Fundamentals – Part One

Professor Olivier Gruber

Université Joseph Fourier

Projet SARDES (INRIA et IMAG-LSR)

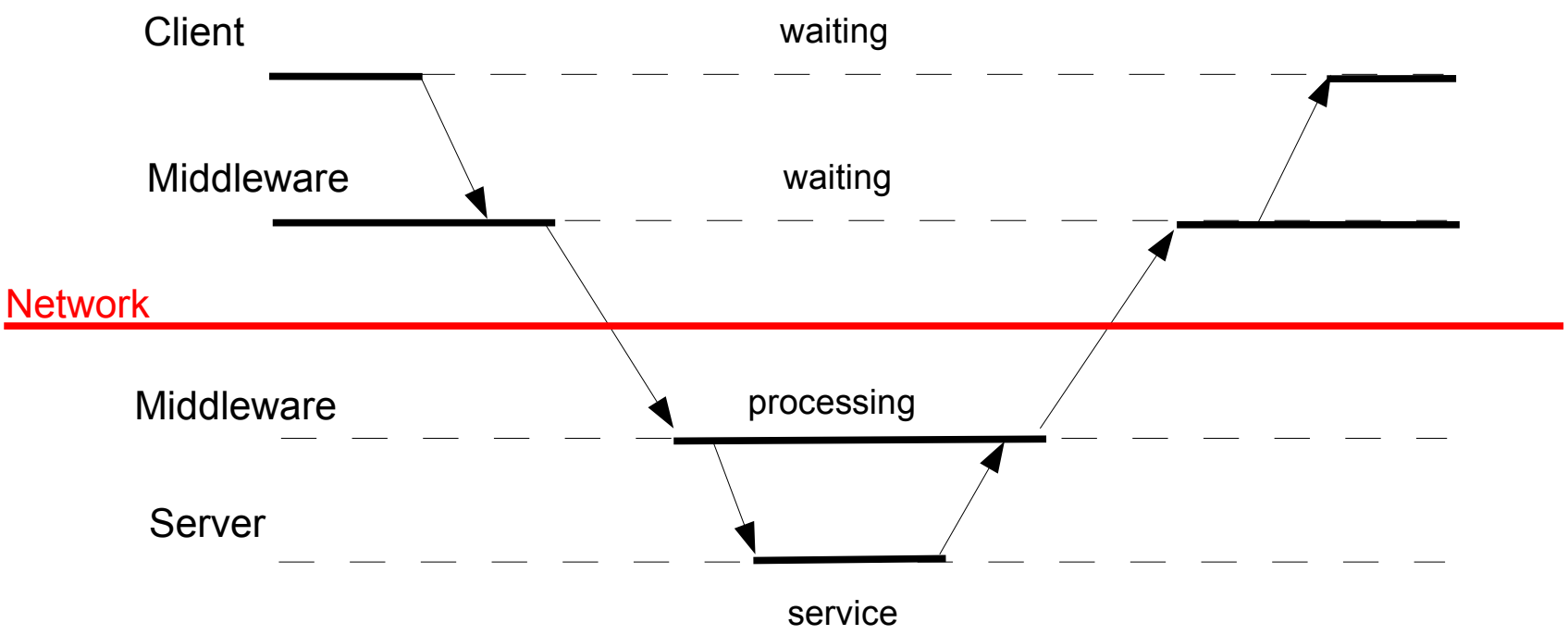
Olivier.Gruber@inria.fr

Outline

- The Message Paradigm
  - Architecture
    - Applications, middleware and operating systems
    - From the ISO stack to the middleware layer
  - Naming and routing
    - To send a message, one needs to name a destination
    - To deliver a message, one needs to find a route to the destination
  - Case Study:
    - The Internet
      - Simple sockets over TCP/IP
      - Domain Name Service (DNS)
      - Mobile IP
    - Peer-to-Peer Overlays
      - Structured overlays: Distributed Hash Table (DHT)
      - Unstructure overlays: Epidemic-style

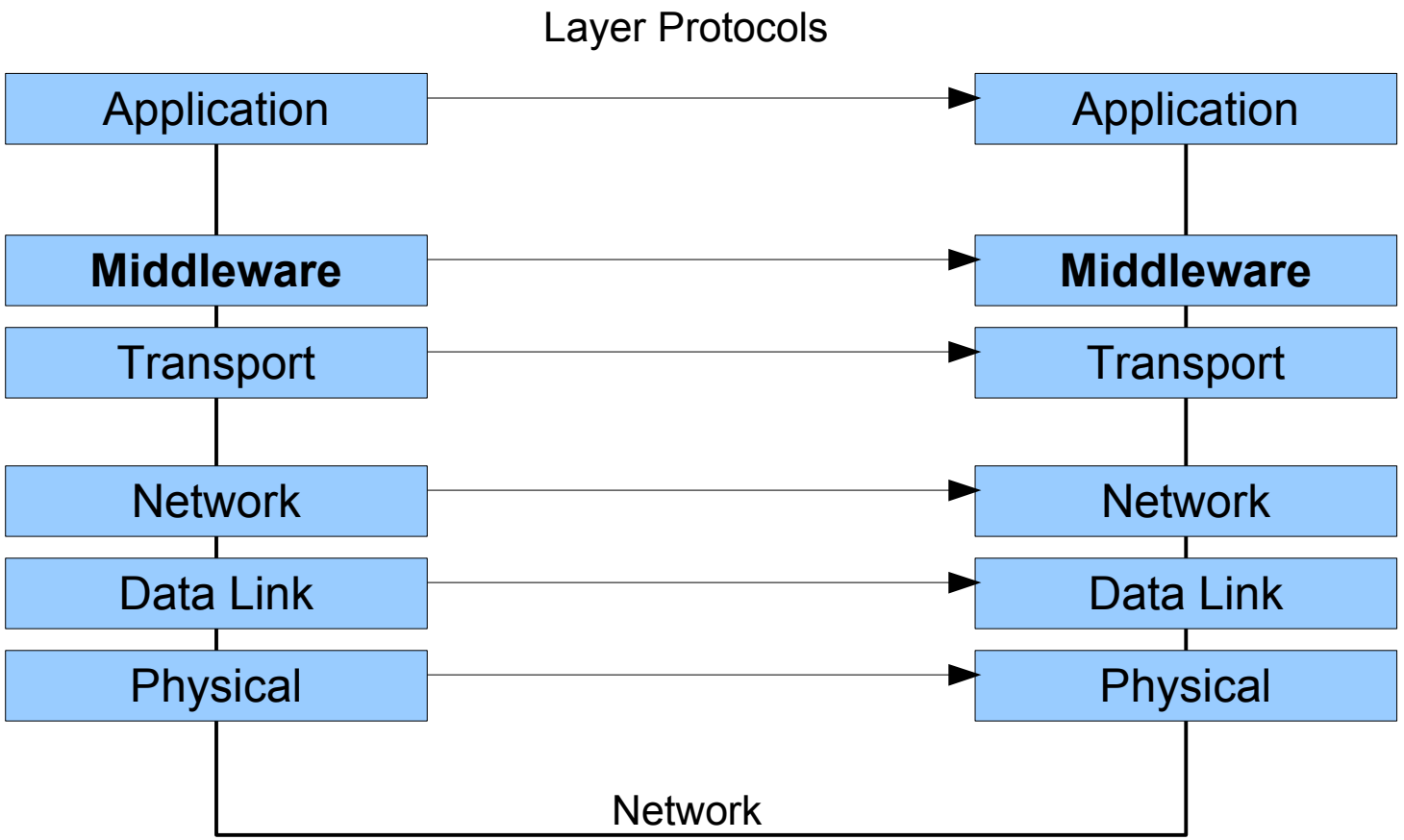
Message Fundamentals

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



Modified ISO Model

- Modified ISO model
  - Shortened stack, using a middleware and application layer
    - Replaces traditional session, presentation and application layers



Olivier.Gruber@inria.fr

Layered Protocols

- Physical layer
  - Transmits 0s and 1s, standardize hardware characteristics
- Data link layer
  - Detects and corrects error, groups the bits into frames, with checksums
- Network Layer
  - Essentially about routing of *network packets*
    - No routing within LAN
      - Just put the frame on the network, receiver takes it off
    - Physical route through LANs to reach the destination
      - Essentially about finding the *fastest* path (not always the shortest)
      - Depends on physic laws but also on-going traffic (queued messages)
  - Internet Procotol
    - Connectionless *Internet Protocol* (**IP**) is the de-facto standard
      - IP packets are sent without any setup
      - IP packets are routed independently from each others

