Should we build Gnutella on a structured overlay?

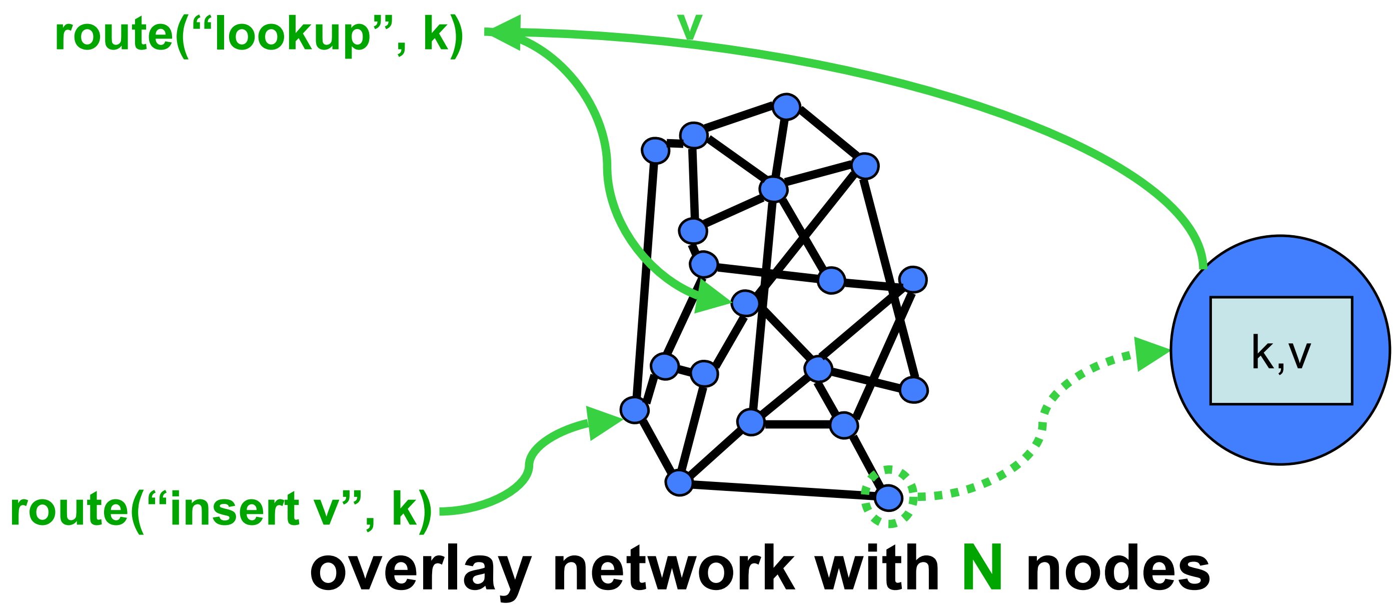
Ant Rowstron

joint work with

Miguel Castro, Manuel Costa

Microsoft Research Cambridge

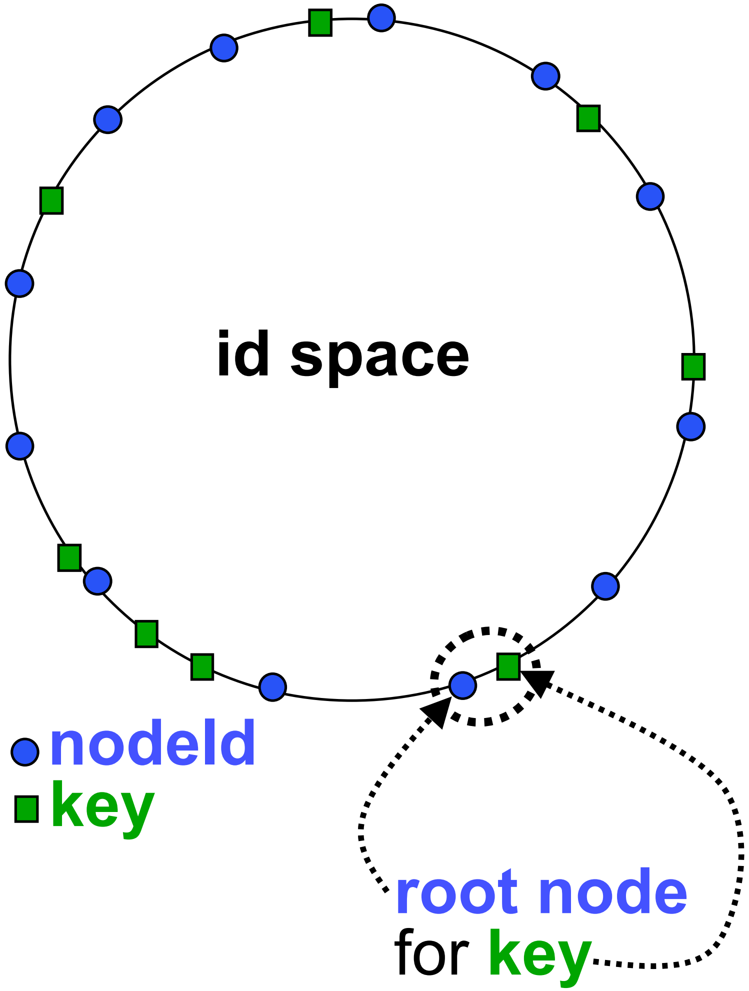
Structured P2P overlay networks



- Structured **overlay network maps keys to nodes**
- **Routes messages to keys**; (can implement hash table)

[CAN, Chord, Kademlia, Pastry, Skipnets, Tapestry, Viceroy]

Mapping keys to nodes

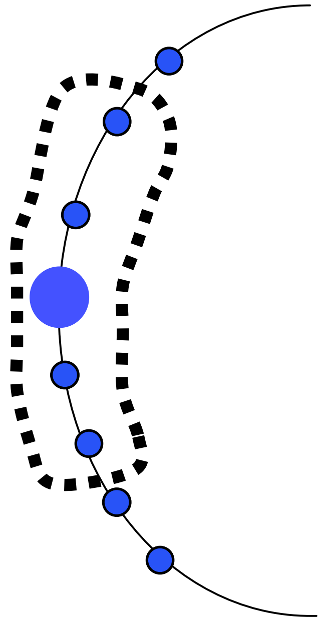


- Large **id space** (128-bit integers)
- **NodeIds** picked randomly from space
- Key is managed by its **root node**:
    - Live node with id closest to the key

