Fundamentals – Part One

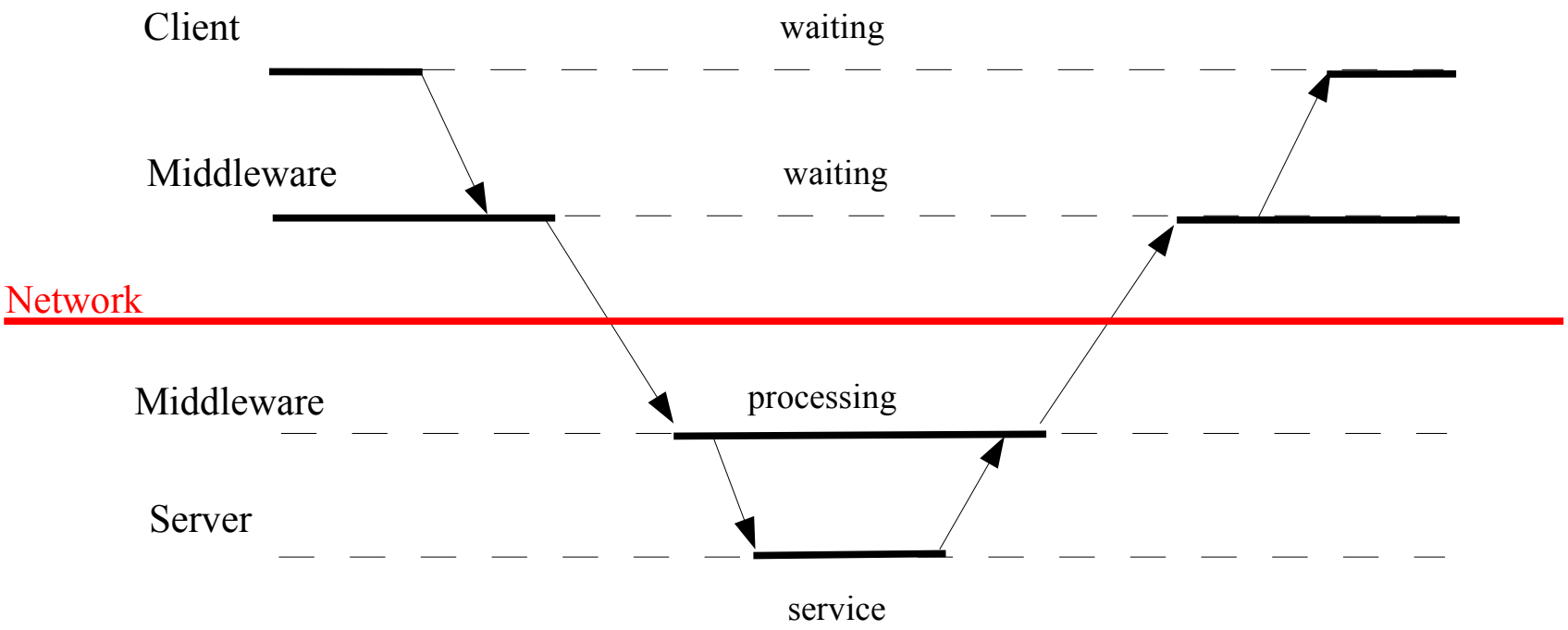
Professor Olivier Gruber

Université Joseph Fourier

Projet SARDES (INRIA et IMAG-LSR)

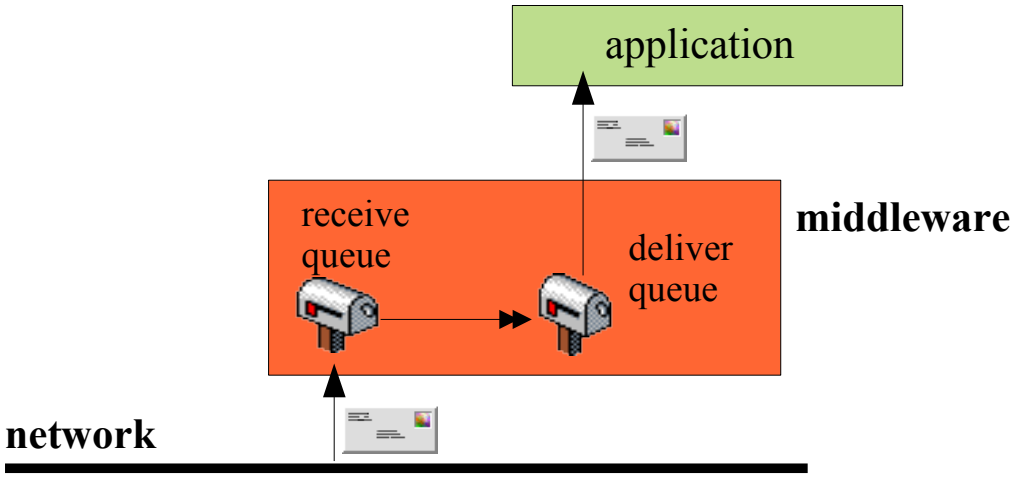
Message Basics

- Communication Architecture
  - How do we name the destination of a message?
  - How do we route the message to its destination?



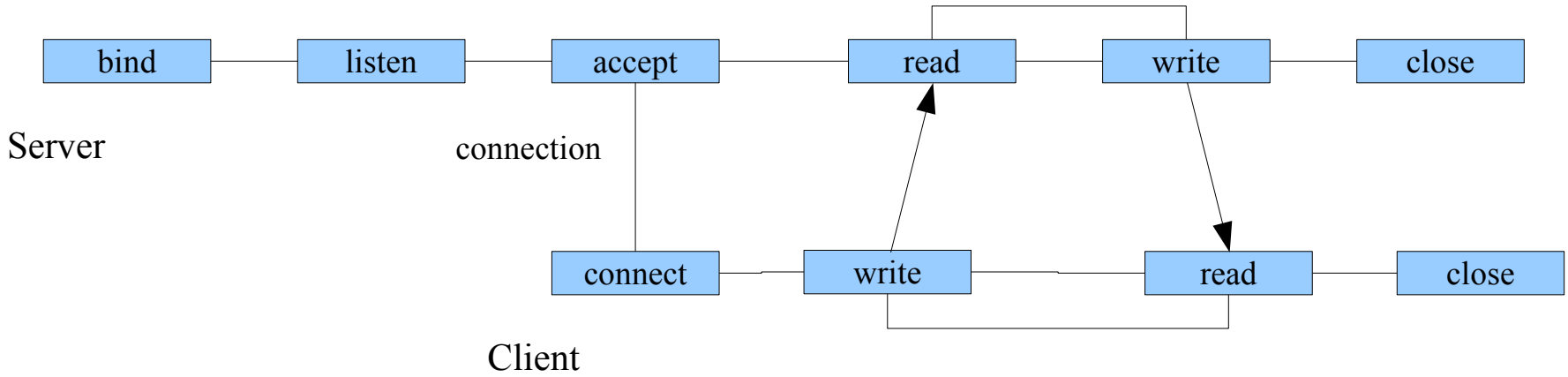
Message Basics

- **Receive** versus **Deliver**
  - The middleware receives a message
  - The middleware delivers a message to the application (process)
- **Interface** versus **Semantics**
  - The interface is about what you can do
  - The semantics is about what it means



Message Basics

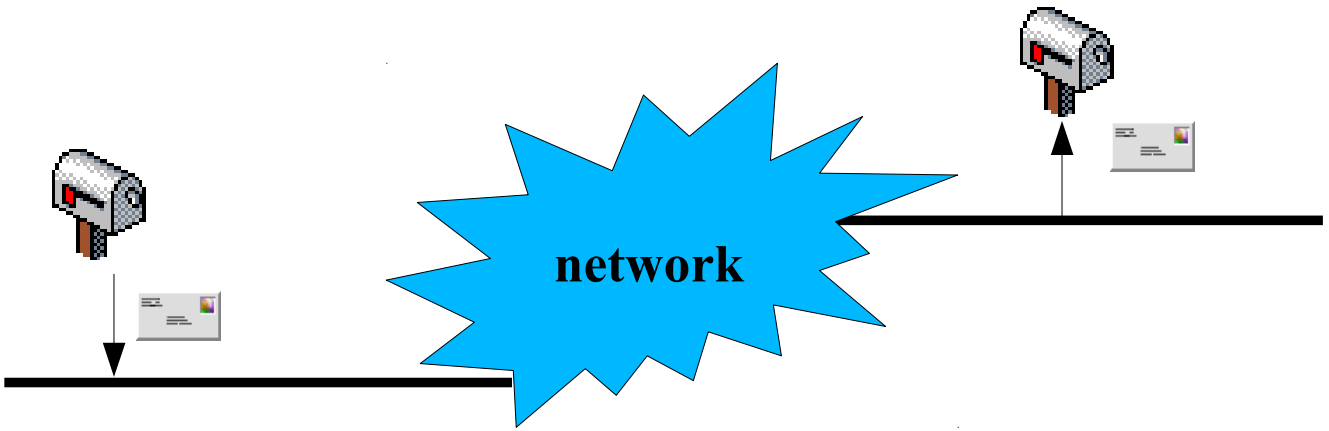
- Socket Example
  - An application programming interface (API)
    - Asymmetric connection
      - A server listens on a port
      - A client connects to a server (IP address, port number)
    - Symmetric data exchange
      - A stream API for both sending and receiving data



Message Basics

- Protocol properties
  - UDP:
    - **Delivered when received**, messages may be lost and received out of order
    - Operating systems and *network routers* store and forward network packets
    - Network packets are discarded on failures
      - Transmission fails anywhere
        - A router or the destination host is down for example
        - Or a checksum error happens
      - No one listen on the destination port number
  - TCP:
    - Same interface, but very different semantics
      - **FIFO: delivered in order**
      - **Lossless: nothing is lost**
    - More complex and expensive implementation
      - Based on ACK and retransmission of lost data

Discussing Naming



How do we name a destination?