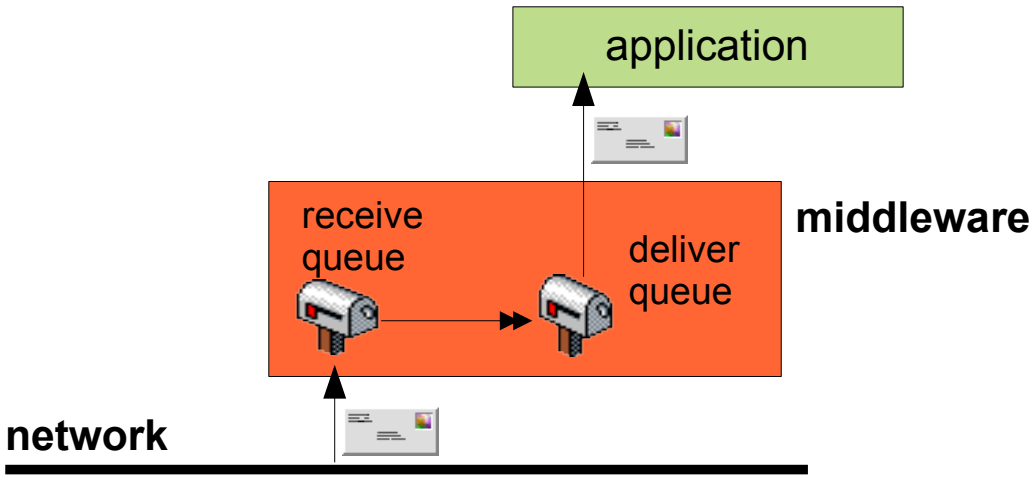
Olivier.Gruber@inria.fr

Layered Protocols

- Transport Protocols
  - Provides reliable connection-oriented communications
    - Support large messages (splits them in network packets)
    - **Ordered delivery without loss**
  - De-facto standards
    - Connection-oriented *Transmisssion Control Protocol* (**TCP**) over IP
    - Connection-less *Universal Datagram Protocol* (**UDP**), essentially IP++
- *Message-Oriented Middleware* (**MOM**)
  - Groups other general-purpose protocols
    - File Transfer Protocol (FTP), HyperText Transfer Protocol (HTTP) or email
    - Authentication and authorization protocols
    - Distributed commit and locking protocols
    - Multicast and group management protocols

Discussion Layers

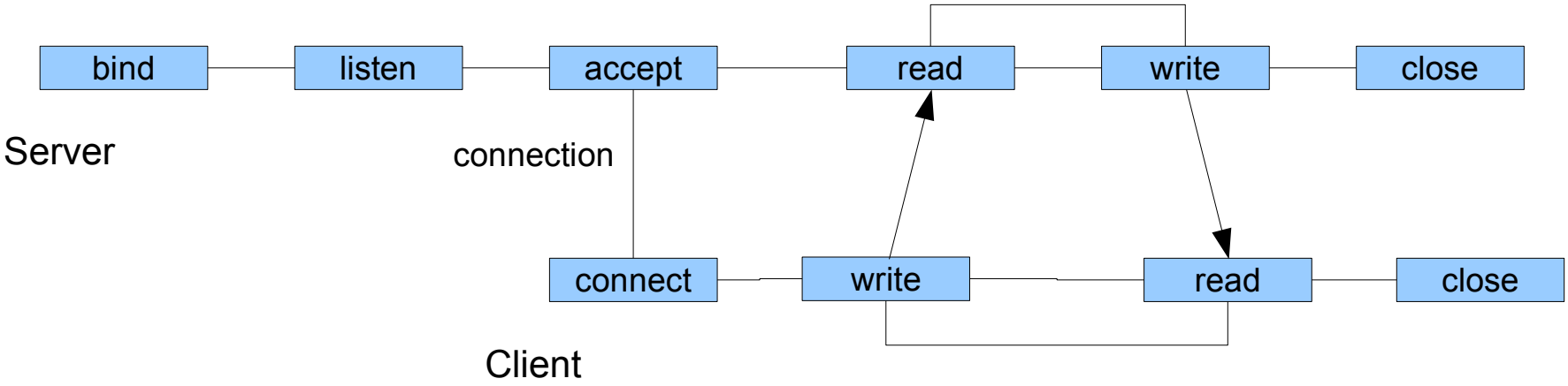
- Message-Oriented Middleware
  - Introduce a stronger separation than usual
    - Transient or persistent communications
    - Synchronous or asynchronous communications
    - Delivery order (total, causal, none)
    - Failures (lost or duplicate messages, lossless, FIFO)
  - Important to understand the difference between
    - The middleware receives a message
    - The middleware delivers a message to the application (process)



Olivier.Gruber@inria.fr

Discussion Layers

- 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
  - Protocol properties
    - UDP: delivered when received, messages may be lost and received out of order
    - TCP: FIFO and loss-less, delivered only when correct to do so



Discussion Layers

- Transient Communications
  - Messages are stored and forwarded
    - Delivered only if the recipient is active
    - Delivered only if there are no transmission interrupts
  - Example
    - Sockets over UDP over IP
      - 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
    - Sockets over TCP over IP
      - Chunk messages, FIFO delivery
      - Already a stronger distinction than UDP over IP

Discussion Layers

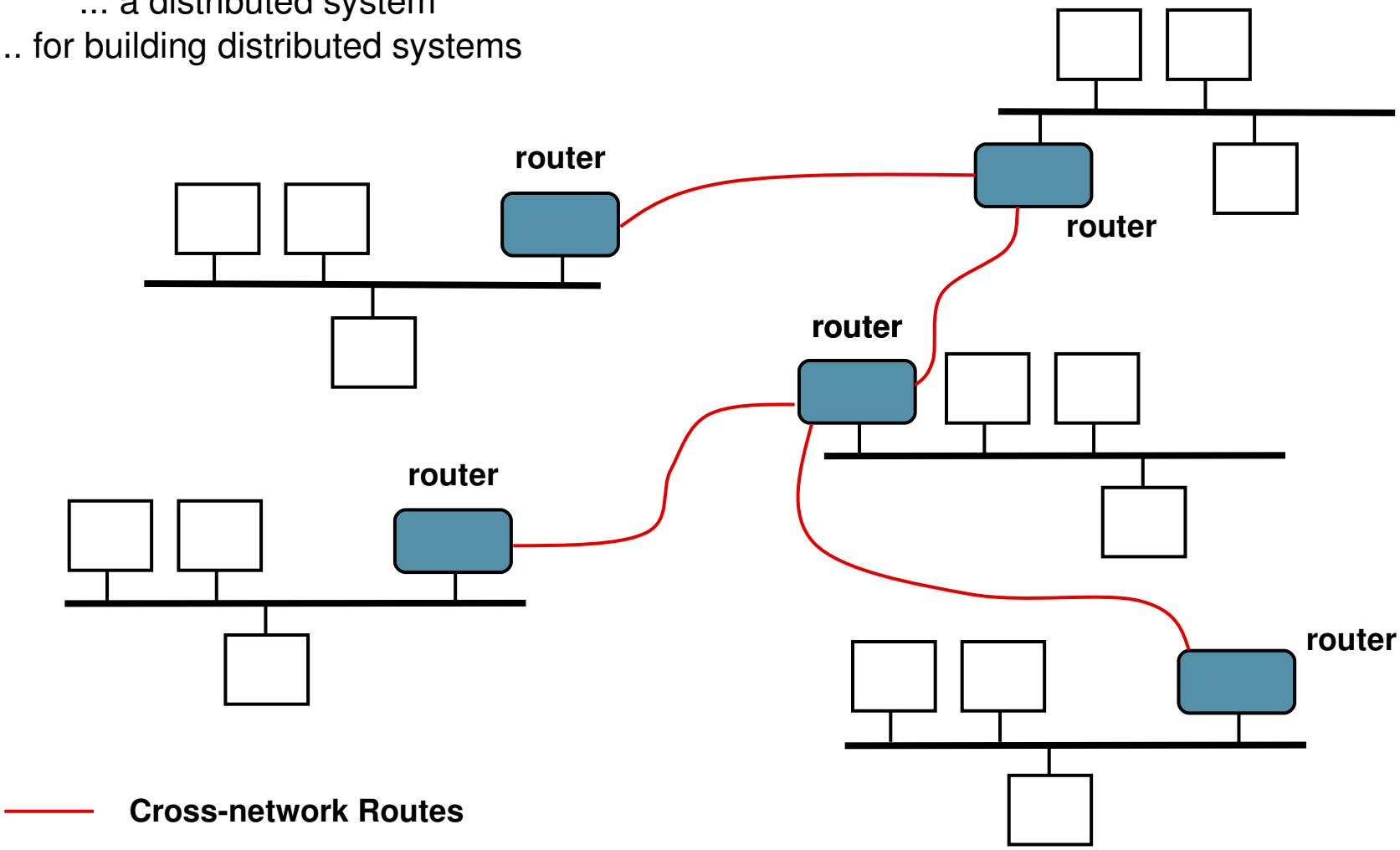
- Persistent Communications
  - Messages are stored by the middleware
    - Different policies as to how long a message is stored and delivery is attempted
  - Persistence may be a collaboration
    - Between different middleware stores on different network nodes
  - Example
    - The email system
      - Use SMTP/POP3 to send or receive emails from the email middleware
        - Complex overlay network of email gateways
        - Each gateway stores the email as it progresses towards its destination
      - Mailboxes store permanently delivered messages
    - Senders and receivers
      - Senders do not need to wait for delivery
      - Receivers do not need to be running for delivery to happen (mailbox)
      - Receive an asynchronous notification (email) in case of failed delivery

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



Olivier.Gruber@inria.fr

Discussing Names

- More 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
- Naming Service
  - Manage human-friendly names like in DNS
  - Manage the mapping from names to addresses
  - May be used for identity (unchanging name, changing IP)
- Discussing Middleware
  - The naming service is part of the message-oriented middleware
  - See how distributed systems are built from distributed systems

