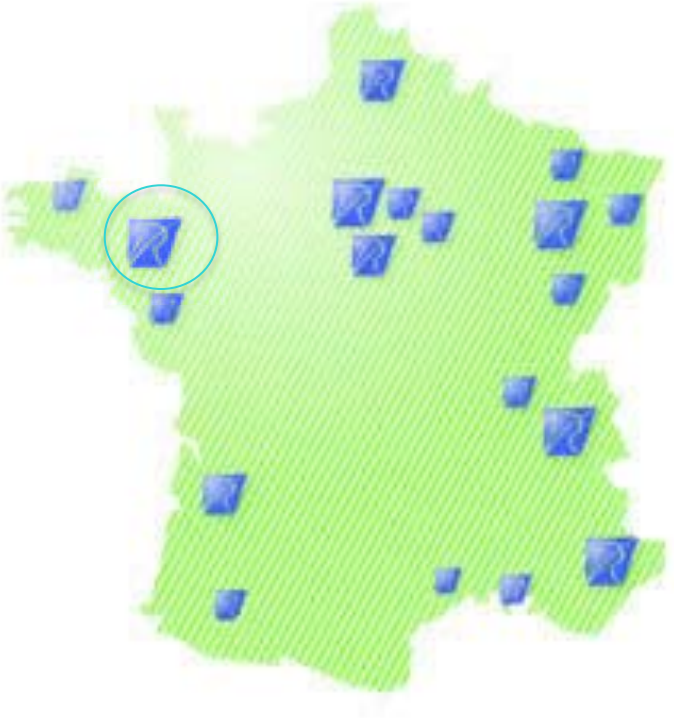
Peer to peer systems and applications

Anne-Marie Kermarrec
INRIA Rennes, France
ASAP group
(As Scalable As Possible)



Roadmap

1. Introduction
2. Structured P2P systems
3. Application-level multicast over structured P2P networks
4. Epidemic-based peer to peer systems and applications

The road to P2P systems

Scaling Techniques

Scalability: ability to increase performance (or not to decrease) as the scale increases

Distribution: partitioning of data and computation across multiple machines

- Decentralized naming service (DNS)
- Decentralized information systems (WWW)

Replication: make copies available at multiples machines

- Replicated file.Web servers
- Replicated databases

Caches: allow client to access local copies

- Web caches (browser/proxy)
- File caching
- Distributed shared memory systems

Context

- Distributed systems are evolving
  - Large-scale distributed systems
    - Number of machines, geographical spreading and data volume
  - Dynamic behaviour
    - Mobility, volatility, connectivity