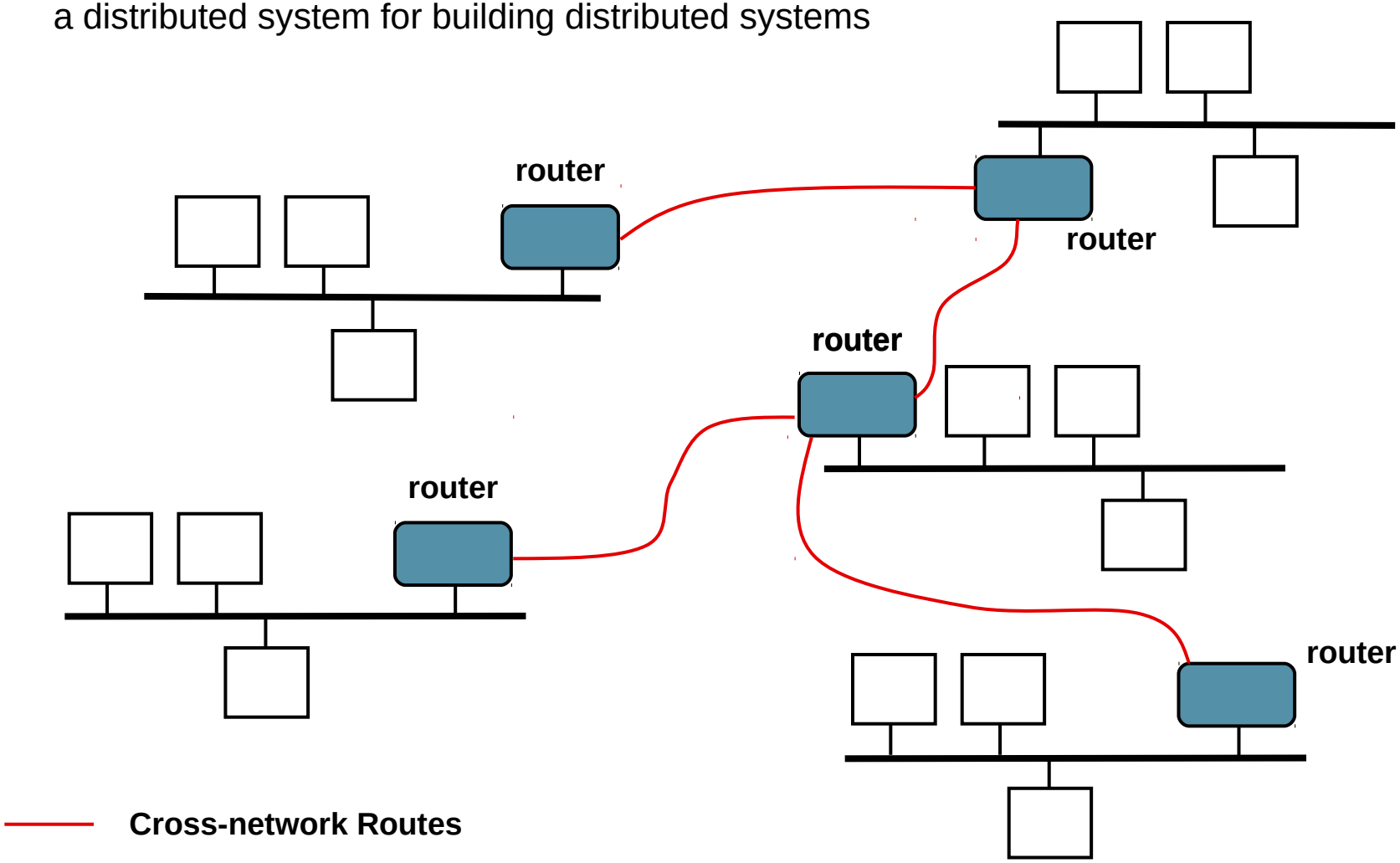
How do we route a message?

Discussing Naming

- IP Naming
  - IP addresses are names
    - String of bits naming a host machine (192.168.2.100)
  - IP addresses are not identities
    - A machine may change IP addresses, it may have multiple IP address
    - IP addresses may be reused (DCHP on a local network for example)
- IP Routing
  - Addresses are special names providing physical access to an entity
  - Access protocol using the address is the IP routing protocol
    - On LANs:
      - Physical layer directly provides this
    - On WANs:
      - It is a collaborative and distributed protocol
      - Routers do exchange their routing tables to build up their routing knowledge

IP Network

Interconnecting Local Area Networks...
a distributed system for building distributed systems



Discussing Names

- Human-friendly Names
  - Addresses are good for machines but difficult for humans
  - Layering name services... as we layer distributed systems
    - Names over IP addresses (also names)
    - DNS (Domain Name Service) over IP network
- Domain Naming Service
  - Manage the mapping from names to addresses
  - May be used for identity (unchanging name, changing IP)

How do we resolve a name?

How do we scale to hundreds of millions of names?

How do we scale to millions of requests per day?

How do we scale to world-wide resolution?

How do we resist failures?

Internet Summary

- LANs support physical access
  - Minimal access protocol, supported by hardware
- Distributed routing
  - Routing across LANs
  - Requires to exchange routing tables
- Domain Name System
  - Built on IP addresses
  - Needs IP routes to DNS servers
  - Map hierarchical domain names to IP addresses

Introducing Identity

- Identity
  - Refers to one and only one entity
  - Each entity has only one identity
  - Provides unambiguous addressing
  - Easier aliasing through logical names maping to the identity
- Routing Challenge
  - Internet names are location-dependent
    - IP addresses or hierarchical names
    - Even more true for URLs (include a web server address and a resource path)
    - Helps routing (because names embed location information)
  - Using identity, routing becomes a challenge
    - Flat identifier space, no information about location

Routing with Identity

- A Simple Solution
  - Using multicasting or broadcasting on a LAN
  - Does not scale well on wide-area networks
- Peer-to-Peer Overlays
  - **Structured overlays**
    - We will look at Distributed Hash Tables
    - Case study: Chord System
  - **Unstructured overlays**
    - We will look at random graphs
    - Case study: CYCLON