Discussing Names

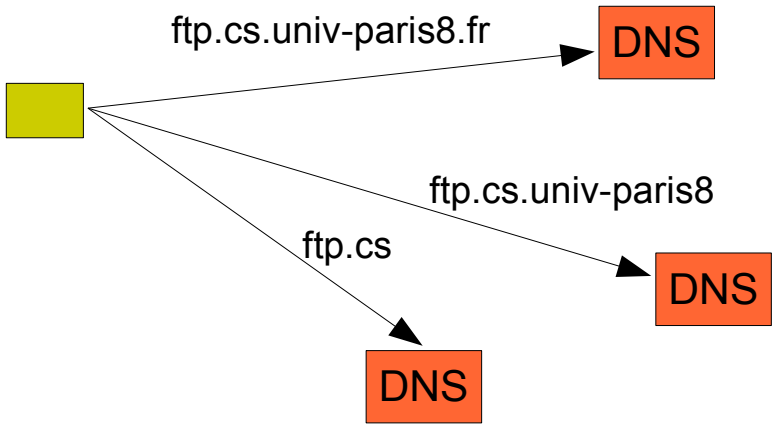
- Name Service Design
  - A centralized approach will not scale
    - Number of names
      - Millions on the Internet
      - A single point of failure
    - Wide-area networks
      - Unacceptable latencies to always go across the world to resolve a name
      - The closer the actual server, the more overloaded the network (routers and bandwith)
  - Need a collaborative and distributed approach
    - Close to the routing problem
      - Given a name, we need to be able to find a route to a server knowing that name
    - Distributed design
      - Distributed name servers will cooperate in managing name-address pairs

DNS – Domain Name Service

- Hierarchical Names
  - Map hierarchical domain names to IP addresses
    - *wikipedia.org* where wikipedia is subdomain of the org domain
    - *www.wikipedia.org* where *www* is the host in the *wikipedia.org* domain
  - Top-level domains
    - Countries (two-letter codes)
    - Domain generic with more than three digits (letters or numbers)
- Introducing DNS zones
  - A zone is a part of the name space managed on a separate name server
  - Hence, zones distribute name resolution
- Example
  - Resolving: *ftp.cs.univ-paris8.fr*

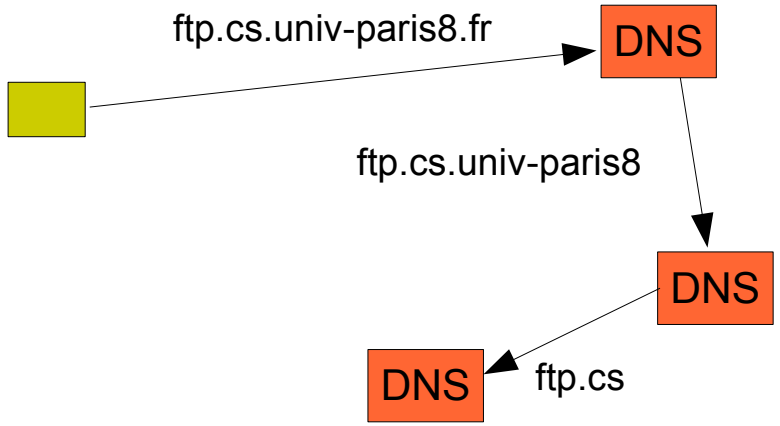
Name Resolution

- First Approach
  - Iterative Name Resolution
    - Client hands to one of the root servers the entire name
      - DNS has 13 well-known servers
      - Resolves as far as it can
      - Returns the next name server to contact and the unresolved suffix
    - Client repeats on the next server
      - With the unresolved suffix of the name
  - Example: *ftp.cs.univ-paris8.fr*
    - Resolves one label: *fr*
    - Resolves one label: *univ-paris8*
    - Resolves two labels: *cs.ftp*



Name Resolution

- Second Approach
  - Recursive Name Resolution
    - Client hands to one of the root servers the entire name
      - DNS has 13 well-known servers
      - Resolves as far as it can
    - Servers forward the request
      - With the unresolved suffix of the name
  - Example: *ftp.cs.univ-paris8.fr*
    - Resolves one label: *fr*
    - Resolves also one label: *univ-paris8*
    - Resolves two labels: *cs.ftp*



Discussing Name Resolution

- Iterative Name Resolution
  - Simpler protocol, lower load on name servers
  - Burden is on client middleware
    - Must iterate and manage connections and failures
    - Responsible for any caching done
      - Without caching, top-level domain servers are overloaded
      - Overall performance is dependent on well-behaving clients
- Recursive Name Resolution
  - Increased load on name servers
    - Manages connections between name servers and therefore eventual failures
  - Performs better
    - Leverage geographical proximity
      - In our example (ftp.cs.univ-paris8.fr), consider clients in the US
      - Less communication overheads than the iterative approach
    - Caching can be done within servers, at all levels

Other DNS Optimizations

- Name Aliases
  - One name may be mapped to several IP addresses
  - Different uses
    - Used for high availability, loading balancing (round-robin policy)
    - Also used for relocation, allowing to leave a forwarding address
    - Abstract names for public services such as ftp or Web servers
      - www.imag.fr => rillette2.imag.fr
      - rillette2.imag.fr => 129.88.34.211

Other DNS Optimizations

- Replicated Name Servers
  - Using name aliasing
    - Internally uses round-robin load balancing between replicated servers
  - Consistency protocol
    - Only one writer
      - Updates happen to the primary copy
    - Replicas request zone transfers
      - Acceptable to return outdated information
      - Eventual consistency
- Discussing performance
  - Top-level zones are expected to have few and rare updates
  - Local updates are often local names only used locally
  - So caching and replication are highly effective

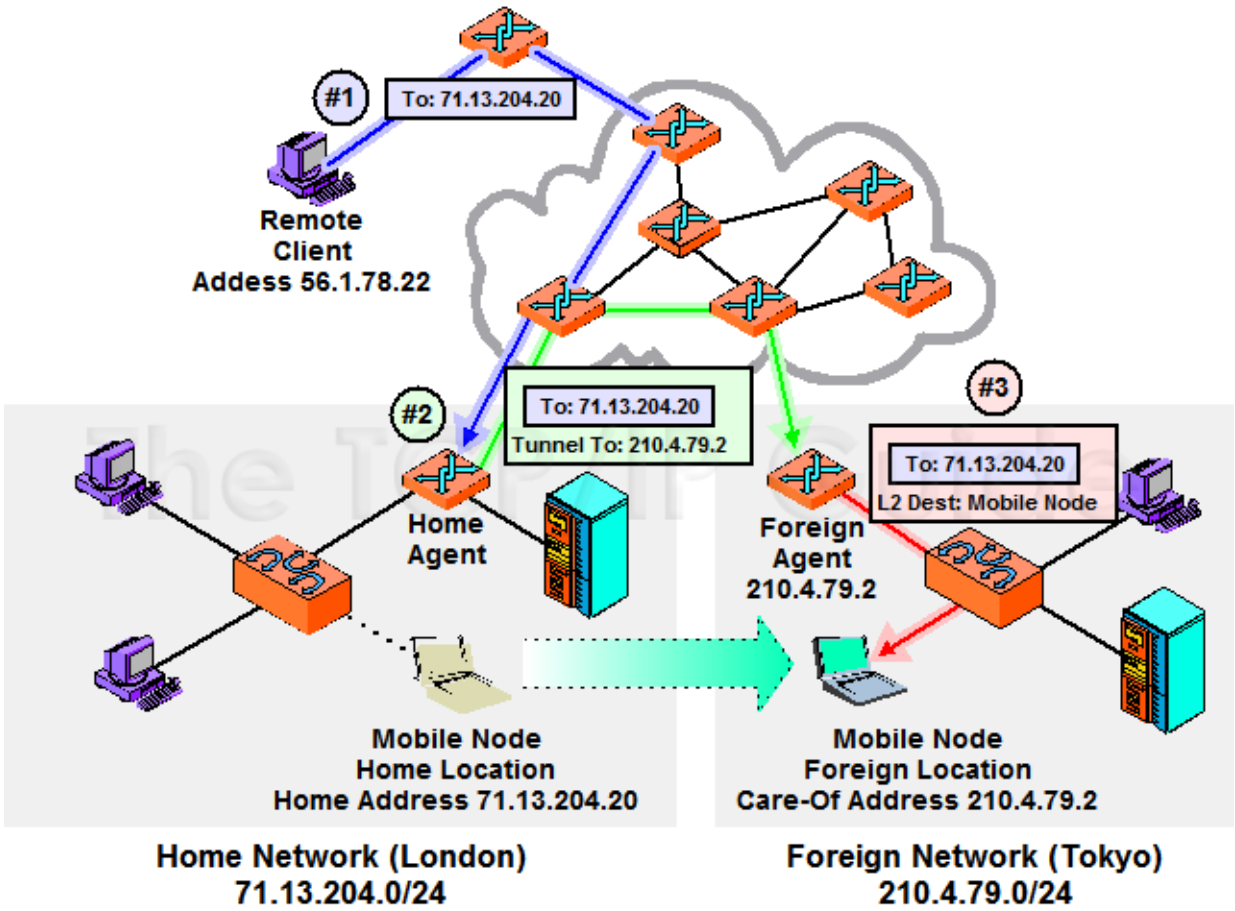
Internet Summary

- IP Addresses
  - 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
    - *wikipedia.org* where wikipedia is subdomain of the org domain
    - *www.wikipedia.org* where *www* is the host in the *wikipedia.org* domain
  - Optimized distributed system
    - Caching, replicated servers

Mobile IP

- The Problem
  - Wanting to using IP has identity while allowing mobility
  - Difficult since IP is by definition location-dependent
- The Home-based Solution
  - When at home, a mobile equipment has
    - A home IP address with a home agent on its local network
    - Home agent is typically a router
  - When not at home, it also has
    - A care-of agent on its current local network (also typically a router)
    - A local IP address on that same local network
  - Principle
    - Registration
      - The care-of agent will notify the home agent of its own IP address
    - Routing
      - Home agent will tunnel datagram packets to the care-of agent
      - Mobile IP routing shortcuts the home agent for further datagram packet routing

Mobile IP



Source: http://www.tcpipguide.com

Discussing Mobile IP

- Drawbacks
  - Fixed home location, must exist and be available (the agent)
  - May incur around-the-globe communication to find mobile devices
  - Does not support well long-term or definitive relocation
- Forwarder Paradigm
  - Typical example of a forwarder paradigm
  - Only works well:
    - For limited and transient mobility
    - For persistent long-live homes
  - Would require a complex overall solution
    - Short-cut forwarders, updating name servers
    - Reclaim forwarders, making effectively the changing address the identity
    - But requires something like distributed garbage collection
    - Quite difficult given that IP addresses and names may be written on a piece of paper...