  Traditional algorithms are no longer efficient

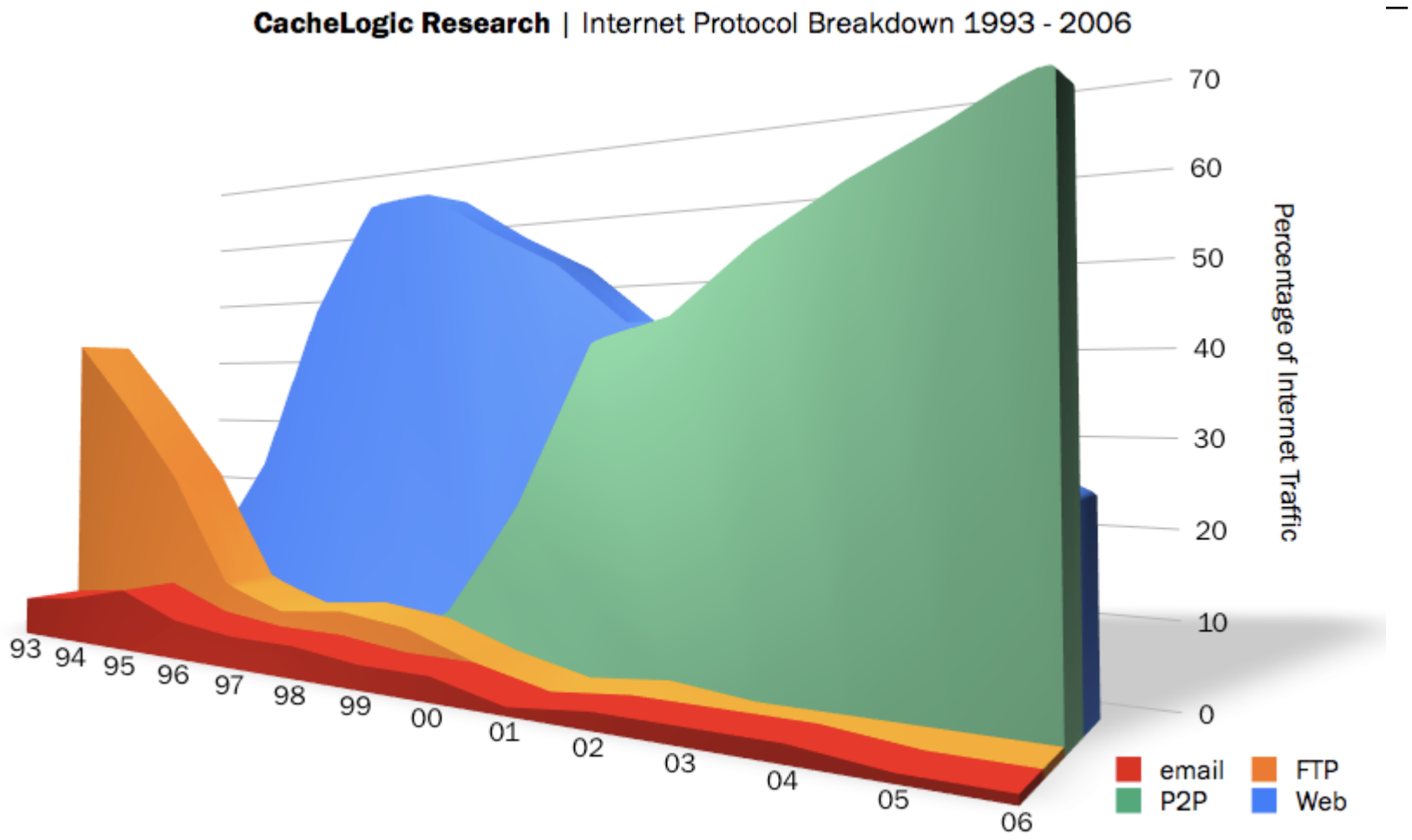
- Peer to peer communication paradigm fills this gap
  - Fully decentralized
  - Self-organizing/enhanced availability
  - Symmetric peers/load balancing
  - Local knowledge of the system/global convergence

Peer-to-Peer Systems

Historical perspective

- 1970s - 1980s: Birth of the Internet
  - Limited reach of the Internet
  - Email, FTP, Telnet
  - Share documents and resources between research centers
  - Central committee to organize and maintain it
- 1990s
  - Tremendous expansion & diffusion
  - Killer apps: WWW and e-Commerce
  - Client/Server model
- Late 1990s - today
  - P2P: An alternative to Client/Server
  - Passive clients → **active peers**
  - End-computers play a role, contribute, interact