Distributed Hash Tables

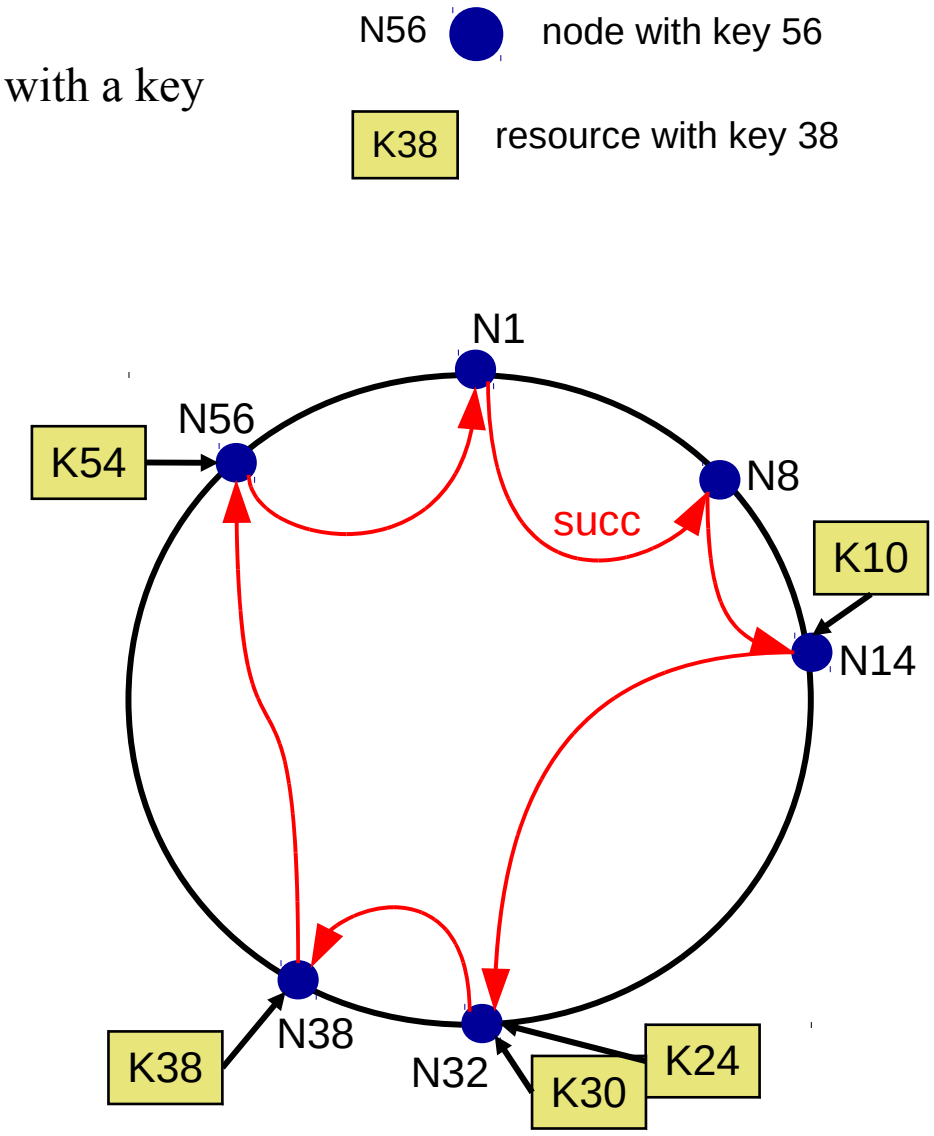
- Adopting Identity
  - Entities are identified by m-bit keys
    - The key space is usually 128 or 160 bits
  - Entities may be anything
    - Host, processes, files, etc.
- Distributed Nodes
  - Each node is responsible for managing certain keys
    - A node store the resources for the keys it manages
  - Each node is identified with a key
    - From the same m-bit key space as resources
- Example: DNS on DHT
  - Instead of using a hierarchy of servers for storing DNS records
  - Use a distributed set of nodes and a DHT
  - Compute key from the name, the resource is the DNS record

Distributed Hash Table

- Dynamic Set of Nodes
  - Nodes may join or leave the DHT
    - No global knowledge, synchronization or management
    - **No single point of failure**
  - Fully scalable
    - **Uniform distribution of resources across nodes**
- Case Study: Chord System
  - I. Stoica et al (2001)
    - *Chord, A Scalable Peer-to-Peer Lookup Service for Internet Applications*
    - IEEE-ACM Trans on Networking
    - http://pdos.csail.mit.edu/chord/papers/paper-ton.pdf

Chord - Basics

- Distributing Resources
  - A resource with a key K_i is managed by a node with a key N_k such as
    - N_k is the smallest node key such as $\boldsymbol{K_i \leq N_k}$
    - Such a node is called the ***succ(K_i)***
- Circle Representation
  - Organizing keys on a circle
    - From 0 to 2^m-1
    - Clock-wise
  - The succ Relationship
    - For a key K_i
    - It is the next available node
    - Clock-wise from key K_i

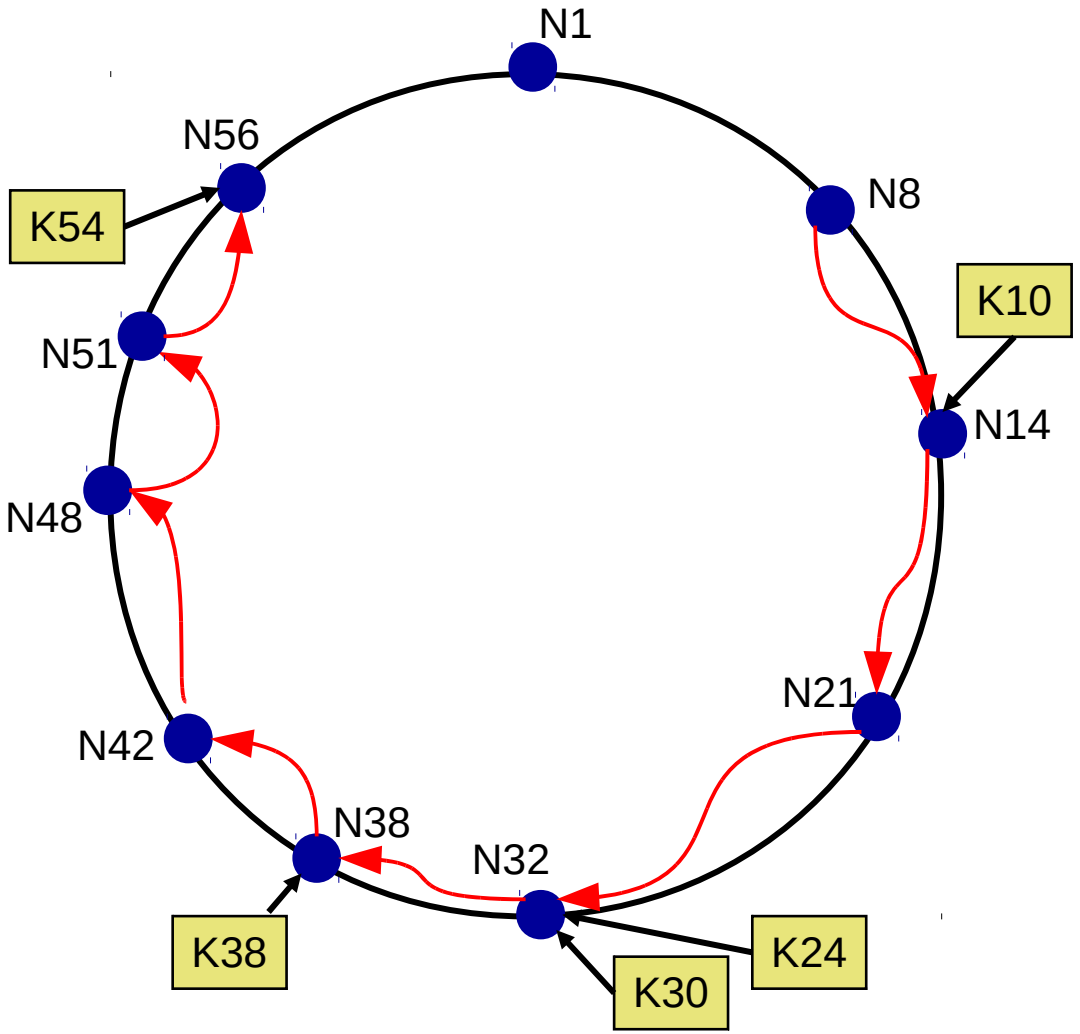


Chord – Simple Lookup

m = 6, 2^m = 64

Only 10 nodes and 5 keys in the hash table

Example: starts in node N8, looking up key K54.