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

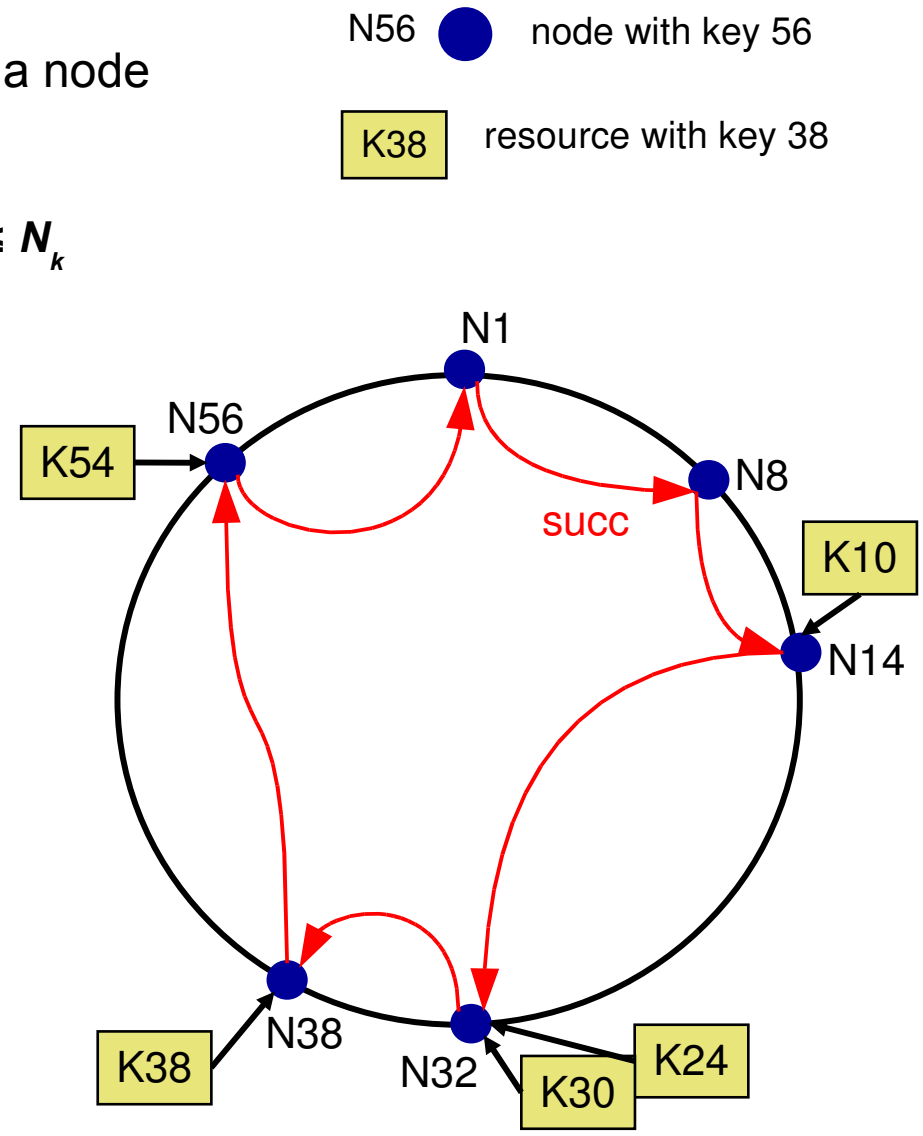
- Adopting Identity
  - Entities are identified by m-bit keys
    - The key space is usually 128 or 160 bits
  - Entities may be anything
    - Host, processes, files, etc.
- Distributed Nodes
  - Each node is responsible for managing certain keys
    - A node store the resources for the keys it manages
  - 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