Internet Traffic



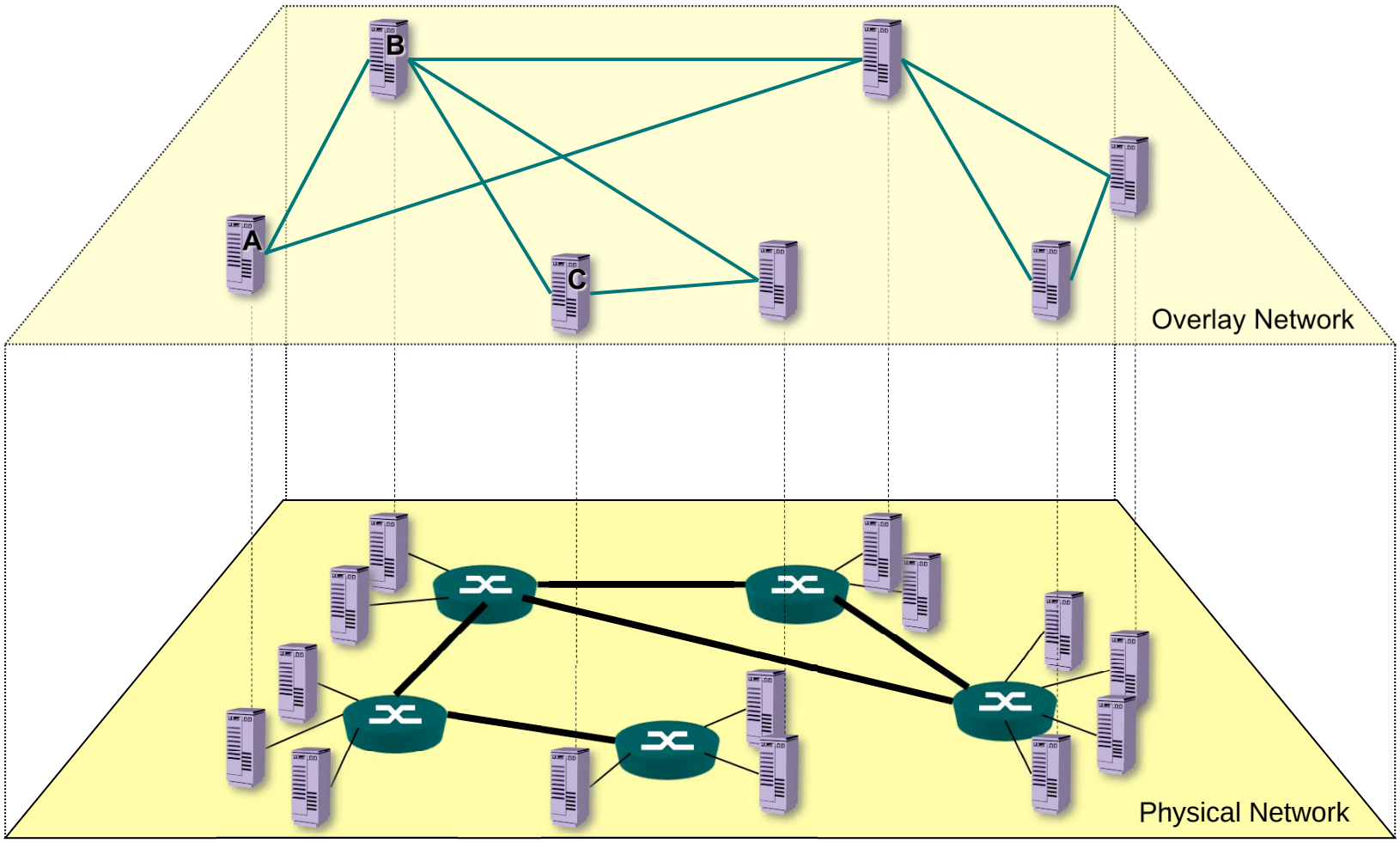
What makes P2P interesting?

- End-nodes are promoted to active components
  (previously they were just clients)
- Nodes participate, interact, contribute to the services they use.
- Harness huge pools of resources accumulated in millions of end-nodes.
- Irregularities and dynamicity are treated as the norm

Peer to peer applications

- Deployed application (60-70% of Internet Traffic)
- Archival systems
- RW file sharing applications
- Application-level multicast
- Streaming content protocols
- Publish-subscribe systems (RSS)
- Grid Computing
- Telco applications

The core: Overlay Networks



Overlay types

Unstructured P2P	Structured P2P
○ Any two nodes can establish a link ○ Topology evolves at random ○ Topology reflects desired properties of linked nodes	○ Topology strictly determined by node IDs

Main Issues in P2P: Self organization

- Avoid central server: No one keeps full state: nodes take local decisions
- Distribute load on multiple peers
- Limit load per peer
- Let emerge global operation from local decisions
  - Self-Management
  - Self-Healing
  - Self-Configuration
  - Self-*