N56 node with key 56

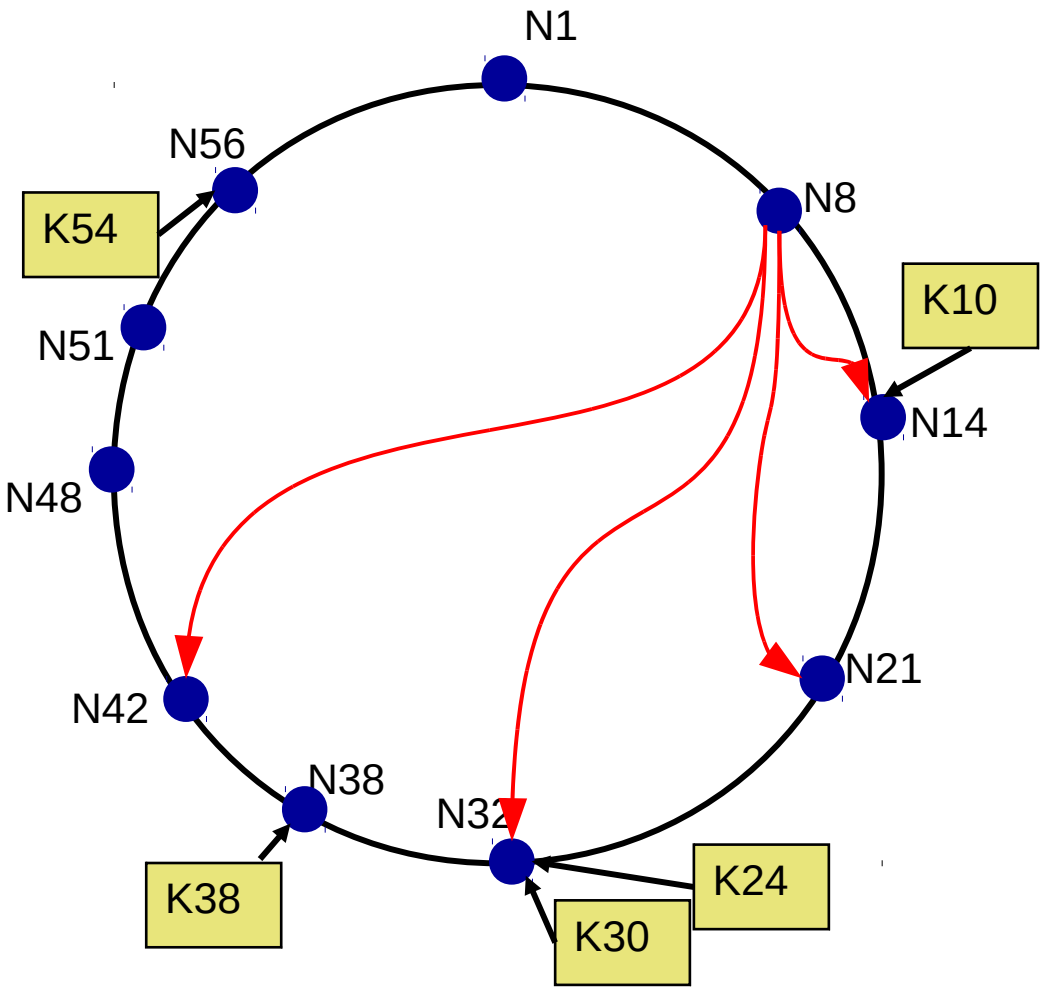
K38 resource with key 38

Chord – Finger Table Principles

- Basic Idea
  - When looking up a key at a node
    - Looks for the successor of that key
    - It is the node managing that key
  - If the node does not know the successor of key
    - It may know of one node that is closer on the ring
    - That node should know more about the successor of the key
- Finger Tables
  - One index of nodes per node
    - Of at most m entries (for m-bit key space)
  - For a node $\boldsymbol{N}_i$, the finger entries are computed as follows:

$$\text{finger[k]} = succ(\boldsymbol{N}_i + 2^{k-1}) \ \textbf{mod} \ 2^m$$

Chord – Introducing Finger Tables



DHT: m = 6, $2^m = 64$

Finger table for a node N_i

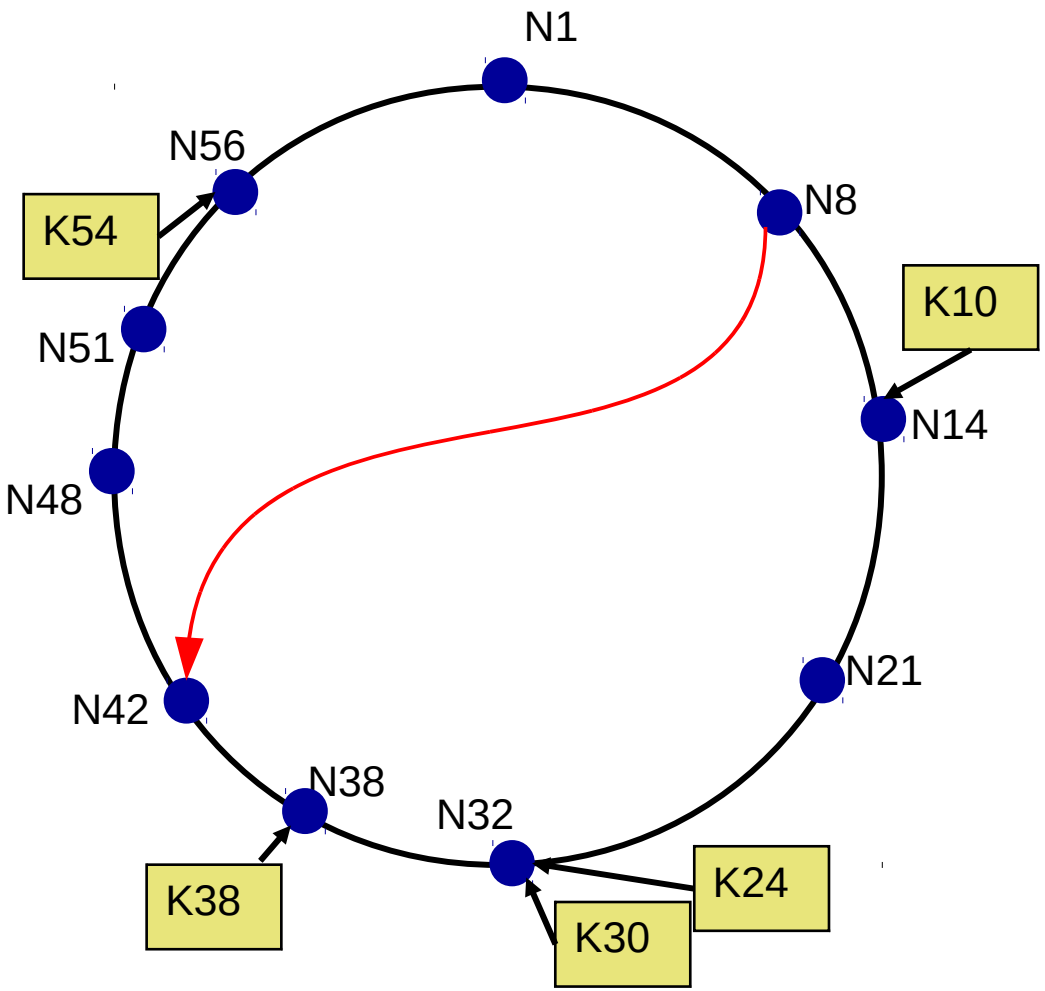
finger[k] = $succ(N_i + 2^{k-1})$ **mod** 2^m