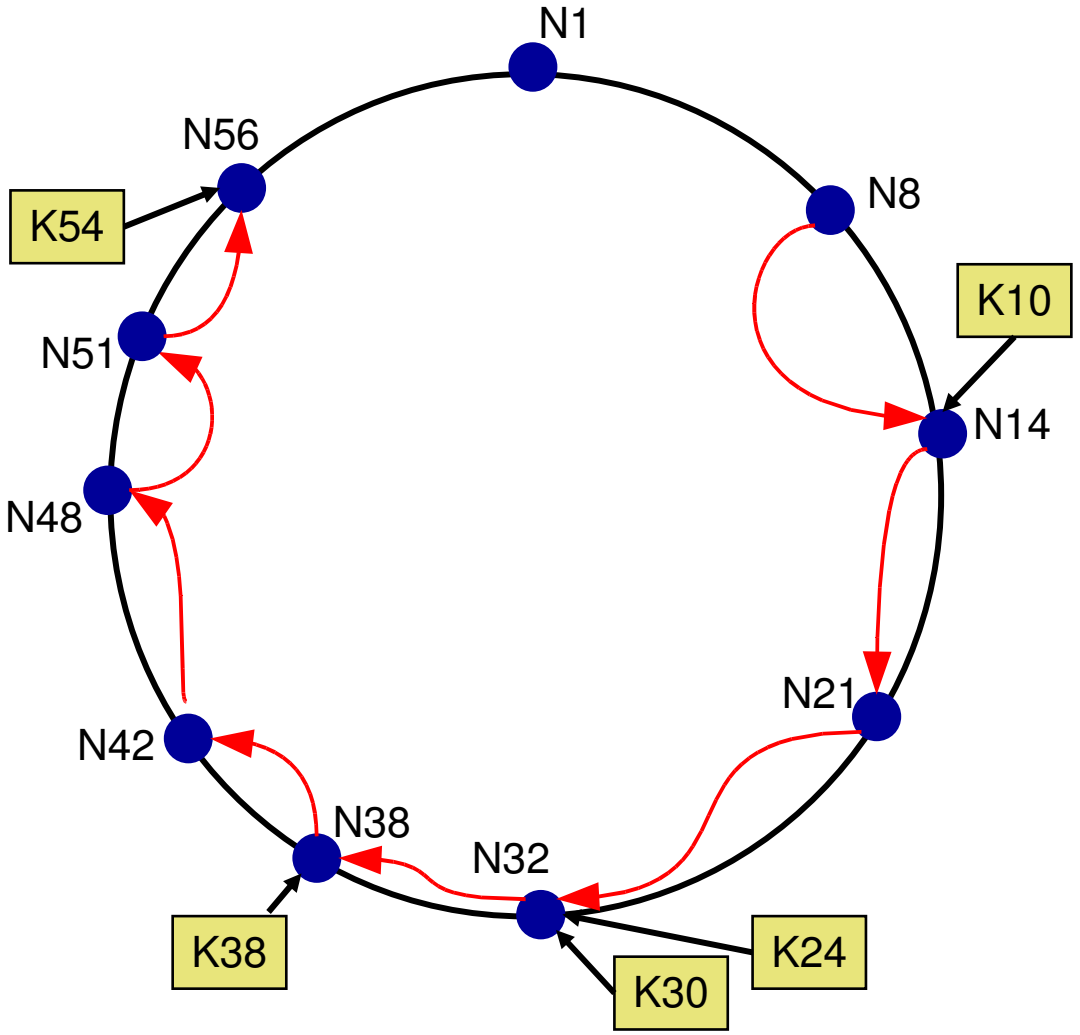
Olivier.Gruber@inria.fr

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

Olivier.Gruber@inria.fr

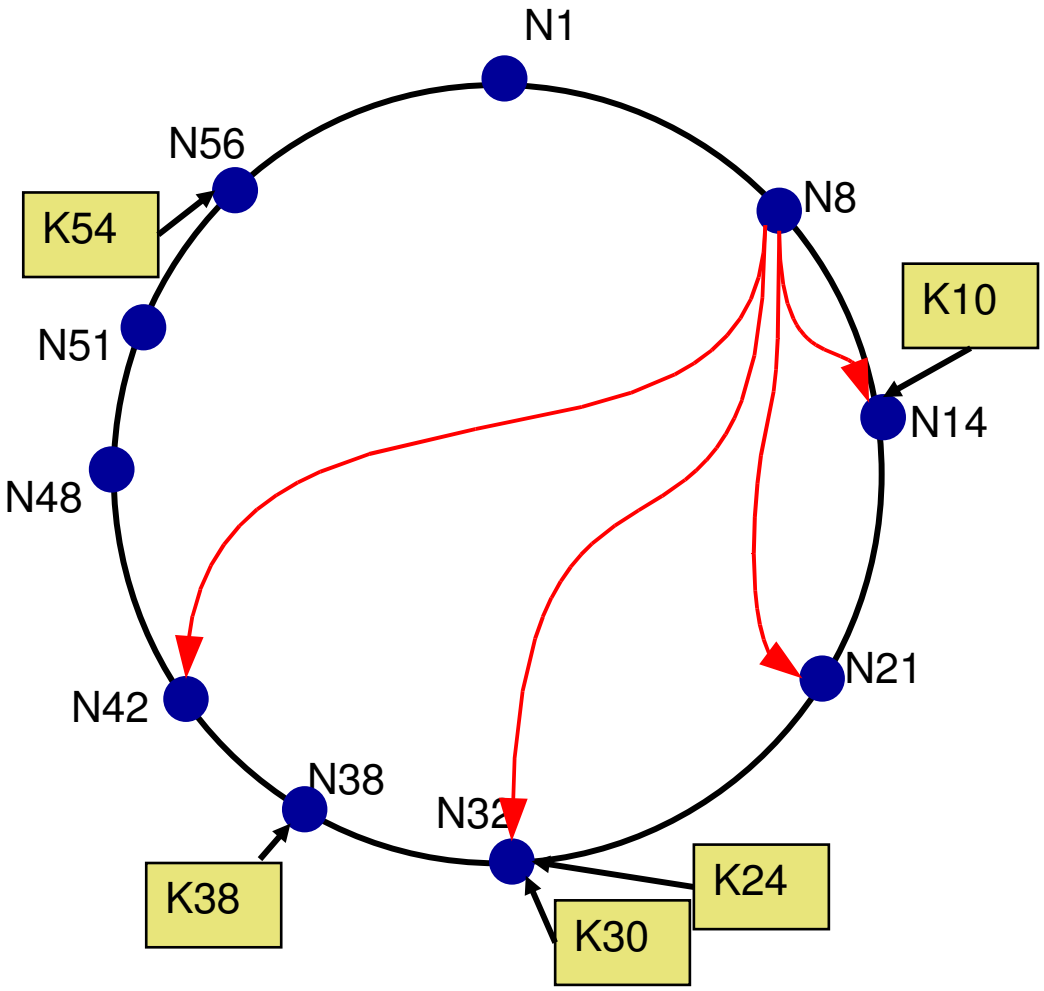
Chord – Finger Table Principles

- 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:

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

Olivier.Gruber@inria.fr

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

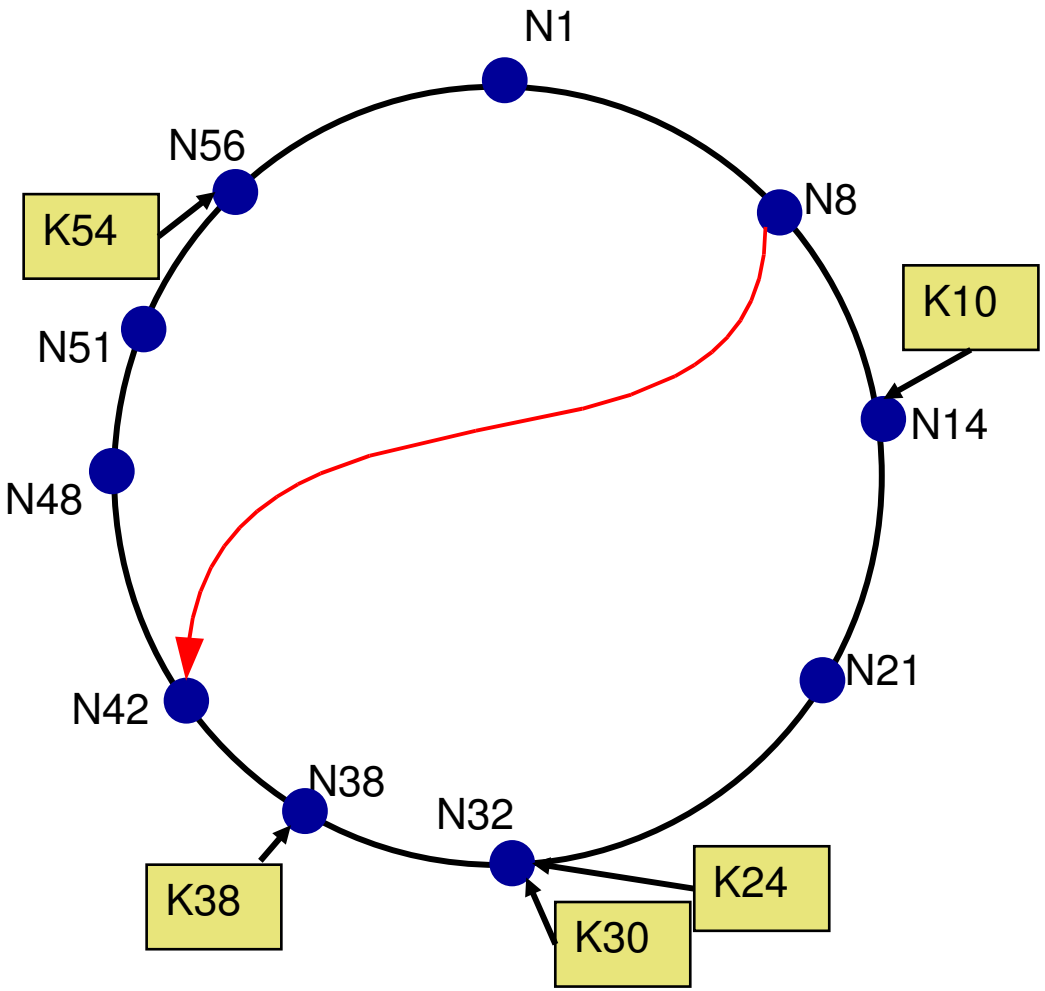
with $1 \le k \le m$

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

Olivier.Gruber@inria.fr

Chord – Lookup with Finger Tables



DHT: m = 6, 2^m = 64

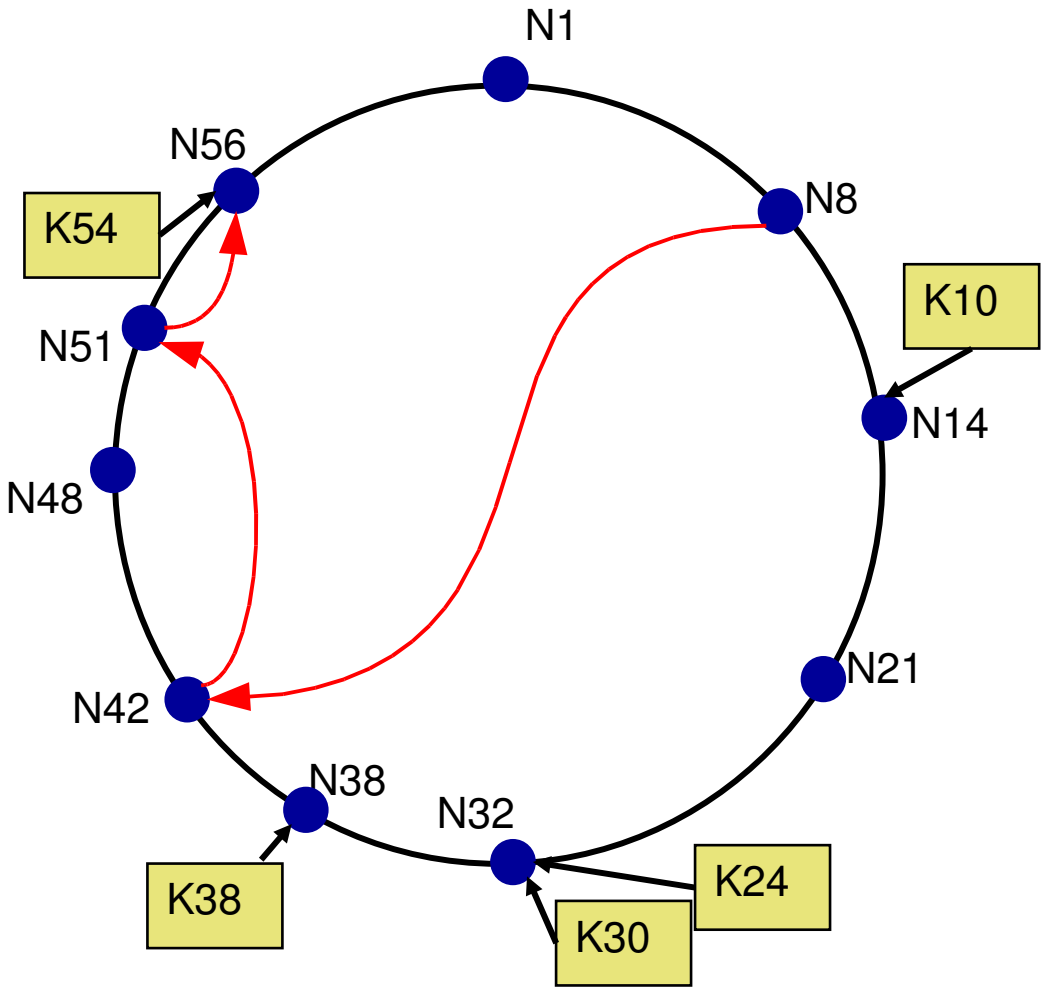
Looking up key 54, from N8

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

Finger Table for N8

Olivier.Gruber@inria.fr

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

Olivier.Gruber@inria.fr

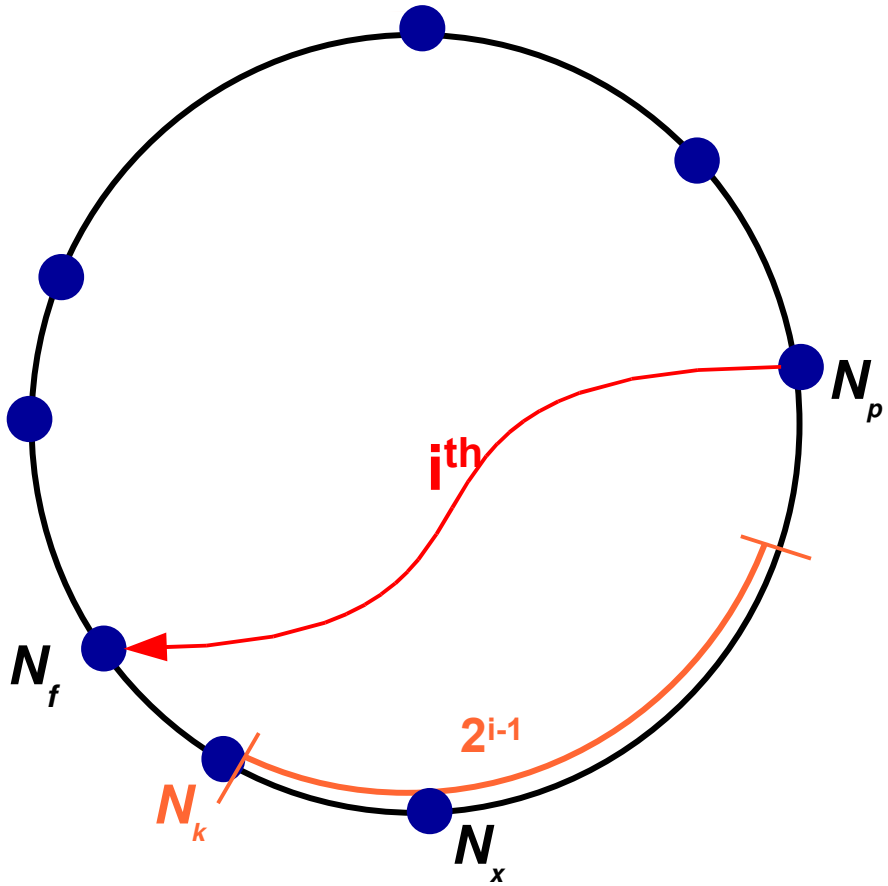
Chord – Joining or Leaving

- 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 $\boldsymbol{N_k}$
    - Find through any node in the ring the succ($\boldsymbol{N_k}$)
      - Insert itself before that node in the ring
      - Builds finger table, asking for succ($\boldsymbol{N_k}+2^{i-1}$) 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)*

Olivier.Gruber@inria.fr

Chord Summary

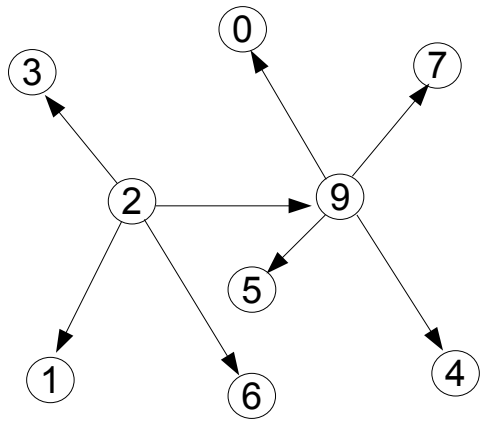
- Distringuishing features
  - Simplicity and provable correctness and performance
- Lookup Performance
  - With high probability, we have O(log N) messages to lookup a key
    - The average is therefore 0.5 log(N) messages (normally distributed keys)
  - Finger Tables
    - In a m-bit space of keys, 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) }

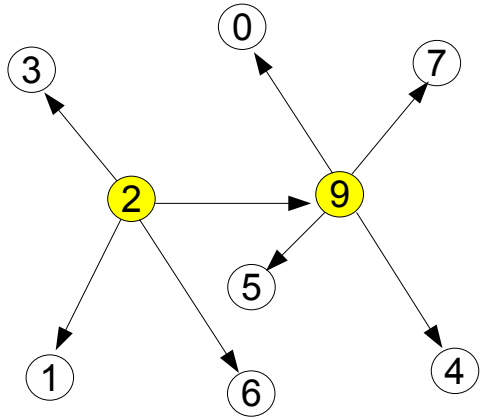
    - Neighbors are randomly chosen
    - Neighbor lists are exchanged
  - Epidemic broadcast
    - To find something, broadcast on the overlay
    - With high-probability, it will be found quickly (just a few network hops)



Olivier.Gruber@inria.fr

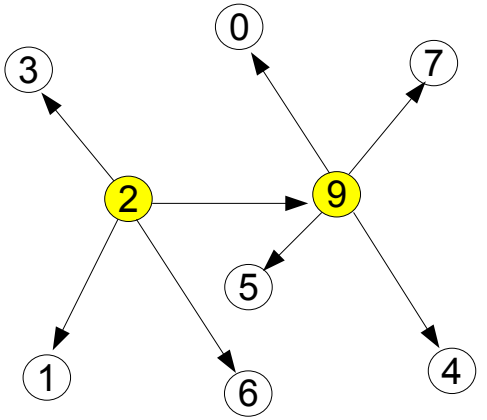
Unstructured Overlays

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

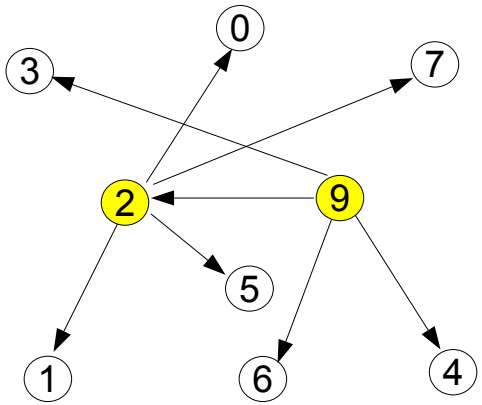


Shuffle Algorithm

- 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}



Olivier.Gruber@inria.fr

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

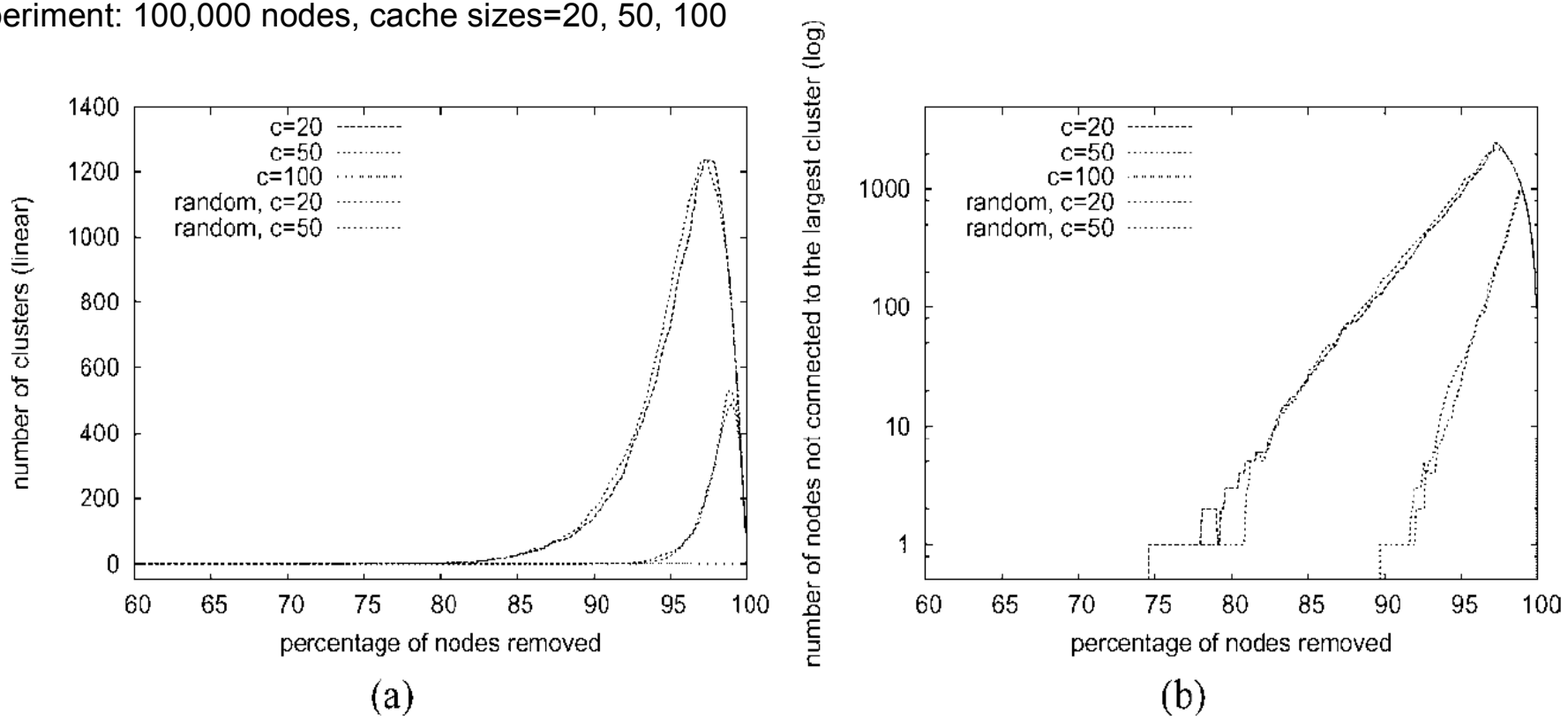
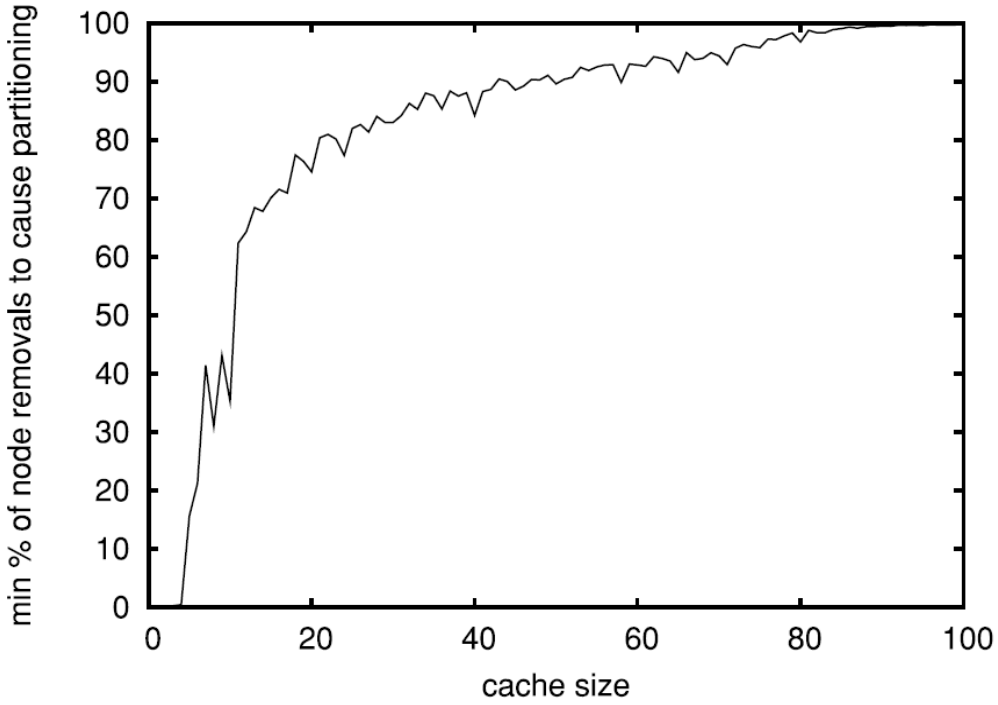