Pastry

0*	1*	2*	3*
20*	21*	22*	23*
200*	201*	202*	203*
2030*	2031*	2032*	2033*

203231
nodeId



leaf set

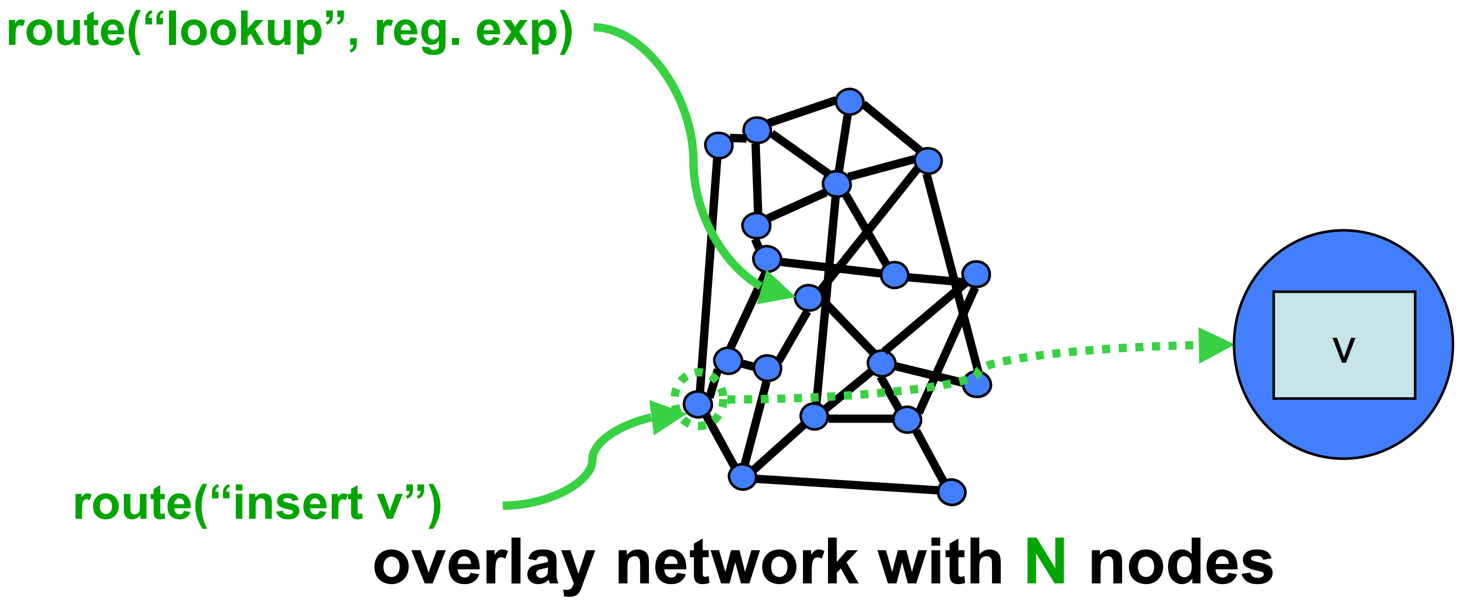
- **routing table**
  - nodeIds and keys in some base 2^b (e.g., 4)
  - prefix constraints on nodeIds for each slot

Structured overlays

- Overlay topology
  - nodes self organize into structured graph
  - node identity constrains set of neighbors
- Data placement
  - data identified by a key
  - data stored at node responsible for key
- Queries
  - efficient key lookups (O(logN))

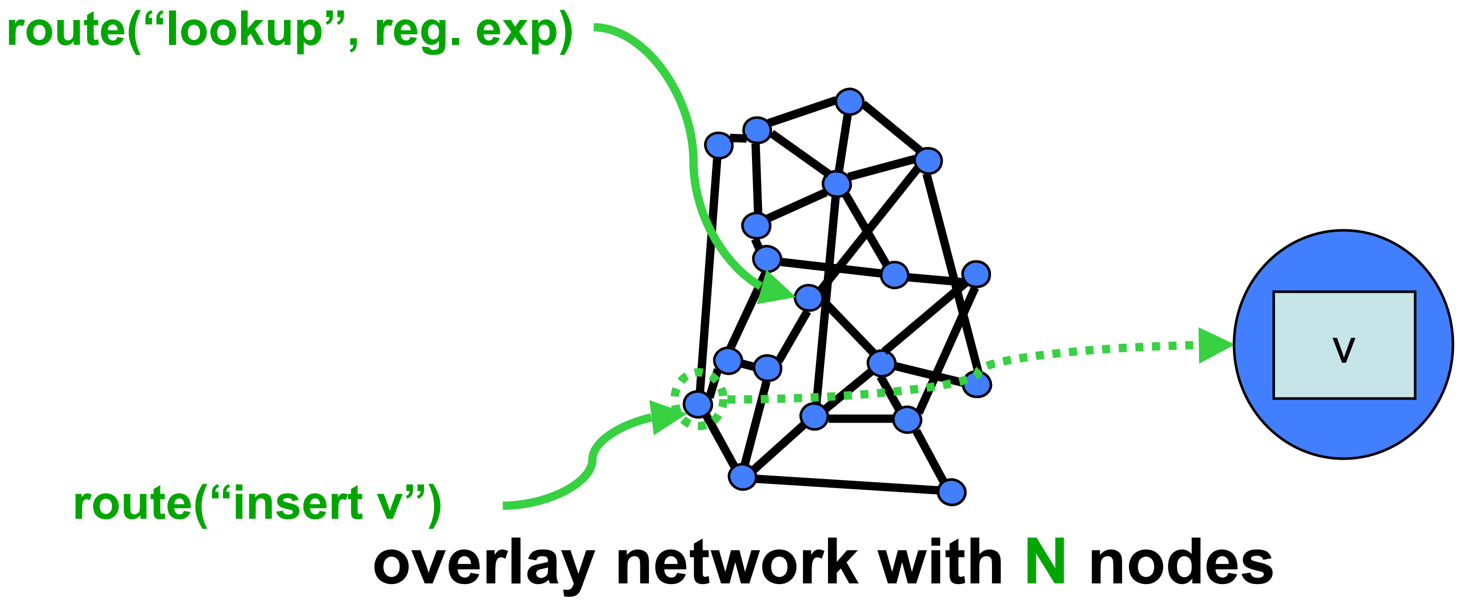
examples: CAN, Chord, Pastry, Tapestry

Gnutella



- Nodes form **random graph (unstructured overlay)**
- **Node stores its own published content**
- Lookups flooded through network (inefficient)

Gnutella



- Nodes form **random graph (unstructured overlay)**
- **Node stores its own published content**
- Lookup using random walks (needles and haystacks!)

Unstructured overlay

- Overlay topology
  - nodes self-organize into random graph
- Data placement
  - node stores data it publishes
- Queries
  - overlay supports arbitrarily complex queries
  - floods or random walks disseminate query
  - each node evaluates query locally

example: Gnutella

Can we build Gnutella on a structured overlay?

- Complex queries are important
  - unstructured overlays support them
  - structured overlays do support them

- Peers are extremely transient
  - unstructured overlays more robust to churn
  - structured overlays have higher overhead

[Chawathe et al. SIGCOMM'03]

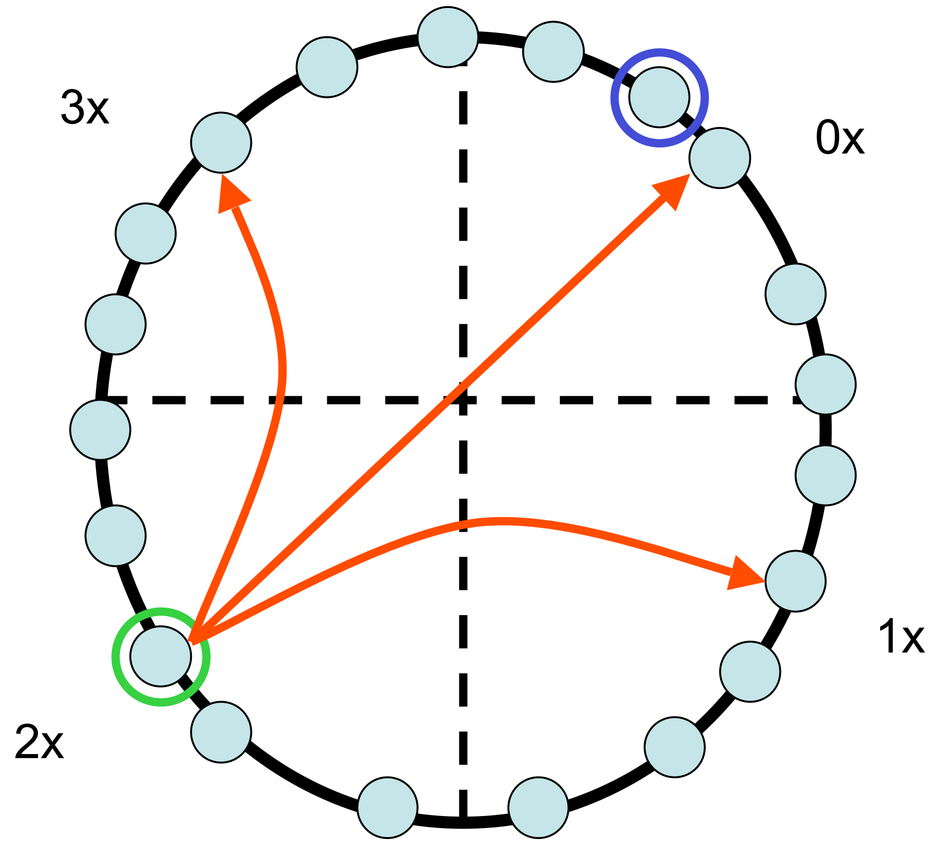
Complex queries

- Arbitrarily complex queries
  - Unstructured overlay
    - Flood
      - High overhead due to duplicates
    - Random walks
      - High lookup latency
  - Support arbitrarily complex queries
  - Structured overlays
    - ?