What I won’t talk about

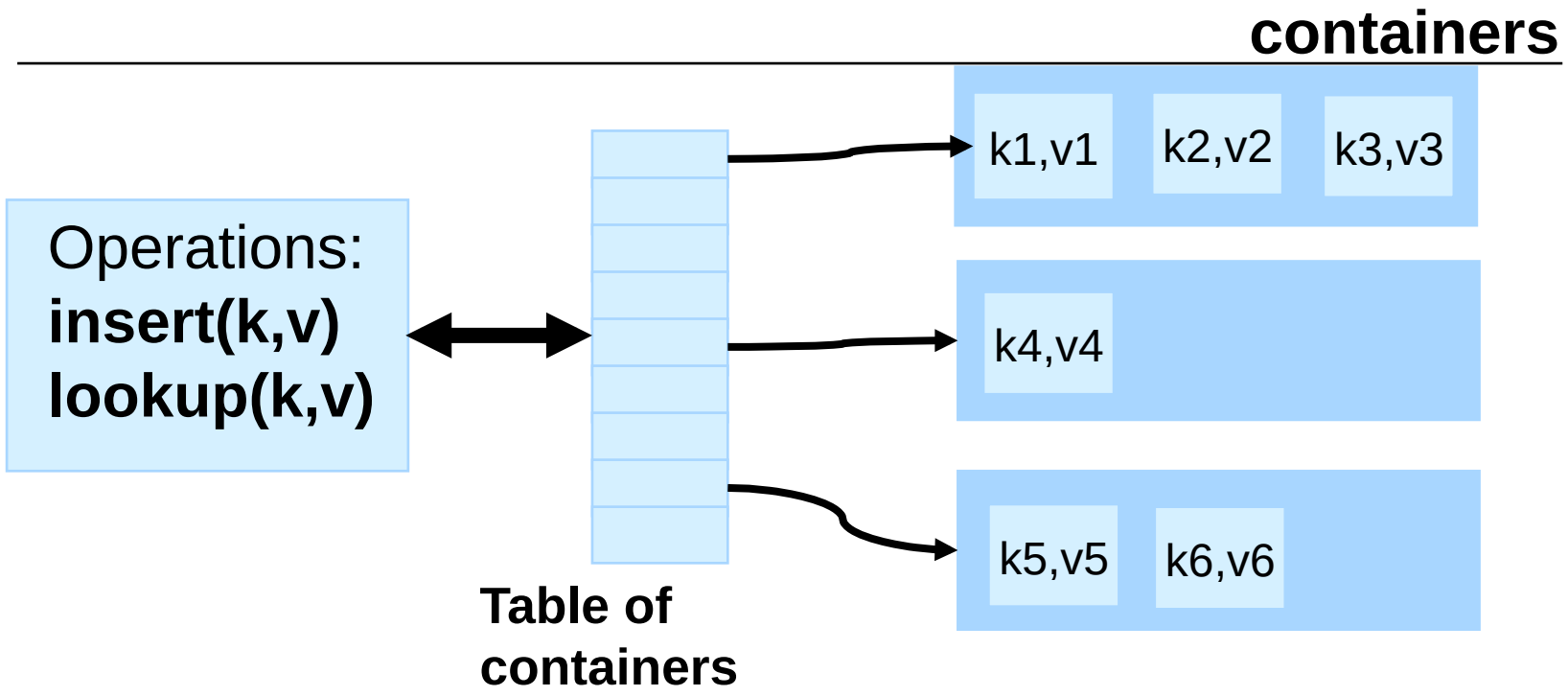
- Security
- Privacy
- Incentives
- Byzantine failures

Structured P2P overlay networks

P2P routing infrastructure

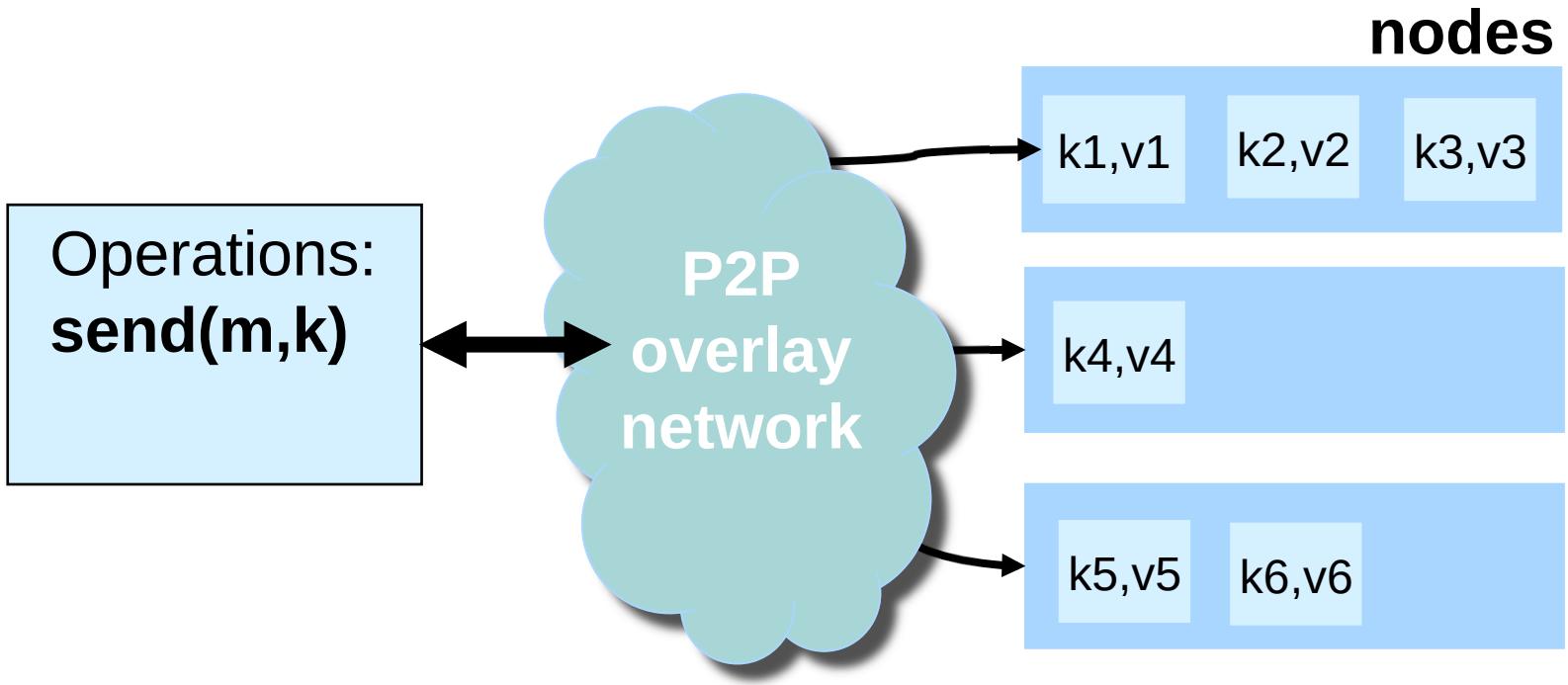
- Overlay: network abstraction on top of IP
- Basic functionality: distributed hash table
  - CAN relies on a multidimensional cartesian space
  - Chord, Pastry, Tapestry: generalized hypercube routing based on prefix matching
  - Exact-match interface
- Applications
  - Content-delivery networks
  - Storage systems, Caching
  - Naming services
  - Multicast