Fig. 7. (a) Number of disjoint clusters, as a result of removing a large percentage of nodes. Shows that the overlay does not break into two or more disjoint clusters, unless a major percentage of the nodes are removed. (b) Number of nodes not belonging to the largest cluster. Shows that in the first 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

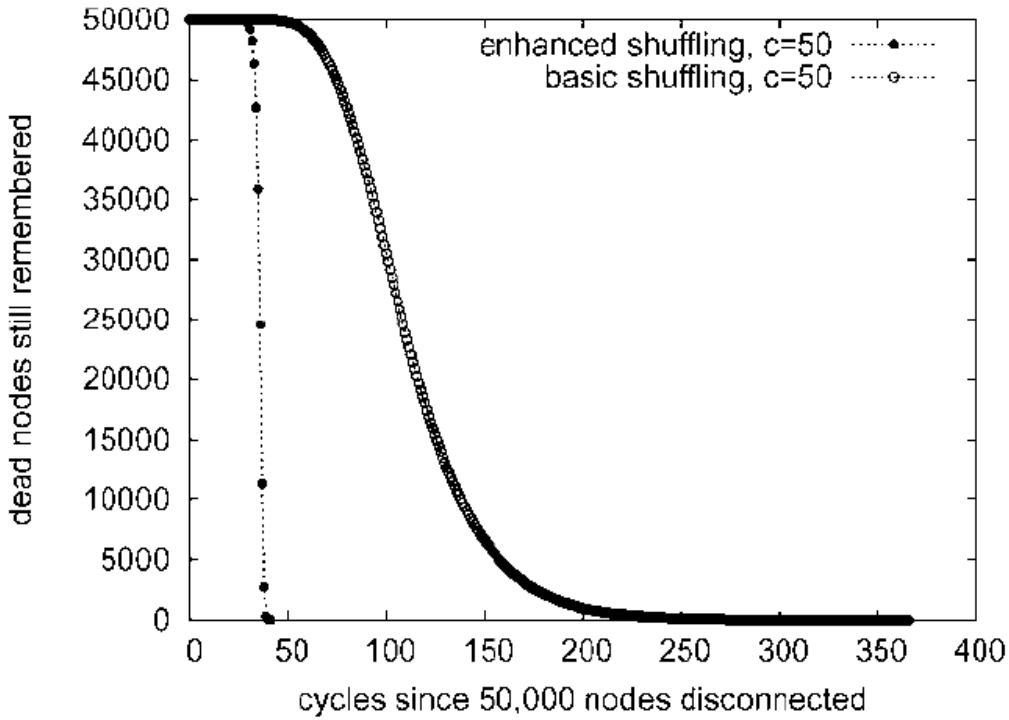


Source: Voulgaris et al
CYCLON, Inexpensive Membership Management for unstructured P2P overlays, 2005

Olivier.Gruber@inria.fr

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



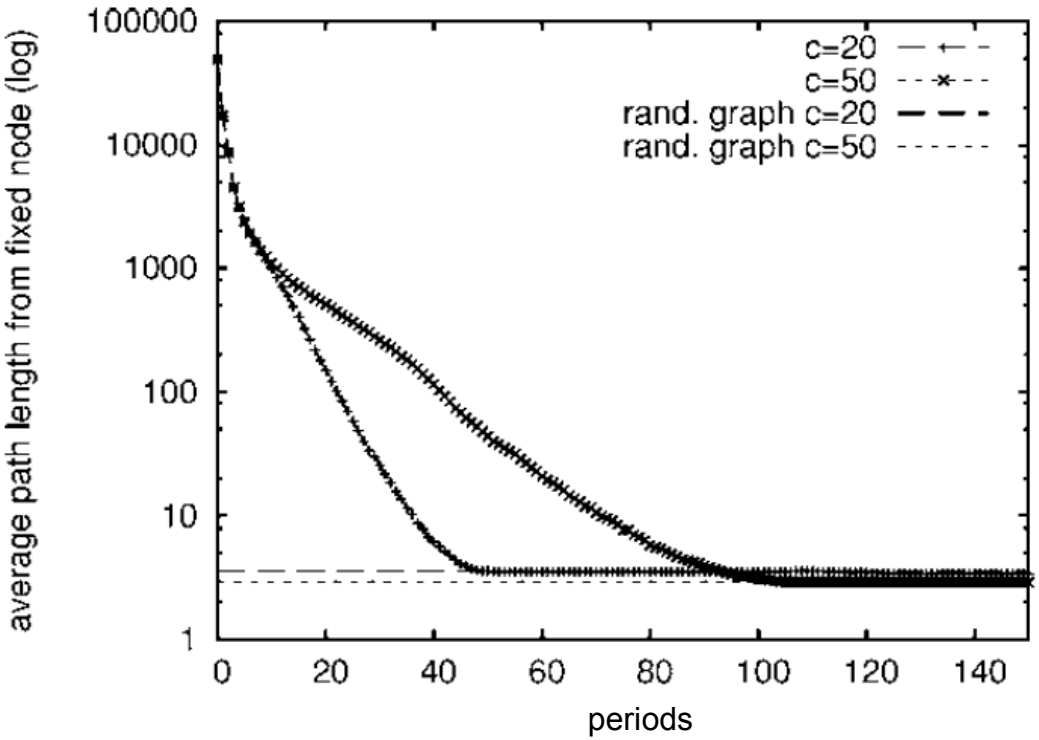
Source: Voulgaris et al
CYCLON, Inexpensive Membership Management for unstructured P2P overlays, 2005

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

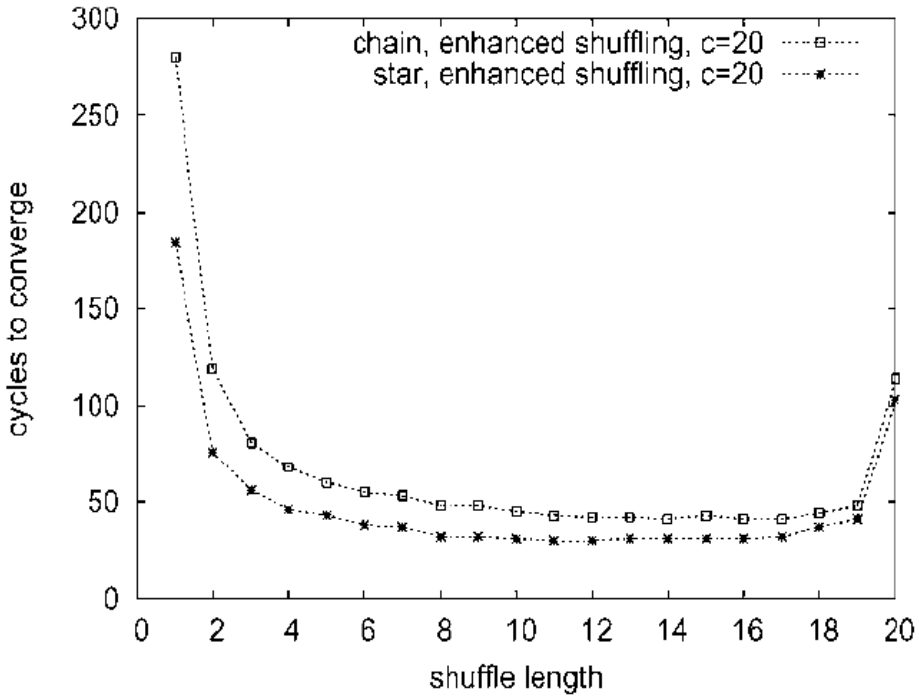
- Small Shortest Average Path
  - From an initial chain topology (linked list)
  - Converges to an average around 3 and 4
  - Equivalent to random graphs (the reference)
- Fast Convergence
  - Within 40 to 80 periods
  - Between 7 and 14 mn (WAN)