Complex queries (structured)

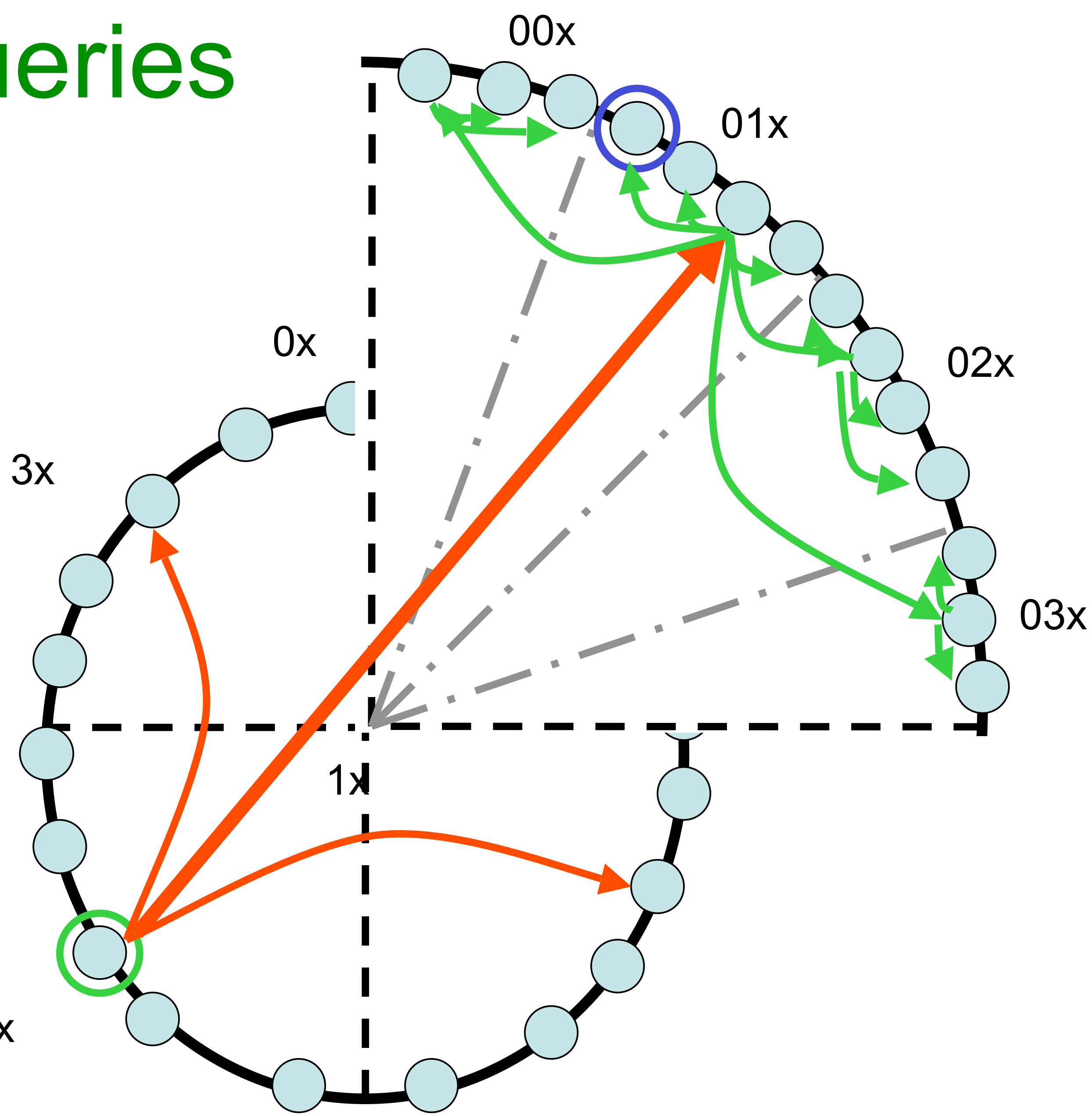
- **Structured overlay topology**
  - nodes self organize into structured graph
- Same data placement as **unstructured**
  - node stores data it publishes
- Same **queries as unstructured**
  - overlay supports arbitrarily complex queries
  - floods or random walks disseminate queries
  - each node evaluates query locally