Distributed Hash Table (DHT)



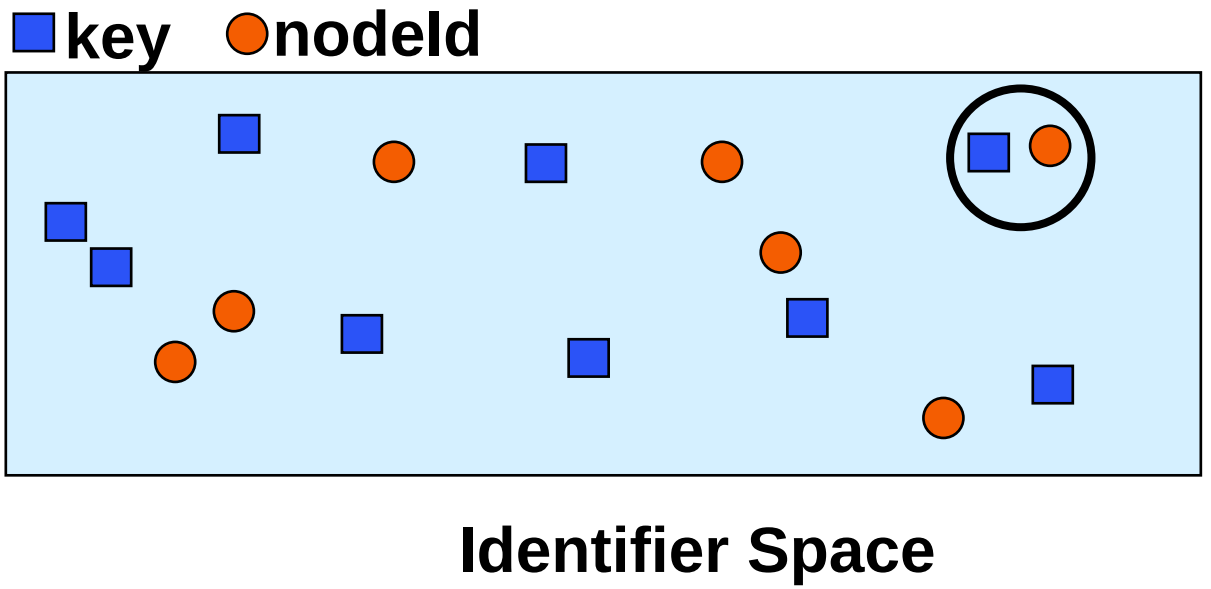
- Store <key,value> pairs
- Efficient access to a value given a key
- Mapping key-value ensured by the table of containers

