expanding Ring

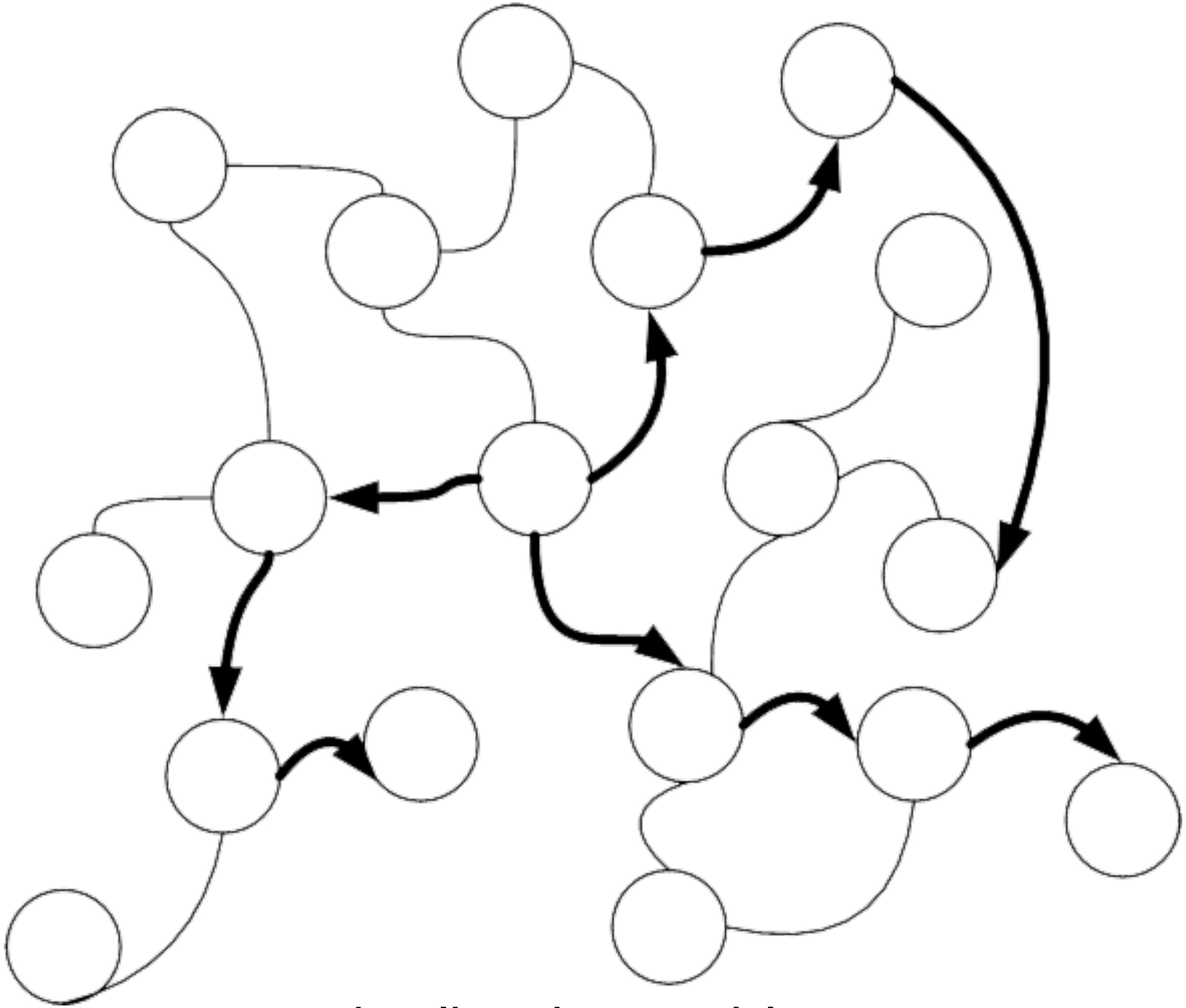
- **Advantages**
  - ultimately as successful as ordinary flooding
  - if a resource is nearby, it is located at a lower overall cost
- **Disadvantages**
  - if the resource is not found, *more* messages are generated than ordinary flooding!
  - successive searches mean longer user perceived delays

Random Walker



Depth-first search: A query transverses the network randomly until a match is found

k Random Walkers



k walkers decrease delay

Random Walkers

- **Advantage**
  - much more efficient in term of overall traffic
- **Disadvantage**
  - longer user perceived delays
- **Typical configuration**
  - TTL = 1024; $32 \leq k \leq 64$
- **Variations**
  - walker checks periodically source for sufficient success (every fourth step)
  - nodes maintain state and do not forward the same query to the same neighbour twice