Flood queries

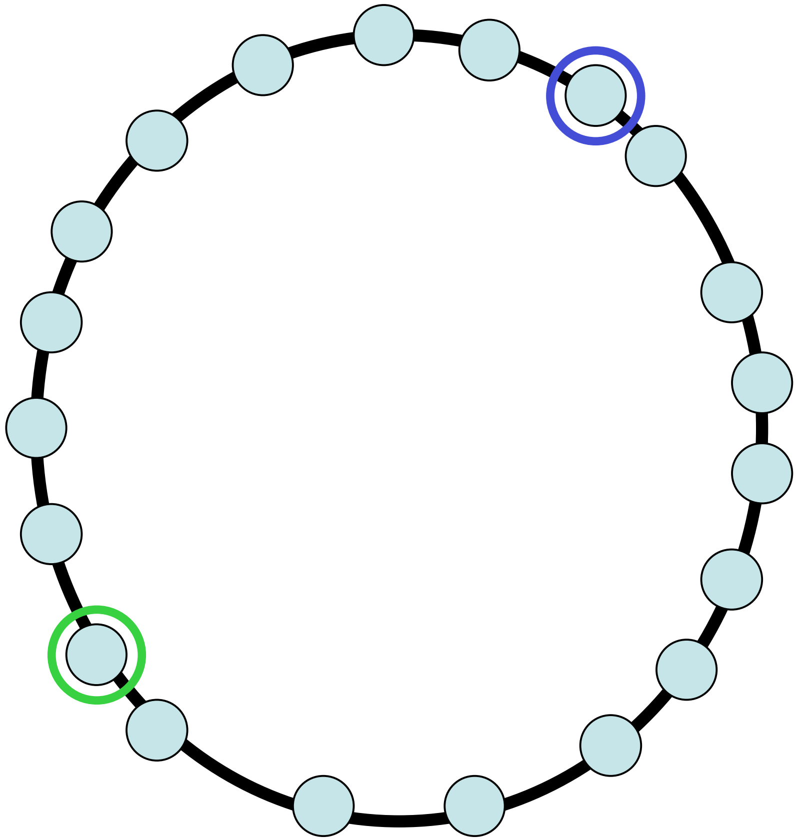


- **Exploit structure to avoid duplicates**

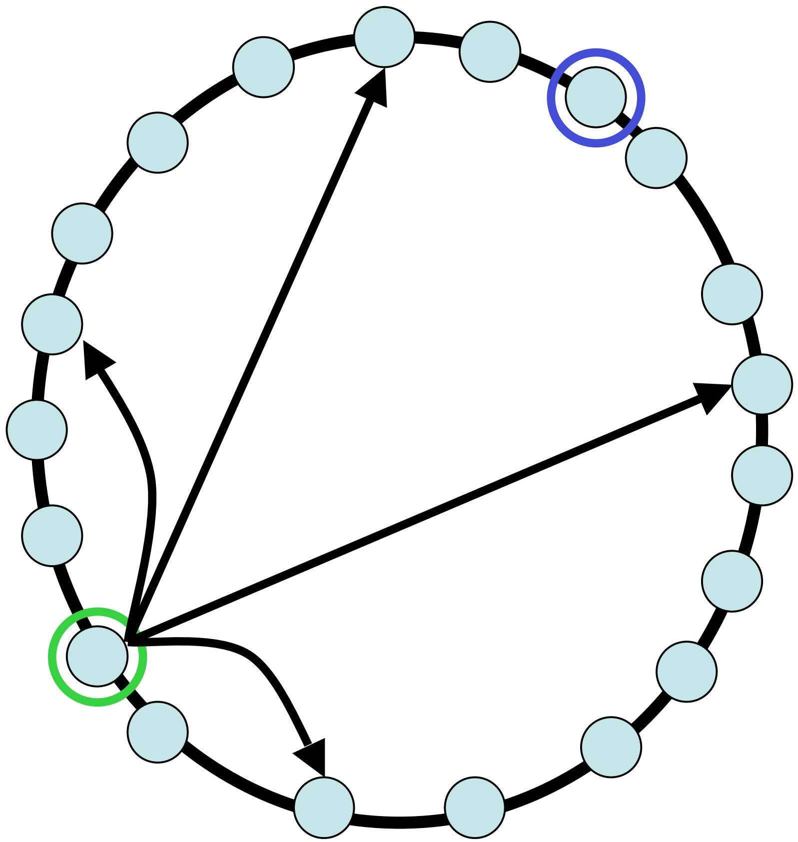
Flood queries



Random walk queries 1

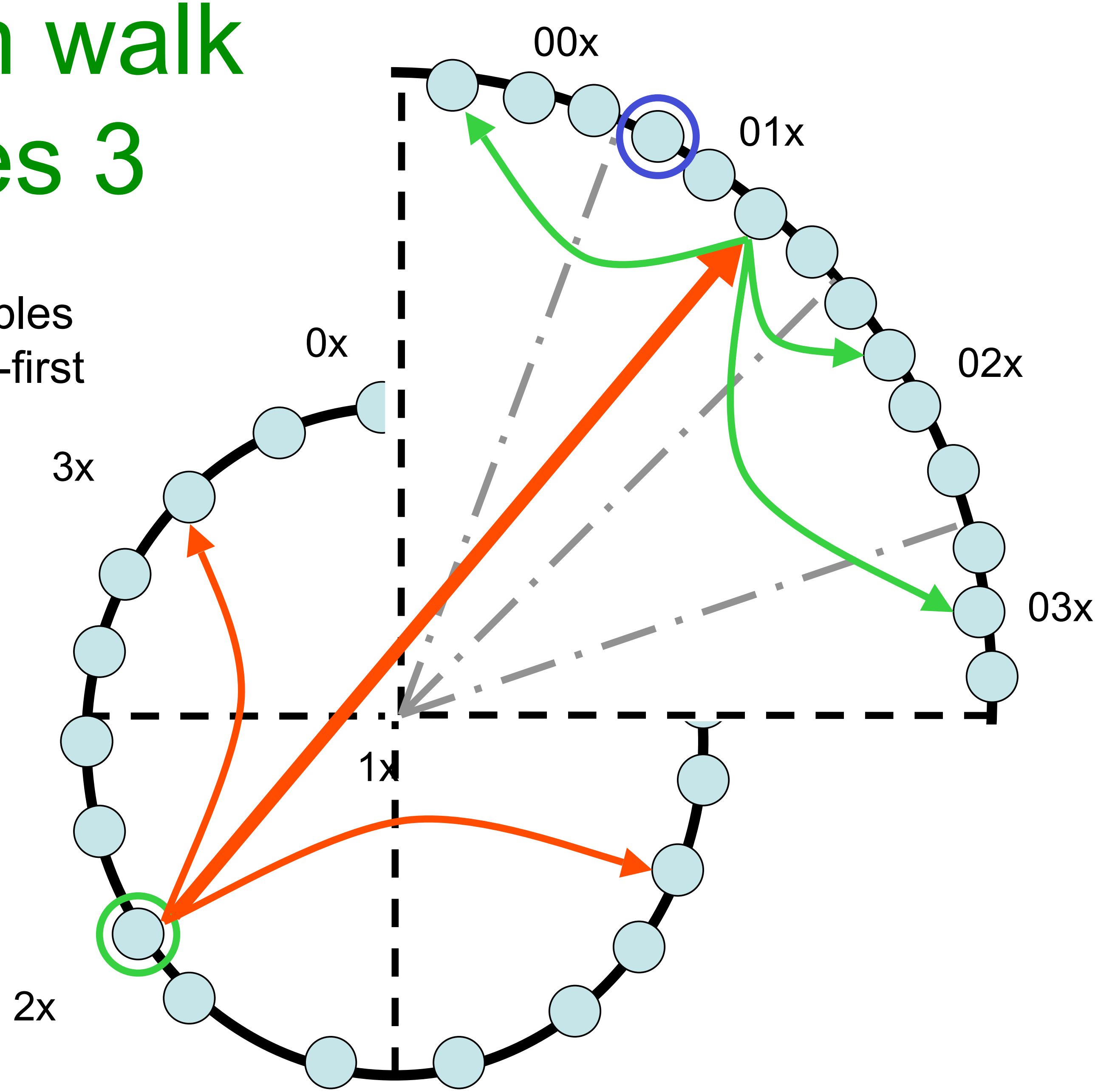


Random walk queries 2



Random walk queries 3

- Exploiting routing tables
  - Breadth-first search



Story so far….

- Gnutella is built using an unstructured overlay
- Described hybrid approach
  - Structured overlay graph
  - Unstructured overlay data placement
- Described how to exploit structure in lookup
  - Same techniques as in an unstructured overlay
  - Implemented more efficiently

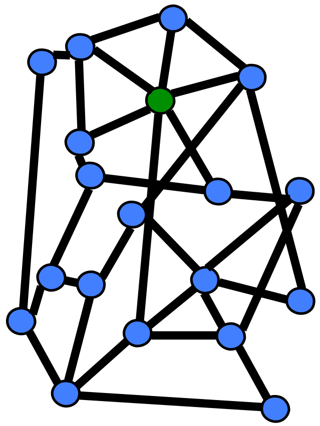
Next part: Churn and overhead

Overhead

- Both structured and unstructured
  - detect failures
  - repair overlay graph when nodes join or leave

Detecting failures

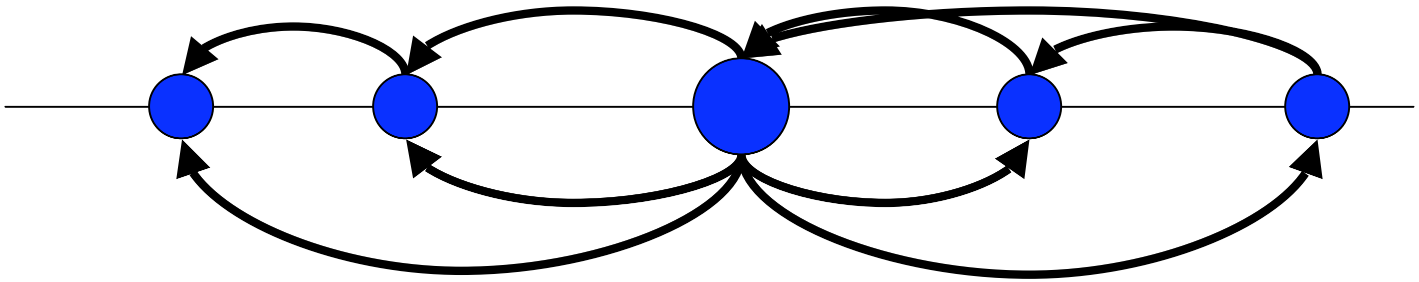
- Probe neighbors in overlay



- Exploit symmetric state
  - Heartbeats versus probes
- Number of heartbeats is number of neighbors
- Supress heartbeats with application traffic

Exploiting structure for maintenance

- Heartbeat sent to neighbor on the left
- Probe node if no heartbeat
- Tell others about failure if no probe reply