Distributed Hash Table



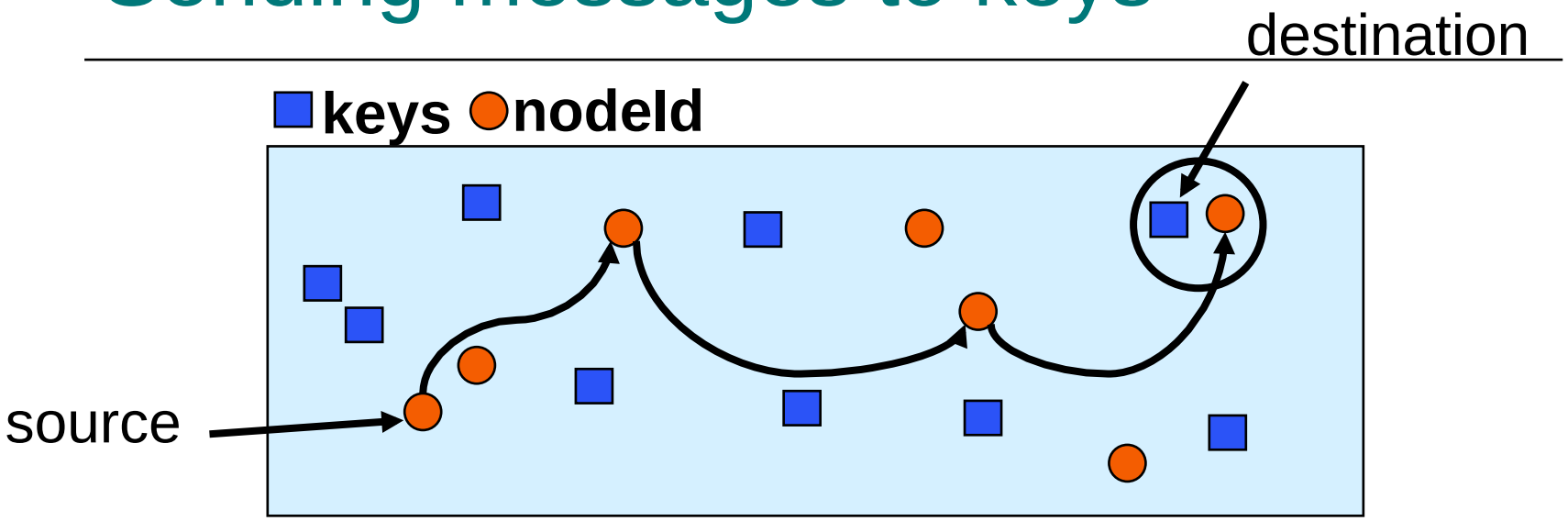
- Message sent to keys: implementation of a DHT
- P2P Infrastructure ensures mapping between keys and physical nodes
- Fully decentralized: peer to peer communication paradigm