with $1 \leq k \leq m$

1	N8+1	N14
2	N8+2	N14
3	N8+4	N14
4	N8+8	N21
5	N8+16	N32
6	N8+32	N42

Finger Table for N8

Chord – Lookup with Finger Tables



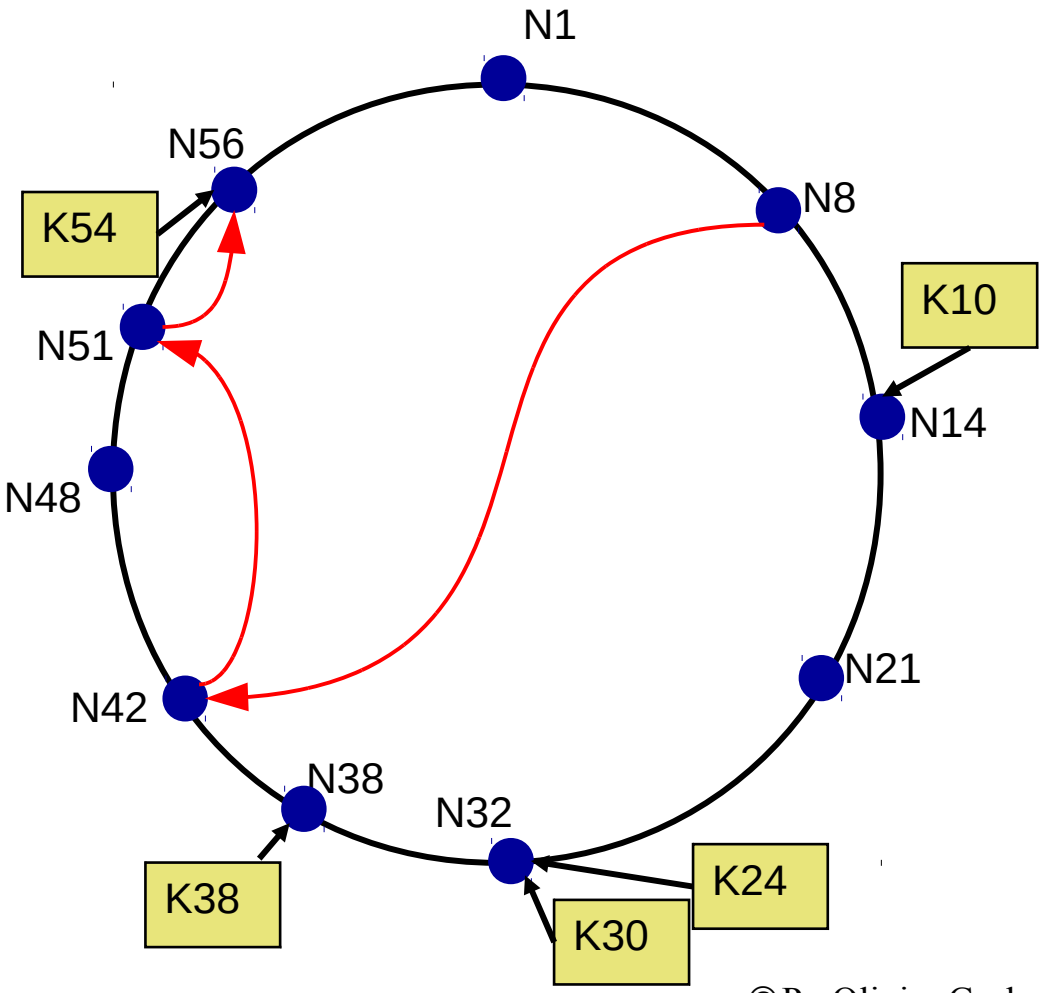
DHT: m = 6, 2^m = 64

Looking up key 54, from N8

1	9	N14
2	10	N14
3	12	N14
4	16	N21
5	24	N32
key 54 → 6	40	N42

Finger Table for N8

Chord – Lookup with Finger Tables



(1) does not know the successor

(2) searches for the node that immediately precedes the looked up key

key 54

1	43	N48
2	44	N48
3	46	N48
4	50	N51
5	58	N1
6	10	N14

Finger Table for N42

(1) knows the successor

key 54

1	52	N56
2	53	N56
3	55	N56
4	59	N1
5	3	N8
6	19	N21

Finger Table for N51

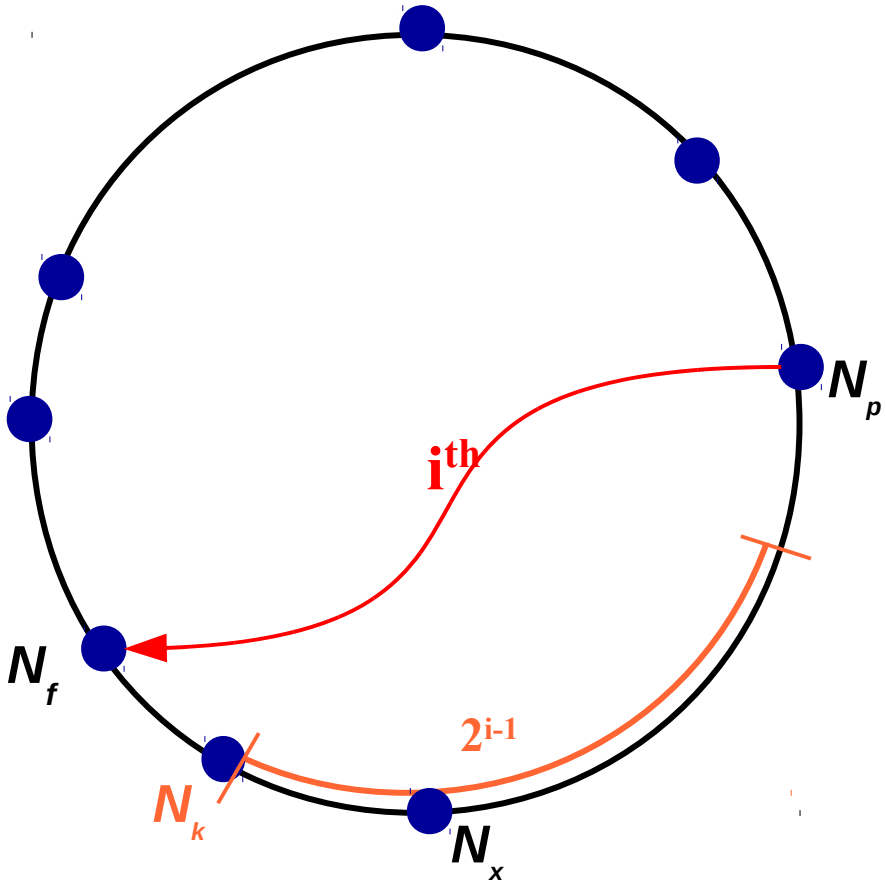
Chord – Joining or Leaving

- Minimal Invariants
  - Each node's successor is correctly maintained
  - For every key K_i, succ(K_i) manages that key
  - For simplicity, all nodes also maintain their predecessors
- Joining the Ring
  - For a node with a key N_k
    - Find through any node in the ring the succ(N_k)
      - Insert itself before that node in the ring
      - Builds finger table, asking for succ(N_k+2^{i-1}) with i ∈ [1,m]
    - Update other finger tables
      - Potentially using background messages
    - **Transfer keys last**
      - Avoids not finding keys as long as finger tables are not correct

Chord – Updating Finger Tables

Updating the i^{th} finger entry for a Node N_p

- If N_p precedes N_k by at least 2^{i-1}
- If the current i^{th} finger, N_f, succeeds N_k



finger[k] = $succ(N_i + 2^{k-1})$ **mod** 2^m

update_others(N_k)
 for (i=1 to m)
  N_p = predecessor(2^{i-1})
  update_table(N_p,N_k,i)

update_table(N_p, N_k, i)
 $N_f = N_p.finger[i]$
 if $N_k \in [N_p, N_f[$
  $N_p.finger[i] = N_k$
  N_p = predecessor(N_p)
  update_table(N_p,N_k,*i*)

Chord Summary

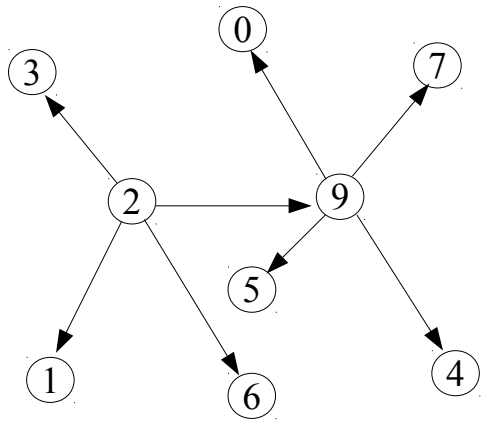
- Distringuishing features
  - Simplicity and provable correctness and performance
- Lookup Performance
  - With high probability, we have O(log N) messages to lookup a key
    - The average is therefore 0.5 log(N) messages (normally distributed keys)
  - Finger Tables
    - In a m-bit space of keys, traditional size is m entries
    - Finger table size could be reduced to O(log N) instead of m
- Dynamic Behavior
  - Joining and leaving the overlay ring
    - First challenge is maintaining the minimum invariants
    - Second challenge is maintaining finger tables
      - Need no more than $O(\log^2 N)$ with high probability
  - Harder in the presence of faults

Unstructured Overlays

- Middleware Platform
  - For highly dynamic environments
  - Networks with potentially major failures
- Approach
  - Based on random graph theory
    - Each node maintains a list of neighbors

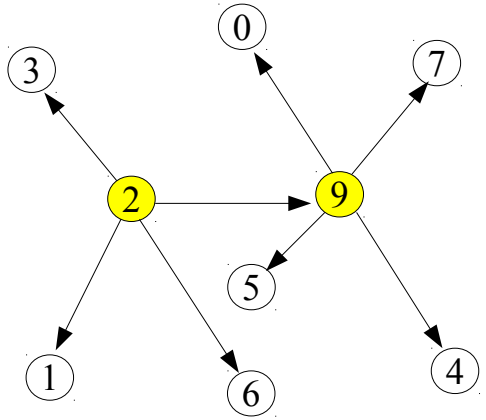
      (2) knows { (3) (1) (6) (9) }

    - Neighbors are randomly chosen
    - Neighbor lists are exchanged
  - Epidemic broadcast
    - To find something, broadcast on the overlay
    - With high-probability, it will be found quickly (just a few network hops)



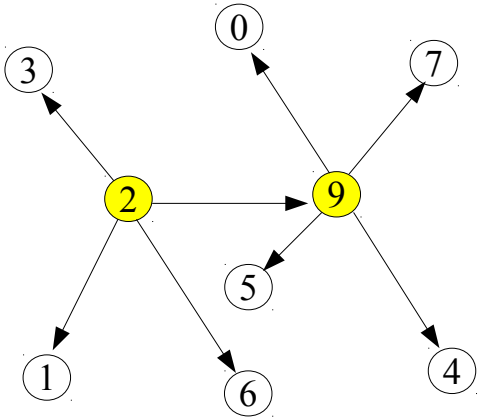
Unstructured Overlays

- Case Study – Basic Shuffling
  - Overlay network
    - Edge cache of *C* entries
    - Shuffle Length (*SL*) is smaller than *C*
  - Periodic ***shuffle*** **algorithm**...