- **Leads to lower overhead**

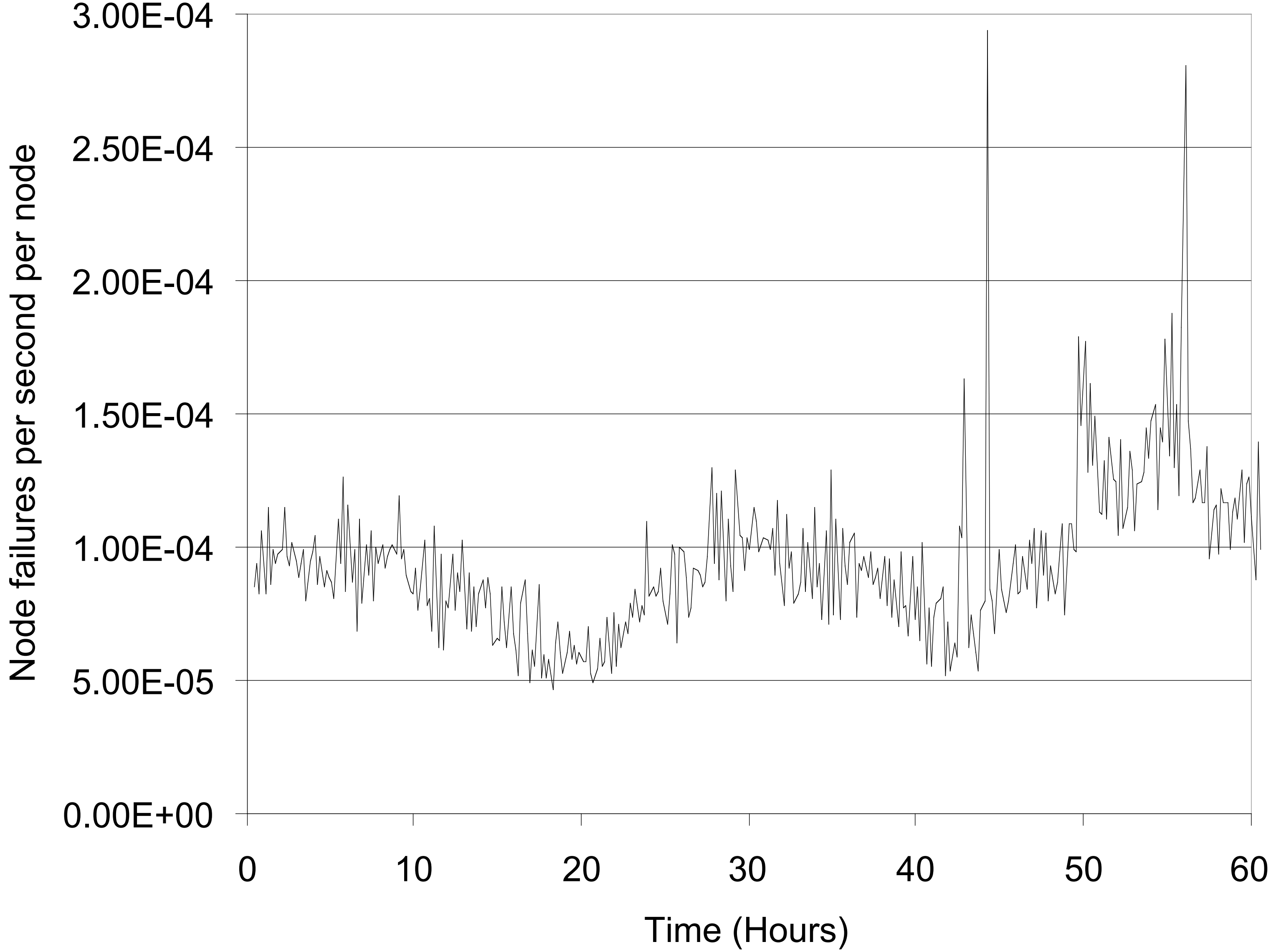
Comparing overhead

- Unstructured overlay (Gnutella 0.4)
  - Max and min bounds placed on # neighbors
  - Node discovery on join using random walks
  - Failure detection heartbeat every 30 seconds
- Structured overlay (MS Pastry)
  - Leafsets
    - Failure detection using heartbeats every 30 seconds
  - Routing table
    - Failure detection using probes (tuned to churn)

Experimental comparison

- Discrete event simulator
  - Transit-stub network topology
- UW trace of node arrivals and departures
  - [Saroiu et al. MMCN’02]
  - 60 hours trace
  - average session = 2.3 hours, median ~ 1 hour
  - Active nodes varies between 2,700 and 1,300