Results

distribution model		50 % (queries for hot objects)			
query/replication	metrics	flood	ring	check	state
Uniform / Uniform	#hops	2.39	3.40	7.30	6.11
	#msgs per node	4.162	0.369	0.051	0.045
	#nodes visited	4556	933	141	151
	peak #msgs	64.9	6.4	1.3	1.2
Zipf-like / Proportional	#hops	1.60	2.18	1.66	1.66
	#msgs per node	2.961	0.109	0.021	0.021
	#nodes visited	3725	357	49	60
	peak #msgs	43.8	2.0	0.7	0.8

Conclusions

- **Results**
  - Random walkers scale much better than flooding – especially with regards to message duplication
  - User perceived delays are increased
  - Blindly using TTL is inefficient – queries should check back periodically
- **However**
  - Simulation assumes a stable network
  - Content/traffic may not be Zipf-distributed after all (Gummadi et al. 2003)

Overview

- *Introduction to the course*
- *Introduction to Peer-to-Peer networking*
- *Client/server compared to Peer-to-Peer*
- *P2P characteristics*
- *Gnutella*
- *k-Walkers*
- **Gia**
- Summary

An Active Topology Adaption with Biased Random Walkers

- Gia: A system combining
  - ***topology adaption*** – peers should connect to strong and well-connected peers able to handle the traffic
  - ***active flow control*** – if a peer is overloaded it should be not bothered until it is ready again
  - ***one-hop replication*** of indices – every peer knows what its neighbours store
  - ***biased random walking*** – queries seek towards high capacity peers

Gia Terms

- **Capacity**
  - ability to handle messages/time – i.e., bandwidth, CPU power, storage capacity…
- **Satisfaction**
  - 0..1: degree to which a peer's own capacity is matched by the sum of its neighbours' capacities/degree

Topology Adaption

Let C_i represent capacity of node i
if $num\_nbrs_X + 1 \leq max\_nbrs$ **then** {we have room}
  ACCEPT Y; return

{we need to drop a neighbor}
$subset \leftarrow i \ \forall \, i \in nbrs_X$ such that $C_i \leq C_Y$
if no such neighbors exist **then**
  REJECT Y; return
candidate $Z \leftarrow$highest-degree neighbor from $subset$

if $(C_Y > max(C_i \ \forall \, i \in nbrs_X)\ )$ {Y has higher capacity}
or $(num\_nbrs_Z > num\_nbrs_Y + H)$ {Y has fewer nbrs}
then
  DROP Z; ACCEPT Y
else
  REJECT Y

Add neighbours if we need them. Replace if there's someone better.
Only replace the well-connected.

Adaptive Flow-Control

- **Each peer sends tokens to its neighbours according to its (and their) capacity**
  - a peer must have a token from a neighbour in order to forward a query to that neighbour
  - if a peer is overloaded, it queues queries and reduces its token publication rate
  - tokens can be sent out separately or piggy-backed on other traffic
- **As tokens are assigned based on advertised capacity, it pays to advertise your true (high) capacity**
  - the opposite holds true in other systems – if you claim low capacity, you are not bothered by other users

One-Hop Indices Replication

- **All peers maintain indices over neighbours' resources**
  - thus all peers are able to answer queries for material held by their neighbours
  - this evens the load for peers with many resources
- **Query results contain pointers to the location of the resource – not the location of the index**
  - thus, duplicate query results are not created
- **But…**
  - what about popular content held by low ranking peers?