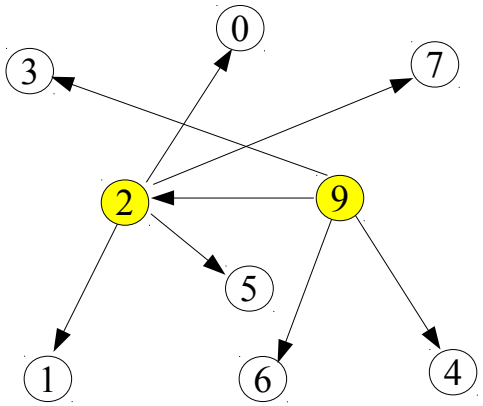


Shuffle Algorithm

- Randomly select *SL* edges from N_p cache
  - Select a random peer N_q from this selection
  - Replace N_q with N_p in this set
- Exchange neighbors
  - N_p sends this set to N_q
  - N_q updates its cache with received edges
    - Using empty slots first
    - Re-using non-empty slots second
    - N_q sends back replaced edges to N_p
  - N_p updates its cache
    - Discard entries to N_p and those already known
    - Saves new edges using empty slots first
    - Then reuse slots for edges sent to N_q



2 → 9 : {**2**,3,6}
2 ← **9** : {0,5,7}



Unstructured Overlays

- What about Connectivity?
  - Without failures, connectivity is always preserved
    - No edges are lost, just exchanged
      - Intuitively, this preserves connectivity
    - Two sets of nodes cannot become disconnected
      - Assume that we are down to one link between two sets of nodes (S_1 and S_2)
      - Shuffling within S_i cannot lose this one link, just move it around
      - Shuffling between S_1 and S_2, just merely reverses the edge
  - With failures, connectivity may be lost
    - But this is true with all approaches in the presence of failures
    - For example, a router failure may disconnect two networks

Unstructured Overlays

- Joining the Overlay
  - A node needs just one node in the overlay
    - Joining is just building a list of neighbors
    - The new node needs to know some neighbor nodes
    - Some other nodes in the overlay need to know the new node as neighbor
  - Simple find and exchange approach
    - Using the known node
    - Achieve random walks to N distinct nodes
    - For each of them, exchange one of their neighbors with the new node
    - Set the new node neighbor list to that set of randomly chosen nodes
- Leaving the Overlay
  - Nothing to do, just leave
    - Provides high failure resistance
    - When failing, a failed node cannot be ask to inform the overlay!
  - Non-responding neighbors are just forgotten by the overlay

Unstructured Overlays

- Broadcast Routing
  - Routing is done through broadcasting on the overlay
  - Is it efficient?
    - One may be afraid of very long paths
- Kevin Bacon Truth
  - Kevin Bacon: a somewhat known movie actor
  - *Anyone in the world would have a link to him in at most six hops!*
- Unstructured Overlays do Better
  - Stable overlay
    - Average distance around 3 and 4 hops
  - Convergence in the presence of updates
    - Converges on WANs between 7 to 14 minutes
    - For overlays of 100,000 nodes

CYCLON

- Enhanced Shuffling
  - Introducing the age of edges in the overlay network
  - Enhanced Algorithm (done at N_p)
    - Increase ages by one of all neighbors when shuffling
    - Create a set of *SL* edges from N_p cache
      - Select the **oldest edge** (refers to N_q) from N_p cache
      - Random select *SL-1* neighbors from N_p cache
      - Replace N_q edge with N_p edge (with age zero) in this edge set
  - Exchange neighbors
    - Same as before
    - N_q **does not** adjust ages within its cache

CYCLON – Connectivity Study

Experiment: 100,000 nodes, cache sizes=20, 50, 100

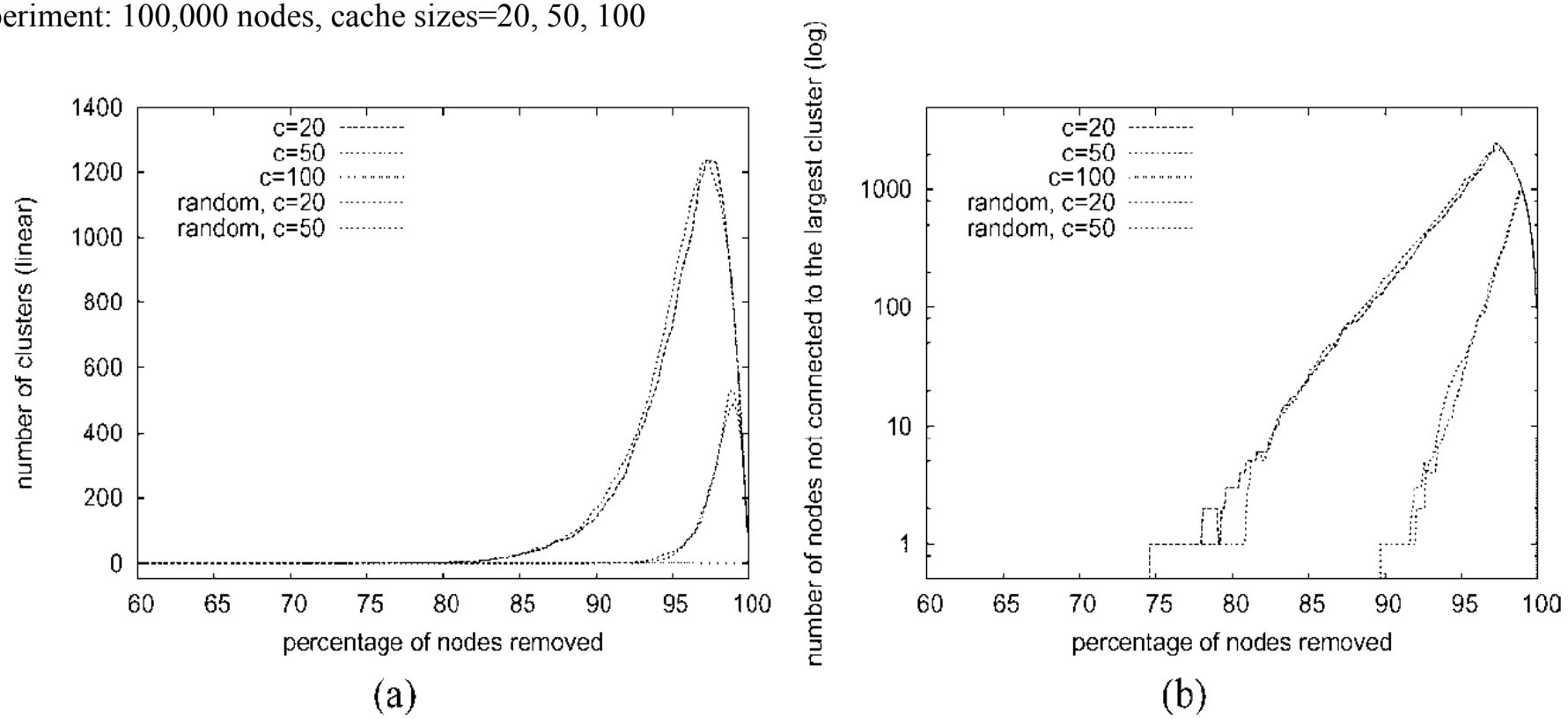
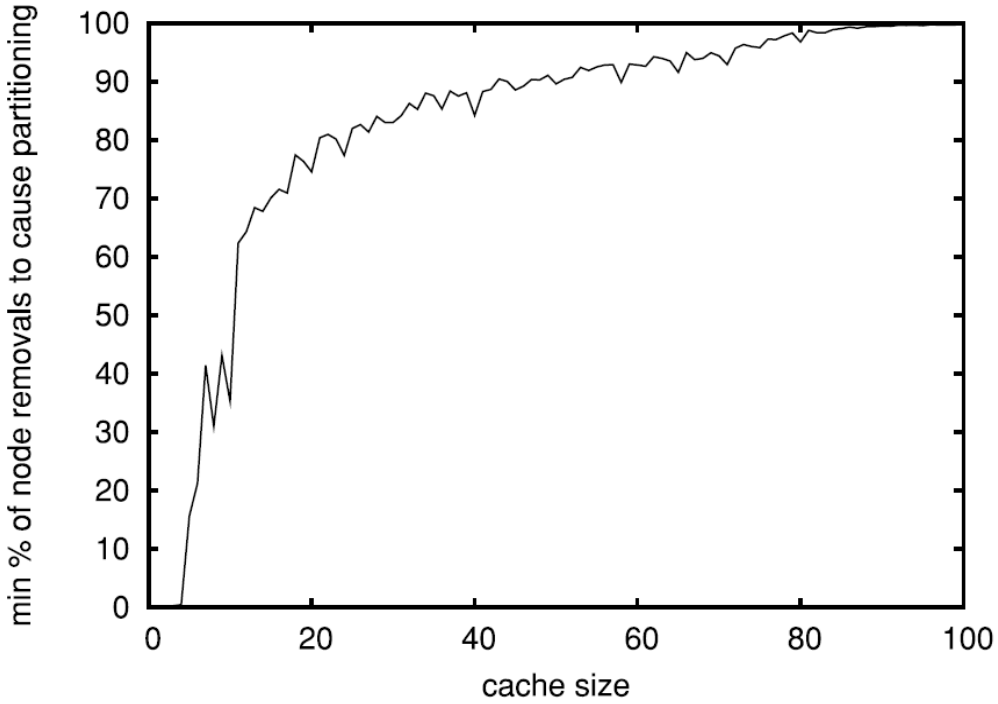


Fig. 7. (a) Number of disjoint clusters, as a result of removing a large percentage of nodes. Shows that the overlay does not break into two or more disjoint clusters, unless a major percentage of the nodes are removed. (b) Number of nodes not belonging to the largest cluster. Shows that in the first steps of clustering only a few nodes are separated from the main cluster, which still connects the grand majority of the nodes.