Mapping



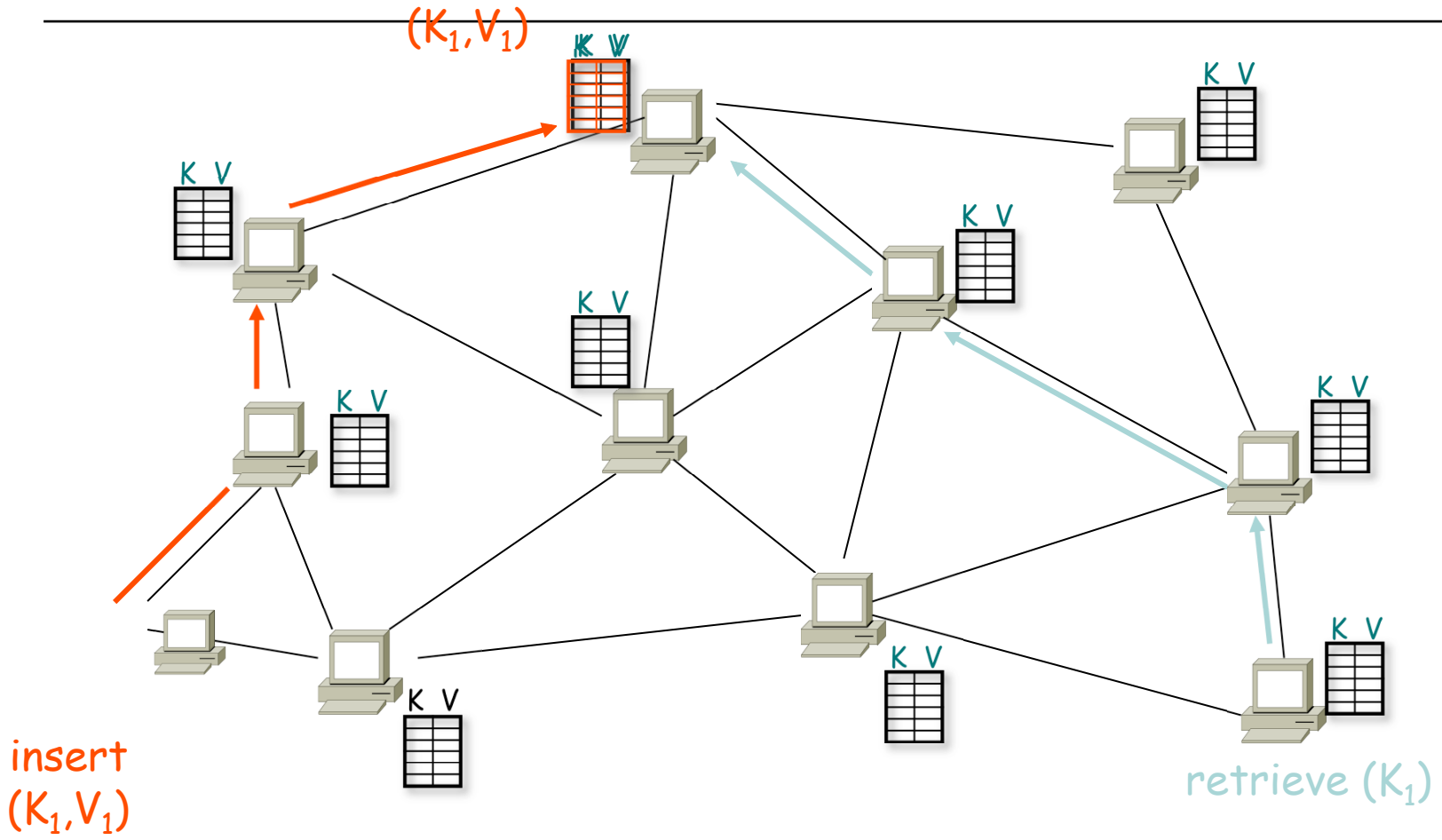
- An identifier is associated to each node (nodeId)
- Large identifier space (keys and nodeId)
- A node is responsible for closest key to its nodeId

Sending messages to keys

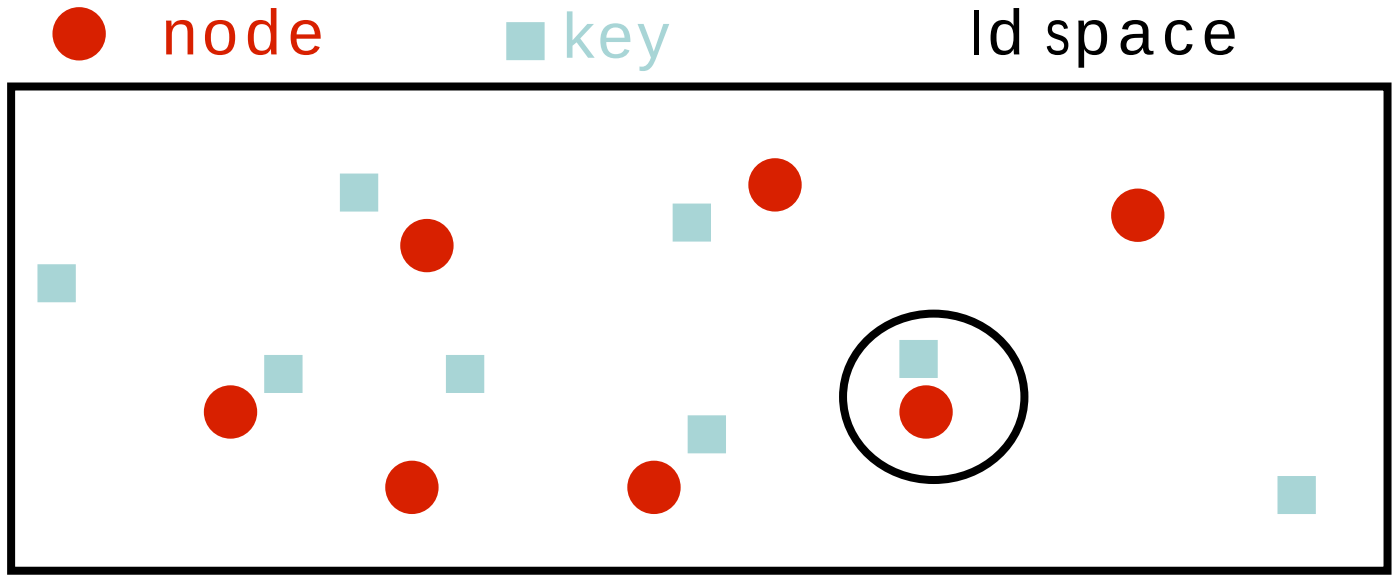


- Each node maintains a routing table (nodeid, IP adress)
- At each routing hop, the message gets closest to the key
- Infrastructure P2P: mapping between key and node