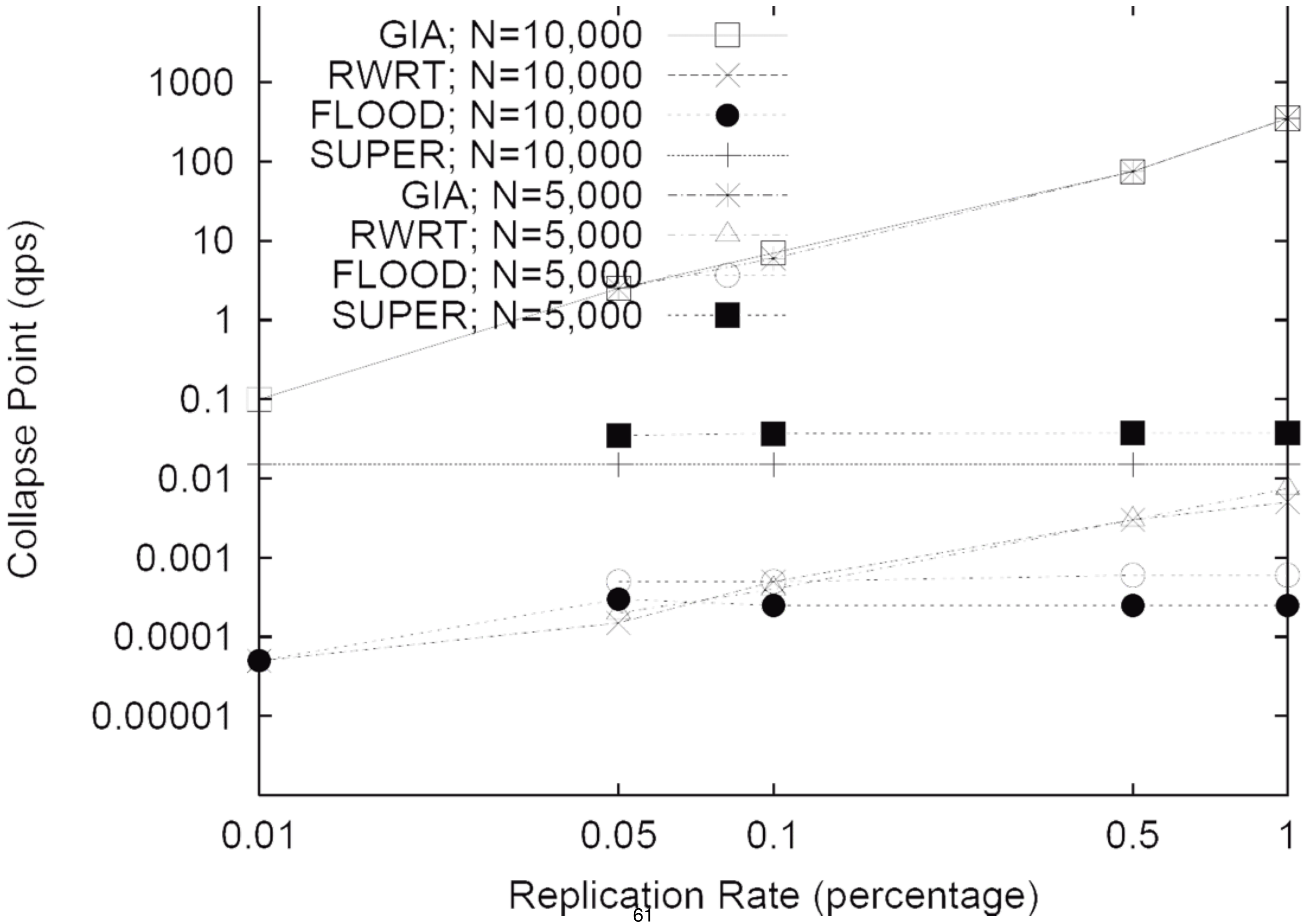
Biased Random Walker

- **Gia utilises a random walker algorithm where walkers are directed by the nodes towards the highest capacity neighbour it has tokens from**
  - queries are limited by TTL and MAX_RESPONSES
  - queries have GUIDs and are not forwarded to the same peer twice (unless the node is out of fresh neighbours)
  - queries return matches to source along query path
  - queries send keep-alive back to source (to handle network failures or rearrangements)

Gia Measurement Terms

- **Hop-count**
  - the number of hops needed to locate a resource
- **Collapse Point (CP)**
  - the point of traffic (queries) at a peer beyond which the success rate drops below 90% (because of traffic overload)
- **Hop-Count before Collapse (CP-HC)**
  - the average hop-count before the Collapse Point
  - simulation done on network with 10.000 nodes

Collapse Point



Conclusions

- A sophisticated system able to withstand high levels of traffic
- Designed with actual capacity in mind
- Many possibilities for fine-tuning and adjustment of the algorithms
- Also tested with actual computers!
- Not entirely unstructured, as neighbours are chosen carefully, but not nearly as rigid as the DHT systems (more about those next time)

Overview

- *Introduction to the course*
- *Introduction to Peer-to-Peer networking*
- *Client/server compared to Peer-to-Peer*
- *P2P characteristics*
- *Gnutella*
- *k-Walkers*
- *Gia*
- **Summary**

Summary

- **The strength of P2P is in numbers**
  - Great number of unused processors
  - Large amount of unused bandwidth
  - Whole lot of storage
- **P2P systems can be built to increase**
  - Computing power
  - Data availability
  - Free speech
- **This involves significant challenges**
  - Routing
  - Searching
  - Churn
  - Security

Summary

- **Searching in an unstructured P2P network is hard**
  - the network will change
  - not much knowledge and no central index
- **Flooding is not an efficient approach (quick, but *dirty*)**
- **Random Walkers improve considerably on the network efficiency**
- **Super node topologies recognise that peers have different capabilities**
- **Significant gains from a multi-pronged approach, affecting topology, flow, replication, and biased walking**

Summary

- **Scalability**
  - random walkers scale much better than flooding – especially with regards to message duplication
- **Performance**
  - user perceived delays are increased with walkers
  - however, increasing *k* leads to shorter delays
- **Fairness**
  - super nodes can improve performance, but should be themselves be rewarded for their extra work
  - well-connectedness is not equal to being able to handle the load

Summary

- **Integrity and security**
  - power-law and super node topologies are more vulnerable to targeted attacks
- **Anonymity, deniability, censorship resistance**
  - a hostile super node would be ideally placed to monitor or disrupt the network
  - adaptive systems can be hurt – being probabilistic helps