Source: Voulgaris et al
CYCLON, Inexpensive Membership Management for unstructured P2P overlays, 2005

CYCLON – Connectivity Tolerance

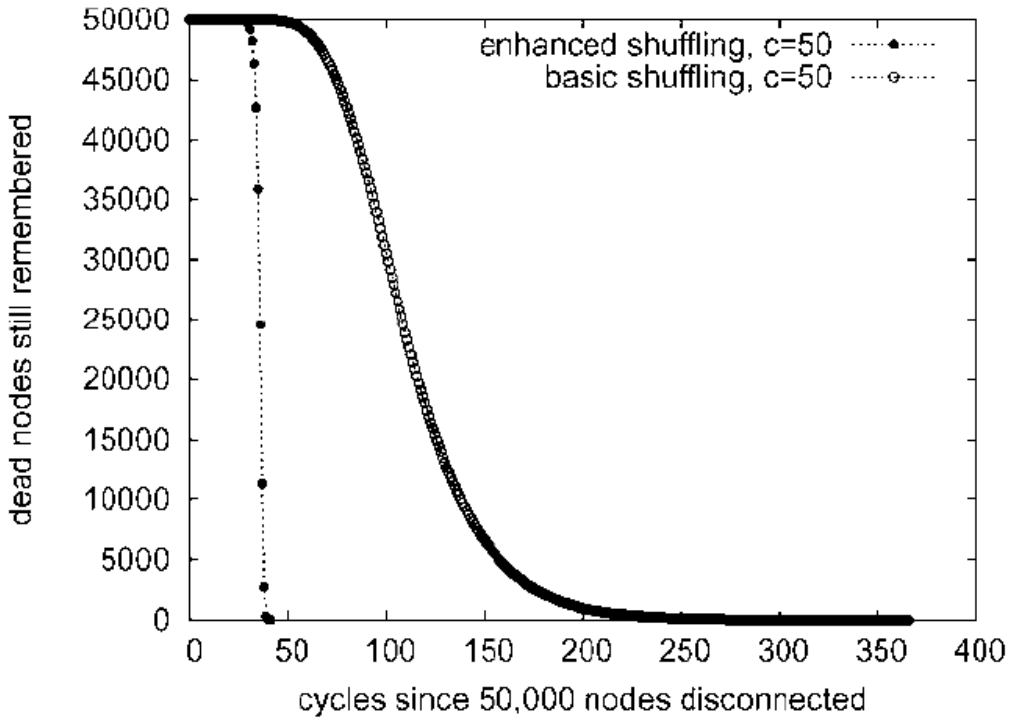
- Tolerance to Node Removals
  - 100,000 nodes
  - Search minimum number of removals to cause partitioning
- Discussion
  - Above cache size 100
    - Overlay is totally robust
  - Above cache size 20
    - Above 80% of removals



Source: Voulgaris et al
CYCLON, Inexpensive Membership Management for unstructured P2P overlays, 2005

CYCLON – Dynamic Behavior

- Dangling Links
  - Because node may fail or leave
    - No special message when a node leaves
  - Optimized dangling link removal (age of edges)
- Experiment
  - 100,000 nodes
  - 50,000 nodes removed at once



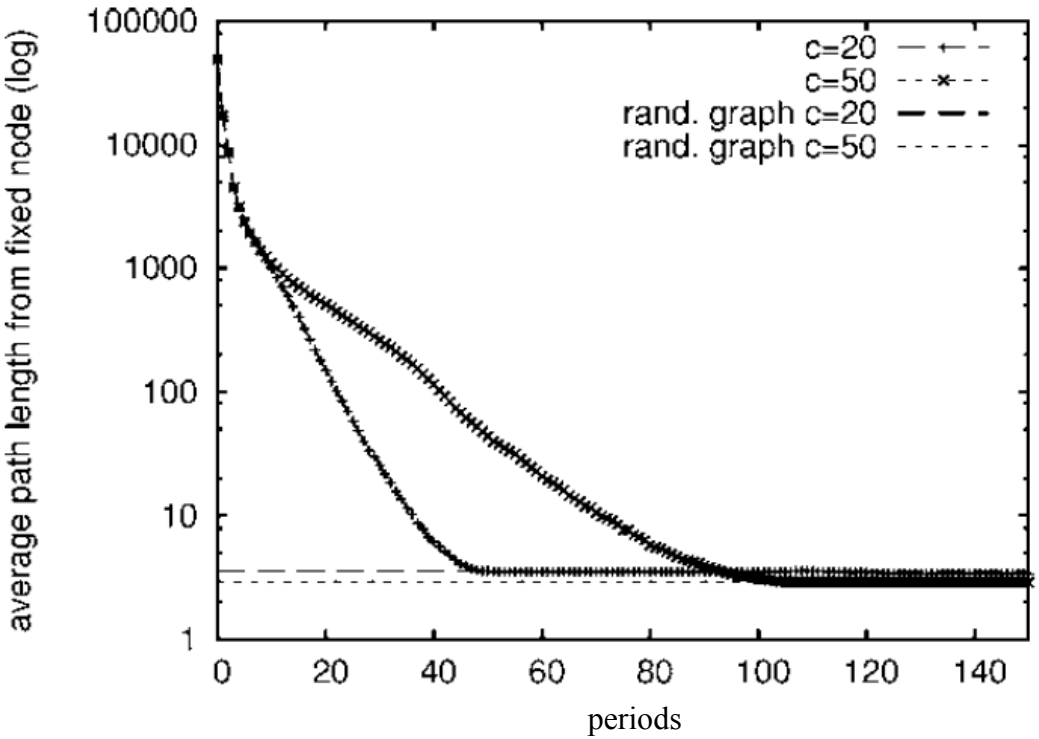
Source: Voulgaris et al
CYCLON, Inexpensive Membership Management for unstructured P2P overlays, 2005

CYCLON – Path Length

- Path Length
  - *Average shortest path*
    - The average number of edges between any two nodes
    - Represent the overall efficiency of the overlay
      - Number of network hops to reach a node from another node
      - Directly related to the cost of disseminating information or searching for information
      - Gives an idea for setting communication timeouts
  - Experiment
    - 100,000 nodes, shuffle period T
      - Typical shuffling period should be larger than twice the average network latency
      - Over wide area networks, period of 10s is good
    - During a period, all nodes have shuffled exactly once
  - Questions:
    - What will be the average shortest path?
    - How long will it take to converge to that value?

CYCLON – Path Length Convergence

- Small Shortest Average Path
  - From an initial chain topology (linked list)
  - Converges to an average around 3 and 4
  - Equivalent to random graphs (the reference)
- Fast Convergence
  - Within 40 to 80 periods
  - Between 7 and 14 mn (WAN)



Source: Voulgaris et al
CYCLON, Inexpensive Membership Management for unstructured P2P overlays, 2005