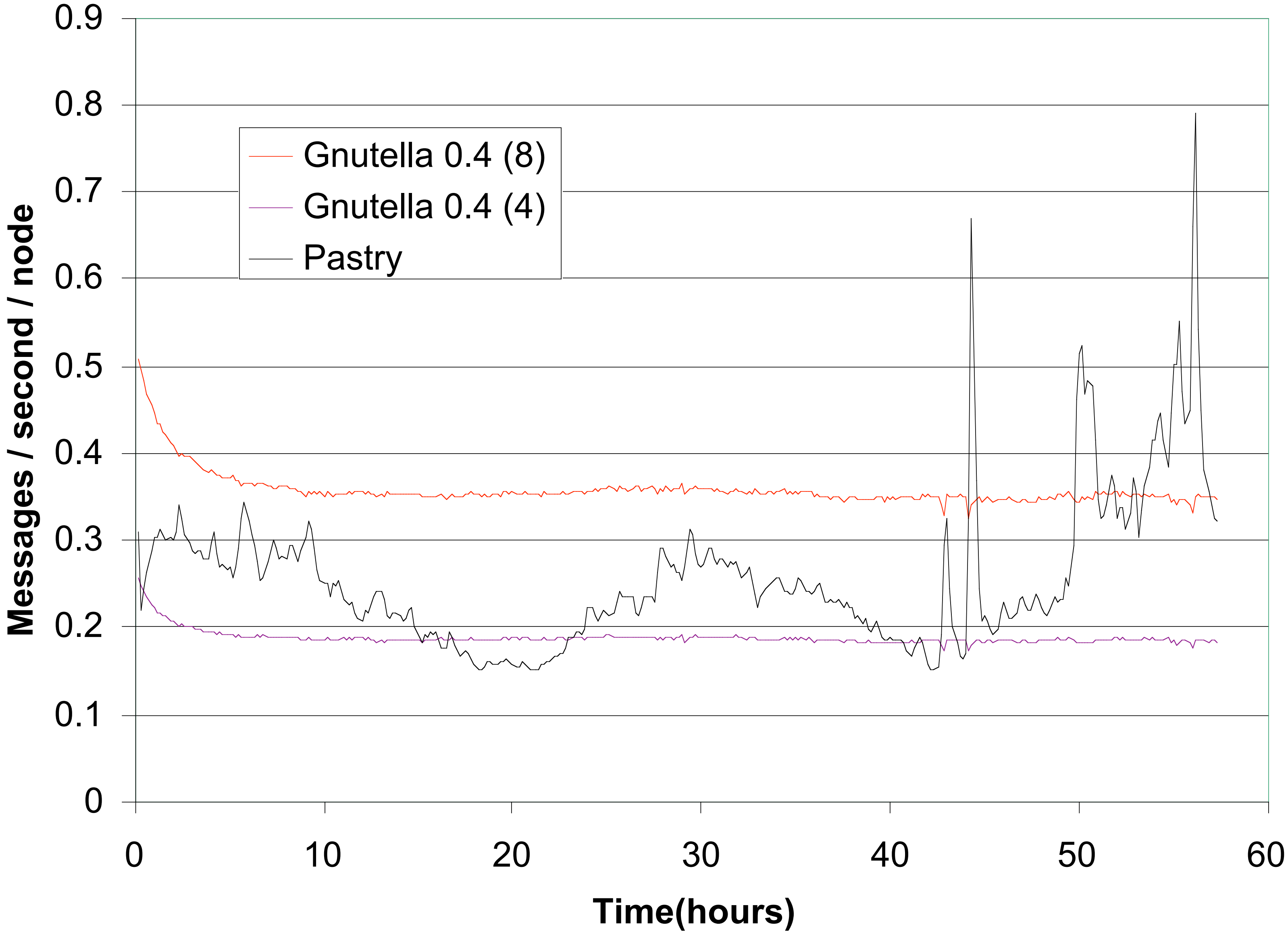
Gnutella trace: Failure rate



Overhead: Configuration

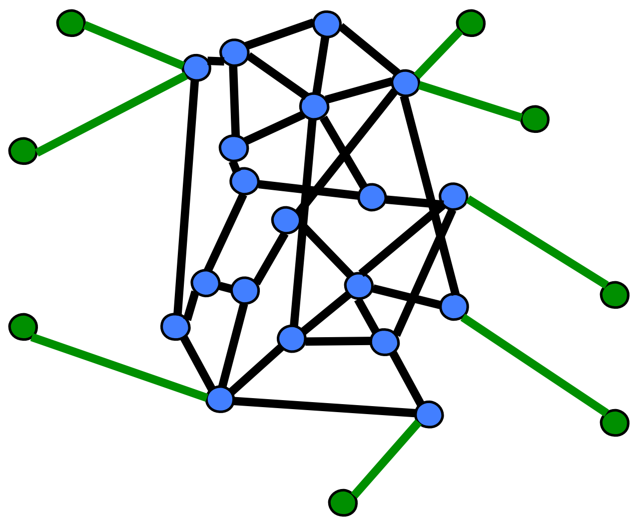
- Gnutella 0.4 (4)
  - Min neighbors 4, max neighbors 8 (avg. 5.8)
- Gnutella 0.4 (8)
  - Min neighbors 8, max neighbors 32 (avg. 11)
- Pastry
  - b=1, no proximity neighbor selection, l = 32

Overhead: Maintenance



Gnutella 0.6 (SuperPeers)

- Super peers form random graph
  - Uses Gnutella 0.4 algorithm



- Normal nodes use super peers as proxies
  - Failure detections using heartbeats (30 secs)
  - Connect to multiple super peers

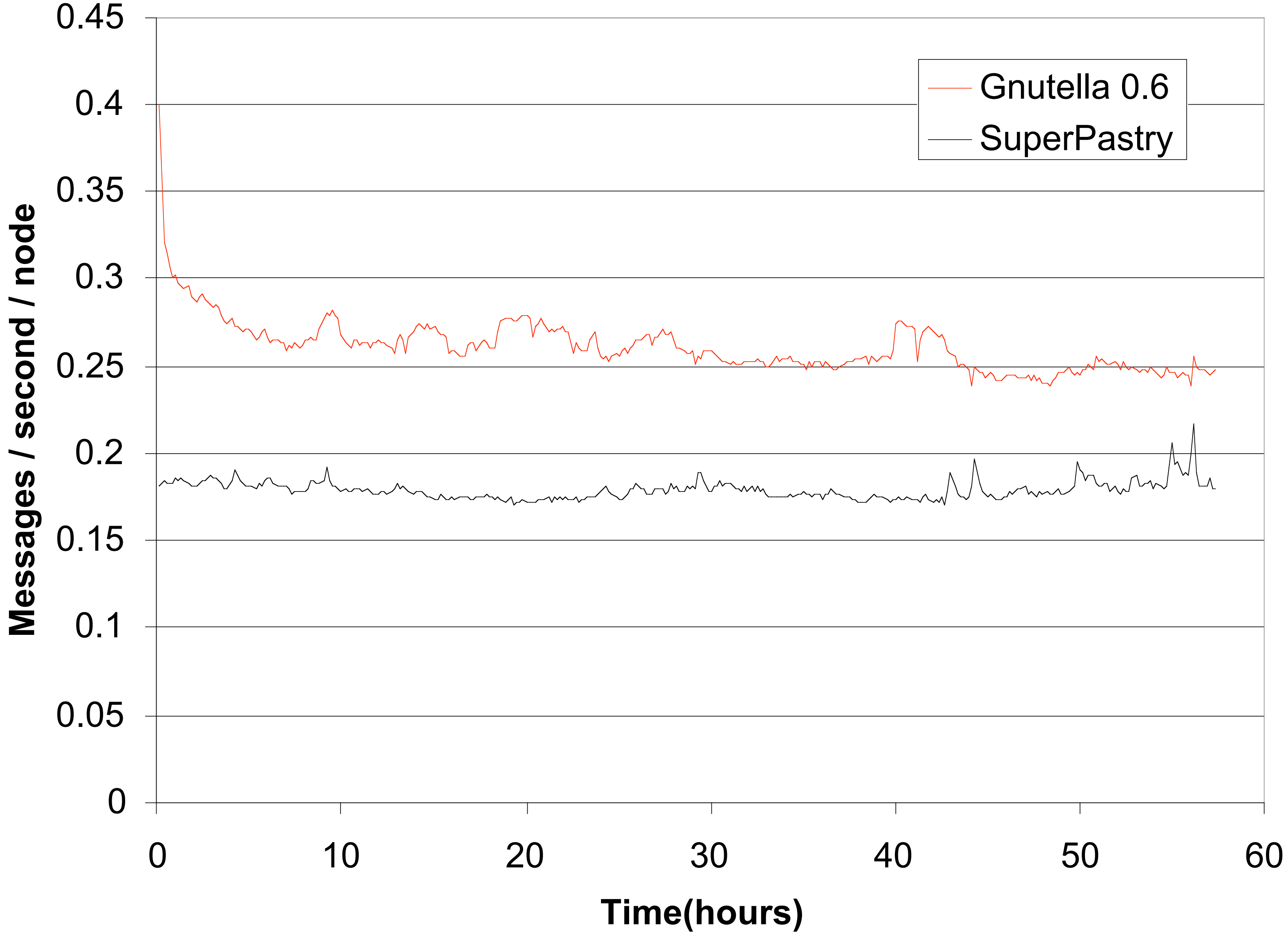
SuperPastry

- Super peers form Pastry overlay
- Normal nodes use super peers as proxies
  - Failure detections using heartbeats (30 secs)

Overhead: Configuration

- 0.2 probability of node being a super peer
- Gnutella 0.6 configured:
  - Min neighbours = 10
  - Max neighbours = 32
- SuperPastry configured
  - Max in-degree from routing table = 32
- Super peers proxy for 30 normal nodes
- Normal nodes pick 3 super peers

Overhead: Maintenance



Gia [Chawathe et al. SIGCOMM'03]