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



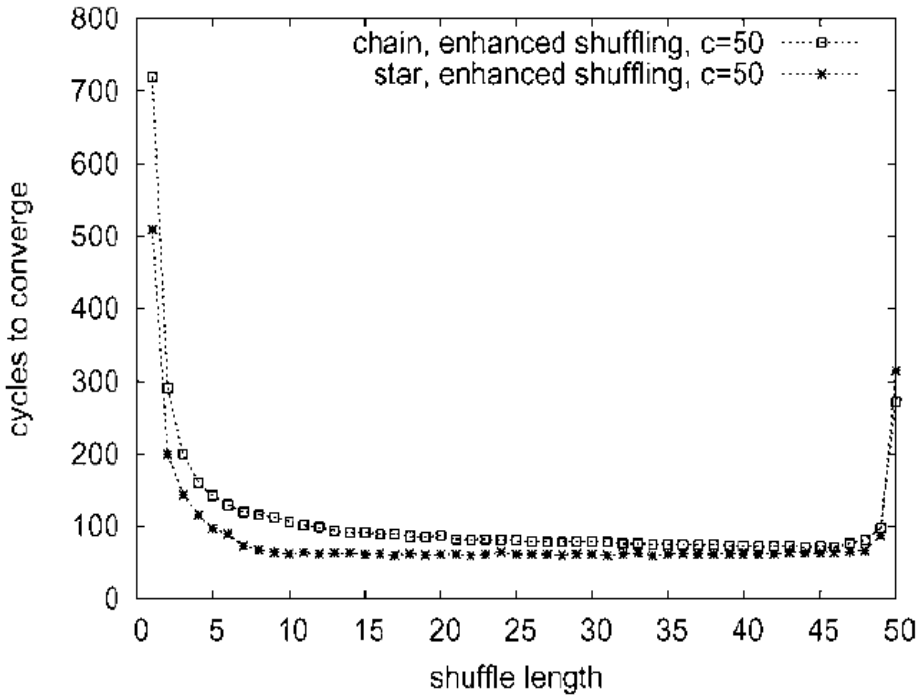


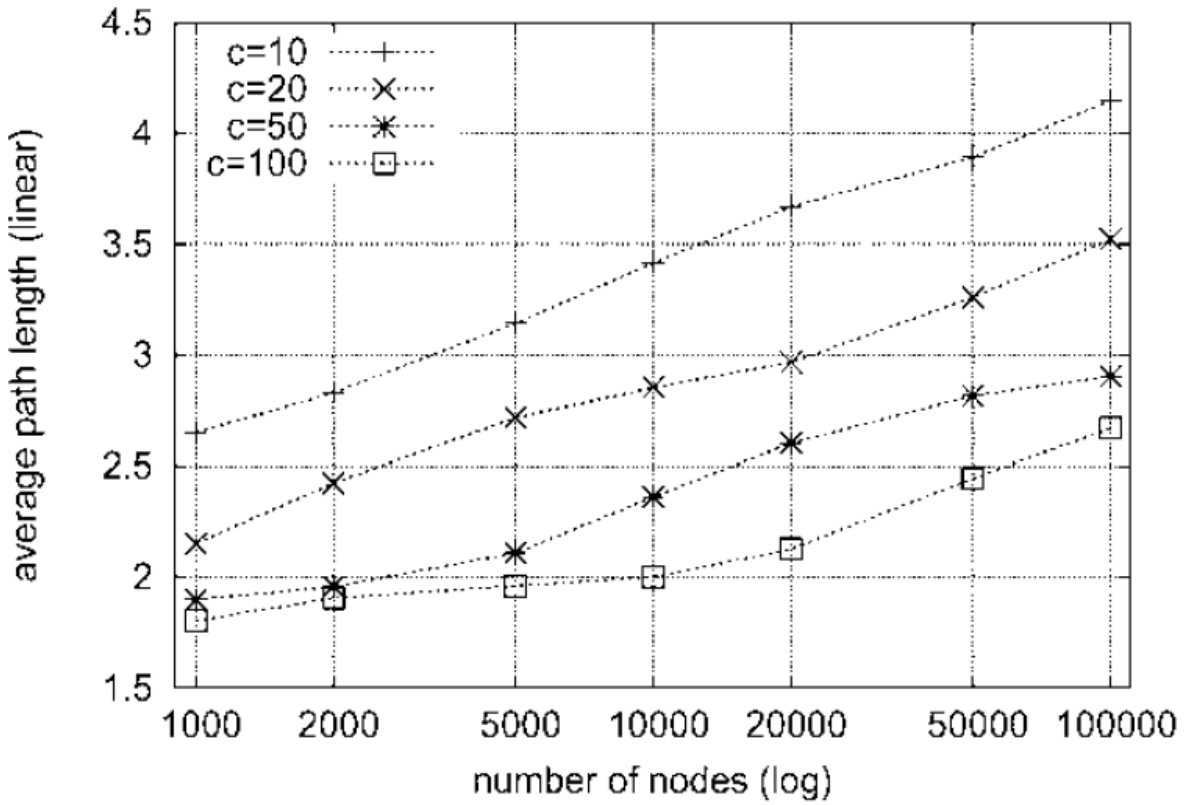
Fig. 5. Effect of shuffle length on convergence speed. $N = 100{,}000$.

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

Olivier.Gruber@inria.fr

CYCLON – Path Length

- Path Length and Cache Sizes



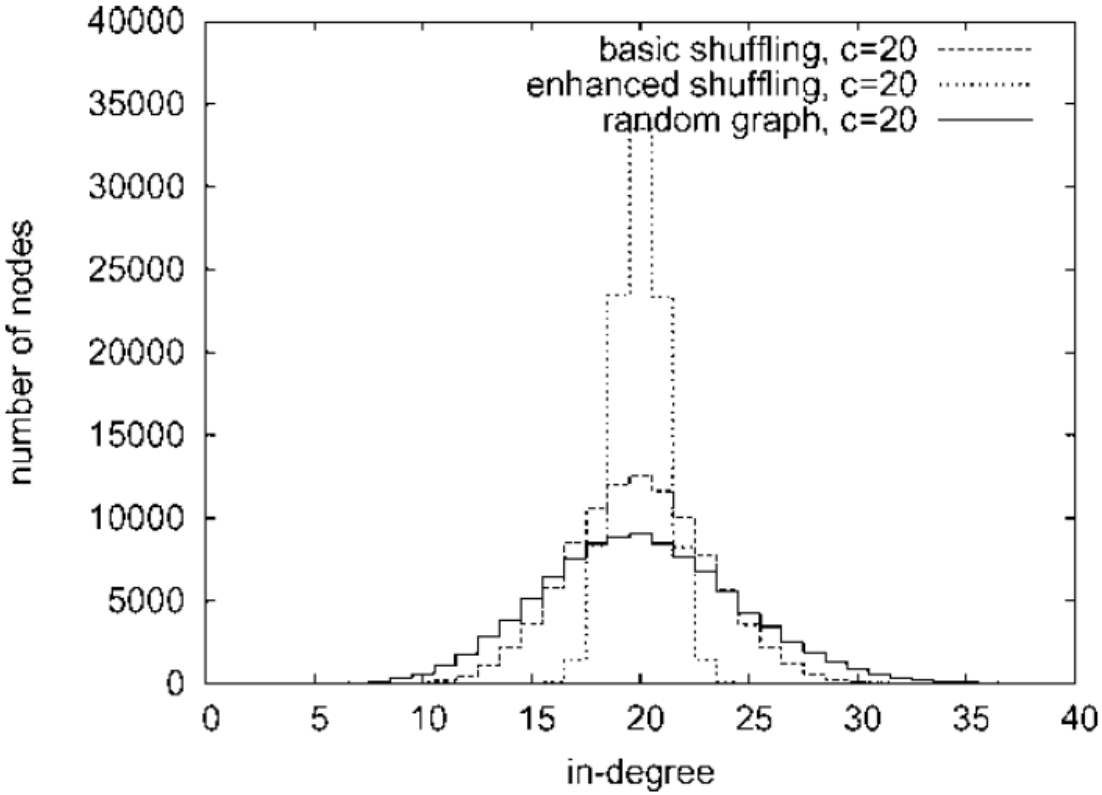
Source: Voulgaris et al
CYCLON, Inexpensive Membership Management for unstructured P2P overlays, 2005

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)

Olivier.Gruber@inria.fr