CYCLON – Convergence and Shuffle

- Initial Topologies
  - Chain: linked nodes
  - Star: one central hub

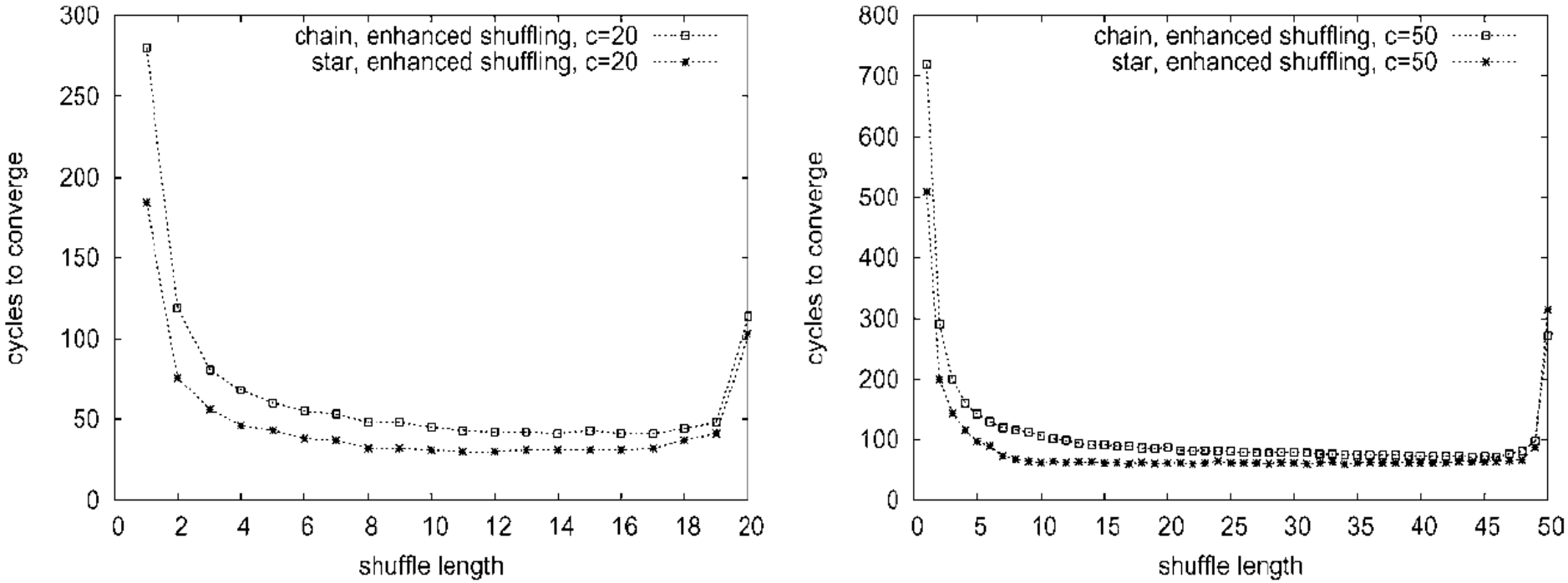
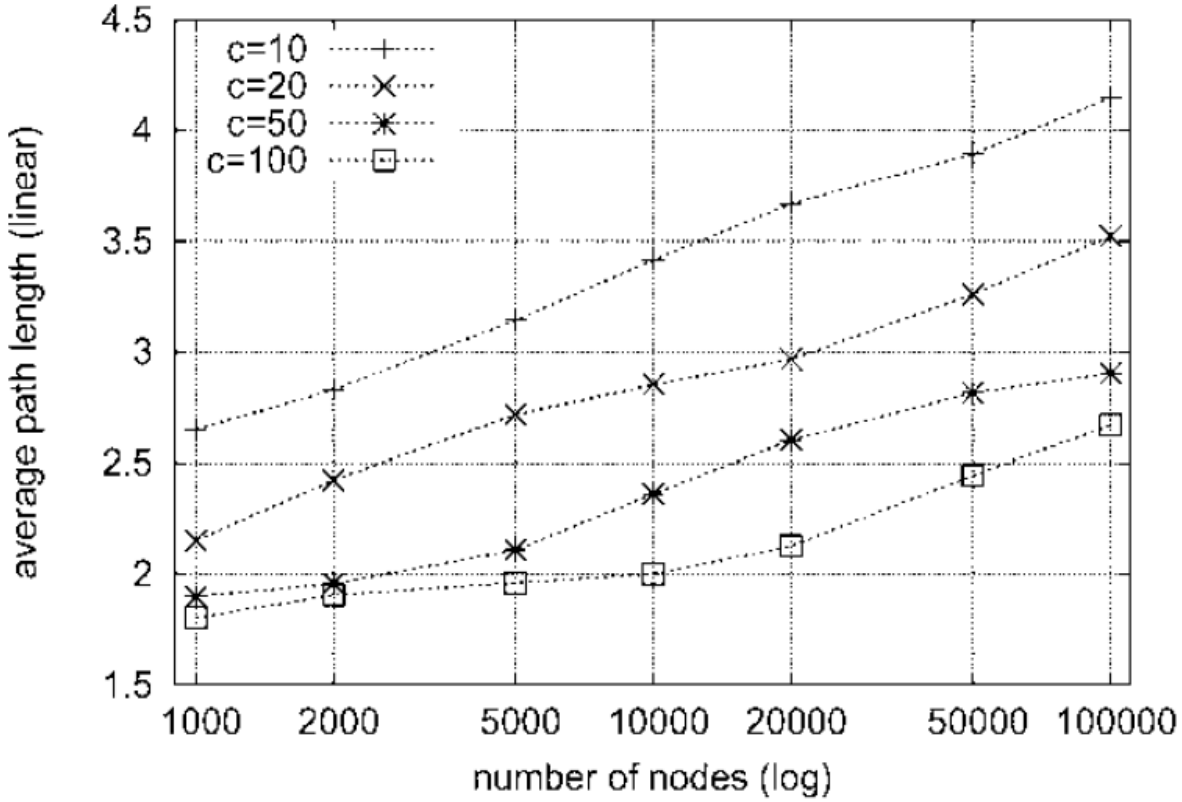


Fig. 5. Effect of shuffle length on convergence speed. $N = 100{,}000$.

Source: Voulgaris et al
CYCLON, Inexpensive Membership Management for unstructured P2P overlays, 2005

CYCLON – Path Length

- Path Length and Cache Sizes



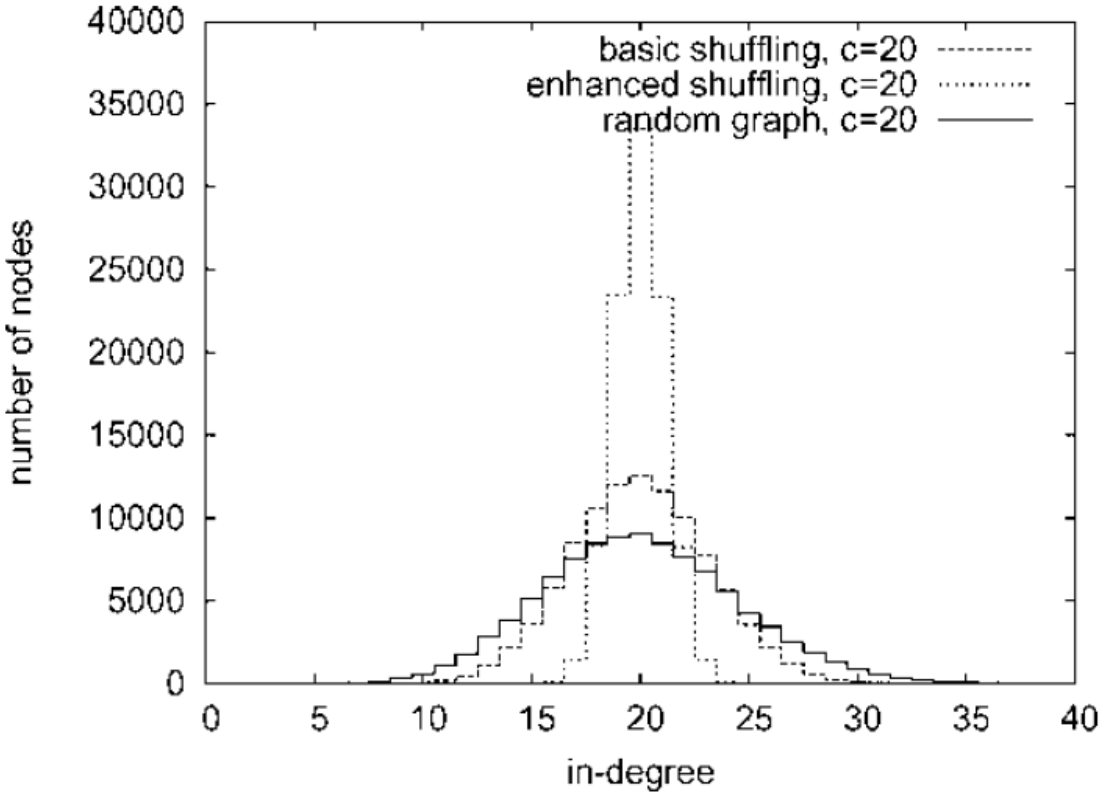
Source: Voulgaris et al
CYCLON, Inexpensive Membership Management for unstructured P2P overlays, 2005

CYCLON – Connectivity

- Degrees
  - Out-Degree
    - Number of outgoing edges
  - In-Degree
    - Number of incoming edges
- Importance
  - Failure robustness
    - Appearance of massively connected hubs versus somewhat isolated nodes
  - Indication of epidemic spread
    - Variations in degree induce irregular epidemic spread
  - Load balancing
    - Both regarding CPU and bandwith

CYCLON – Connectivity

- Cyclon Degrees
  - Out: fixed, this is the cache size
  - In: variable
- Discussion
  - Same number as random
  - Smaller deviation
  - Better design



Source: Voulgaris et al
CYCLON, Inexpensive Membership Management for unstructured P2P overlays, 2005

CYCLON - Bandwith

- Bandwith Considerations
  - Bandwith needed for gossip messages
  - Related to both the shuffle period and the size of the gossip information
- Fine Tuning
  - Gossip message
    - Per entry (10bytes): One IP address, a port number, an age
    - Message size = 10 * ShuffleLength
  - Shuffle Period
    - During each period, each node initiates a shuffle exactly once
  - Choice
    - ShuffleLength = 8
    - ShufflePeriod = 10s
  - Bandwith per node
    - Extremely low: 32 bytes per second (256bps)
    - Practival even over traditional modems (56kbps)