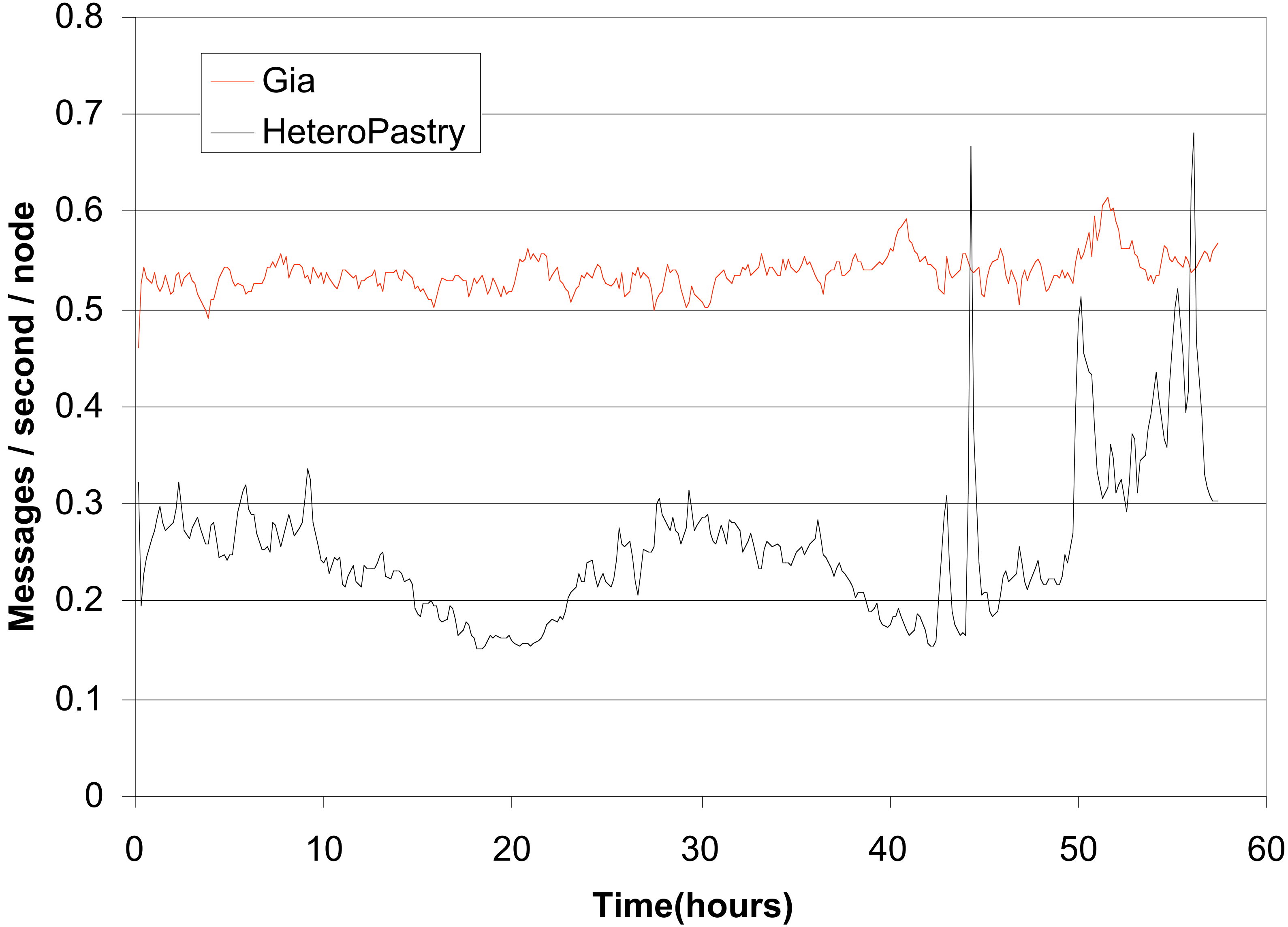
- Adapts overlay to exploit heterogeneity
  - Uses a per-node metric of satisfaction
  - Seeks new neighbors if unsatisfied
  - Use parameters in Sigcomm Paper
  - Neighbors [min = 3, max = max(3,min(128,C/4))]
    - Average 15.8

Capacity	Probability	Neighbors
1	0.20	3
10	0.45	3
100	0.30	25
1000	0.049	125
10000	0.001	128

HeteroPastry

- Routing table neighbor selection using capacity metric
- Uses routing table in-degree bound
  - Calculated as for Gia

Overhead: Maintenance



The story so far….

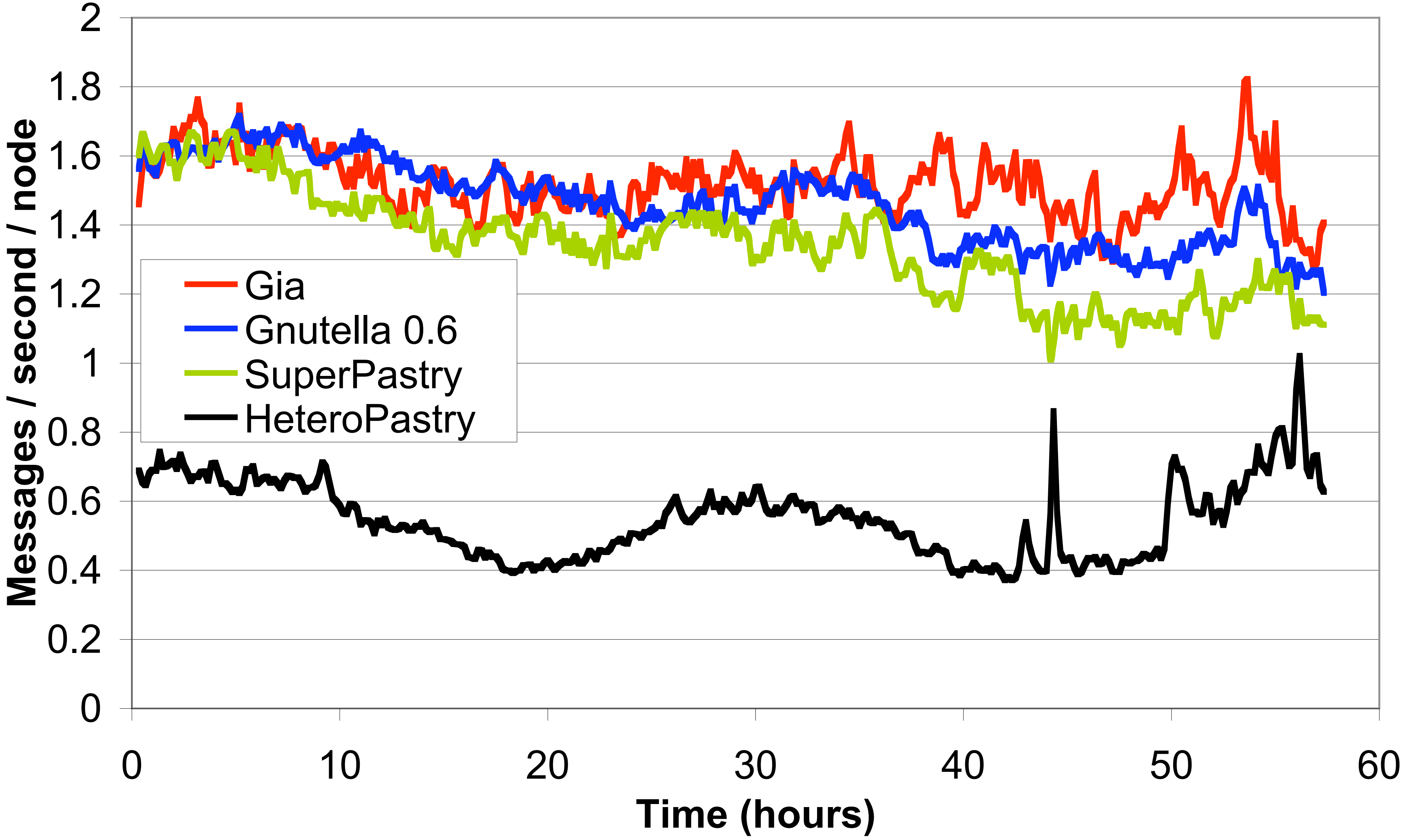
- Both structured and unstructured
  - detect failures
  - repair overlay graph when nodes join or leave
- Structured exploits structure
  - Lower overheads
- Unstructured overlays sensitive to neighbors choice
  - Random walks between node discovery

Finally: Putting it all together….

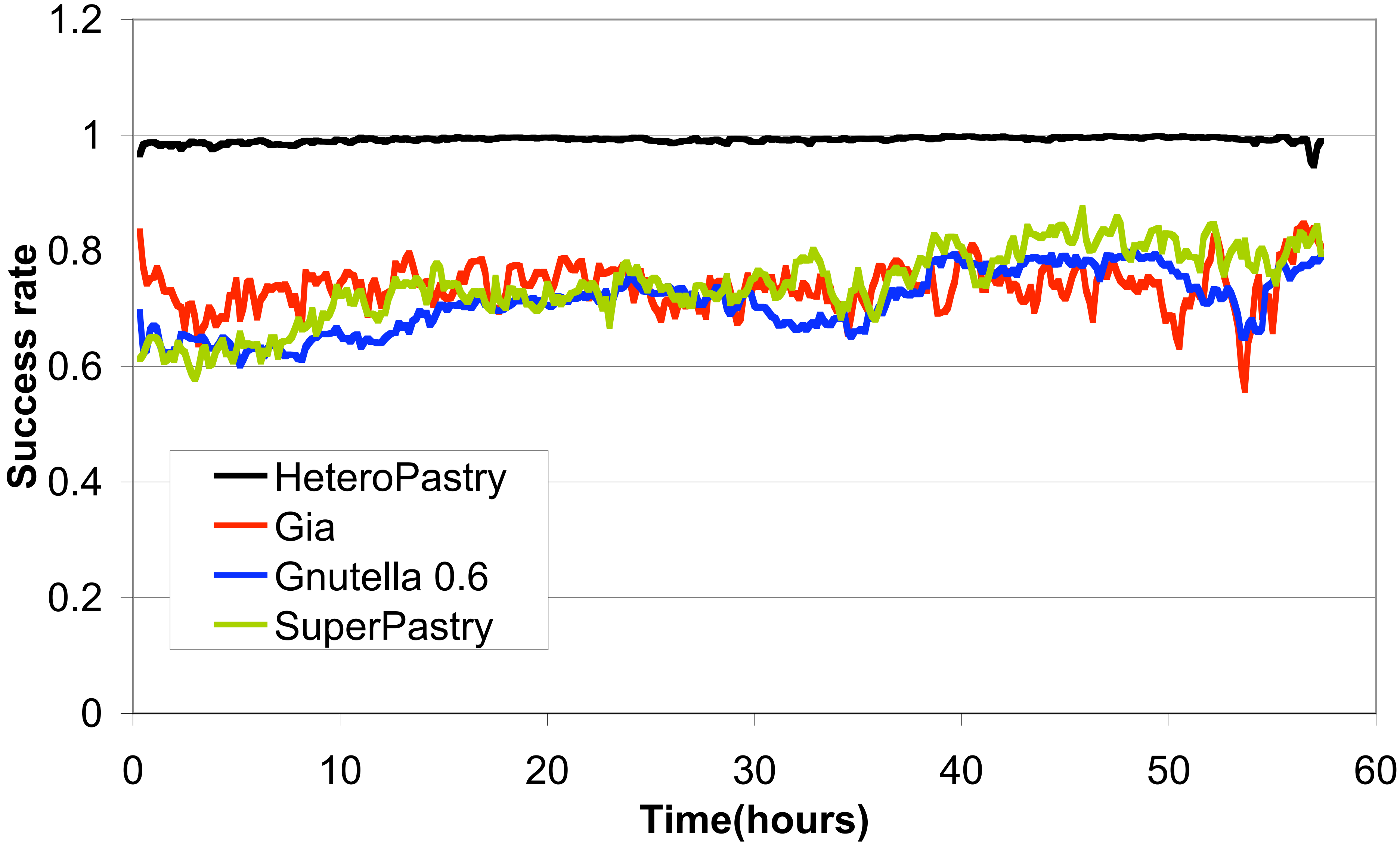
Search: Configuration

- eDonkey file trace [Fessant et al. IPTPS'04]
  - 37,000 peers (25,172 contribute no files)
  - 923,000 unique files (heavy tail zipf-like)
- Each node performs 0.01 lookups per second (using a Poisson process)
  - Random walks TTL 128
- One hop replication [Chawathe et al. SIGCOMM'03]
  - Uses routing table in structured overlays (***)

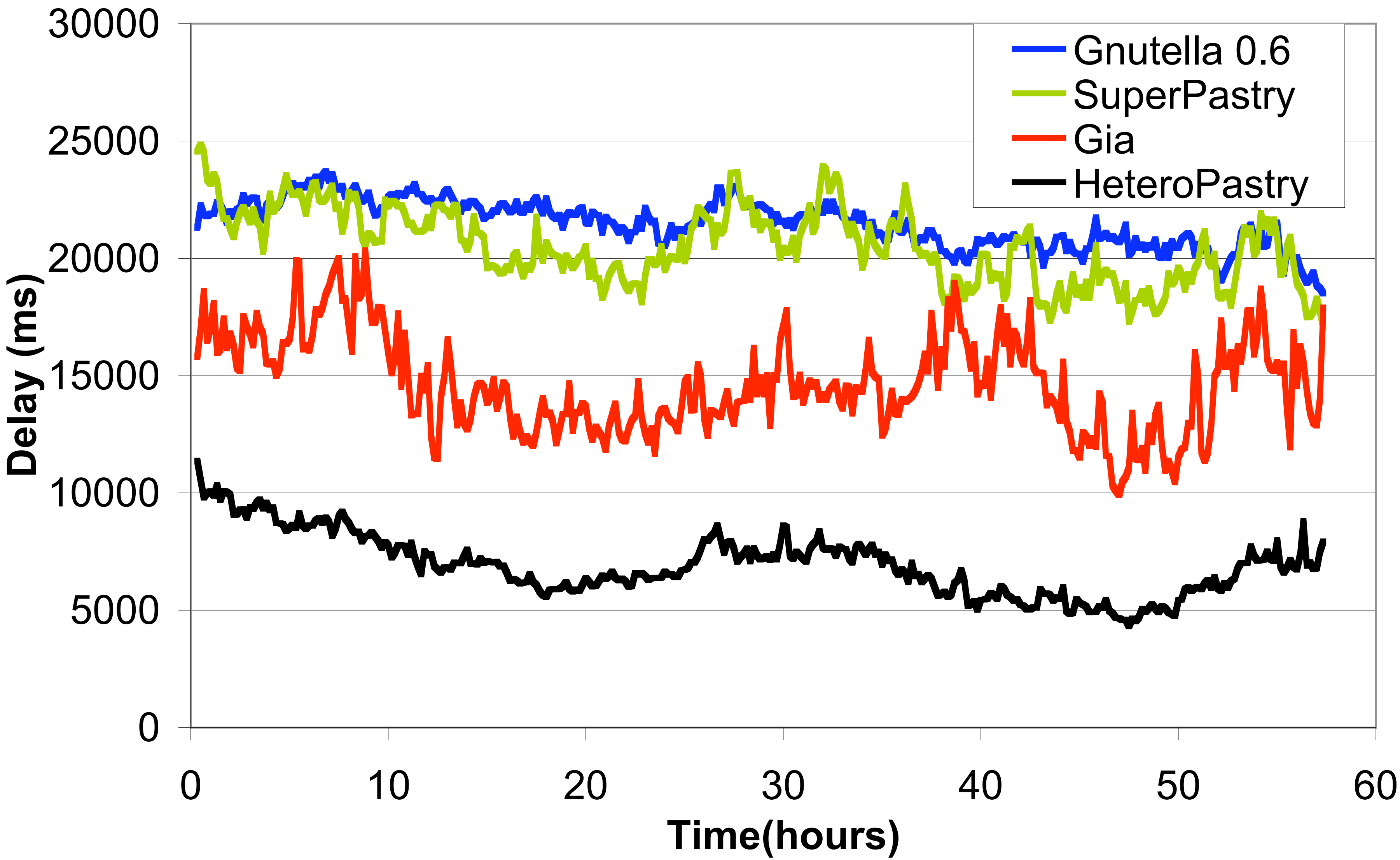
Search: Messages



Search: Success rate



Search: Delay



Conclusions

- Structure can improve Gnutella
  - Handles transient peers well
  - Exploits structure to reduce maintenance overhead
  - Supports complex queries
  - Can also support DHT functionality
  - Can exploit heterogenity

And finally a question…

Does structure make security easier?

For slides:

http://www.research.microsoft.com/~antr/camb-ast.ppt

For more information:

http://www.research.microsoft.com/~antr/Pastry

Flooding queries

- **exploit structure to avoid duplicates**
- flooding a query *q*
  - if node is source of *q* do

    for each routing table row ***r***

    send <*flood, q,* ***r***> to nodes in row ***r***
  - if node receives <*flood, q, s*> *do*

    for each routing table row ***r*** **such that *r* > *s***

    send <*flood, q,* ***r***> to nodes in row ***r***
- recursively partitions nodes into disjoint sets