Distributed Hash Table



Pastry (MSR/RICE)

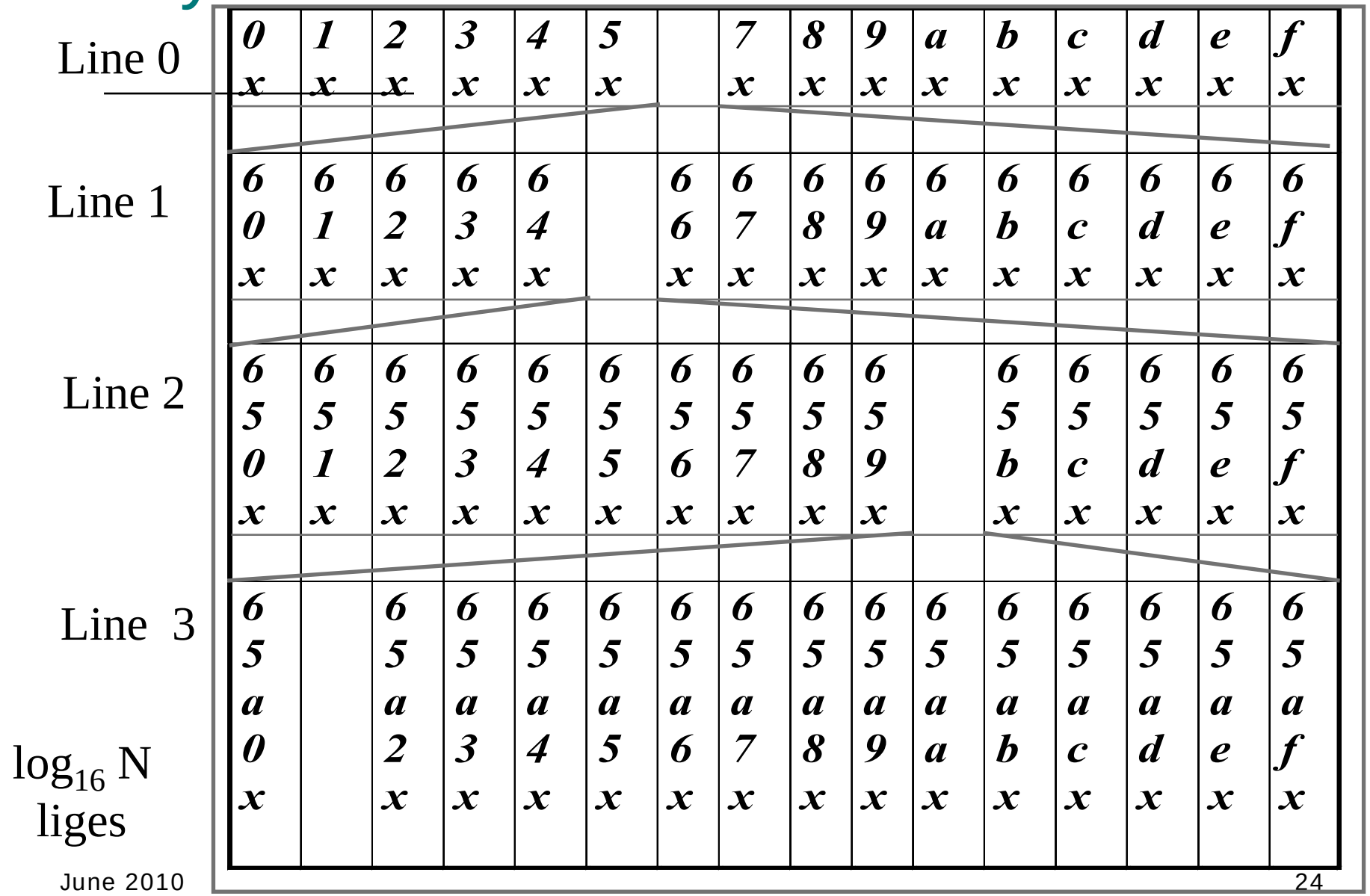


NodeId = 128 bits
Nodes and key place in a linear space (ring)
Mapping : a key is associated to the node with the numerically closest nodeId to the key

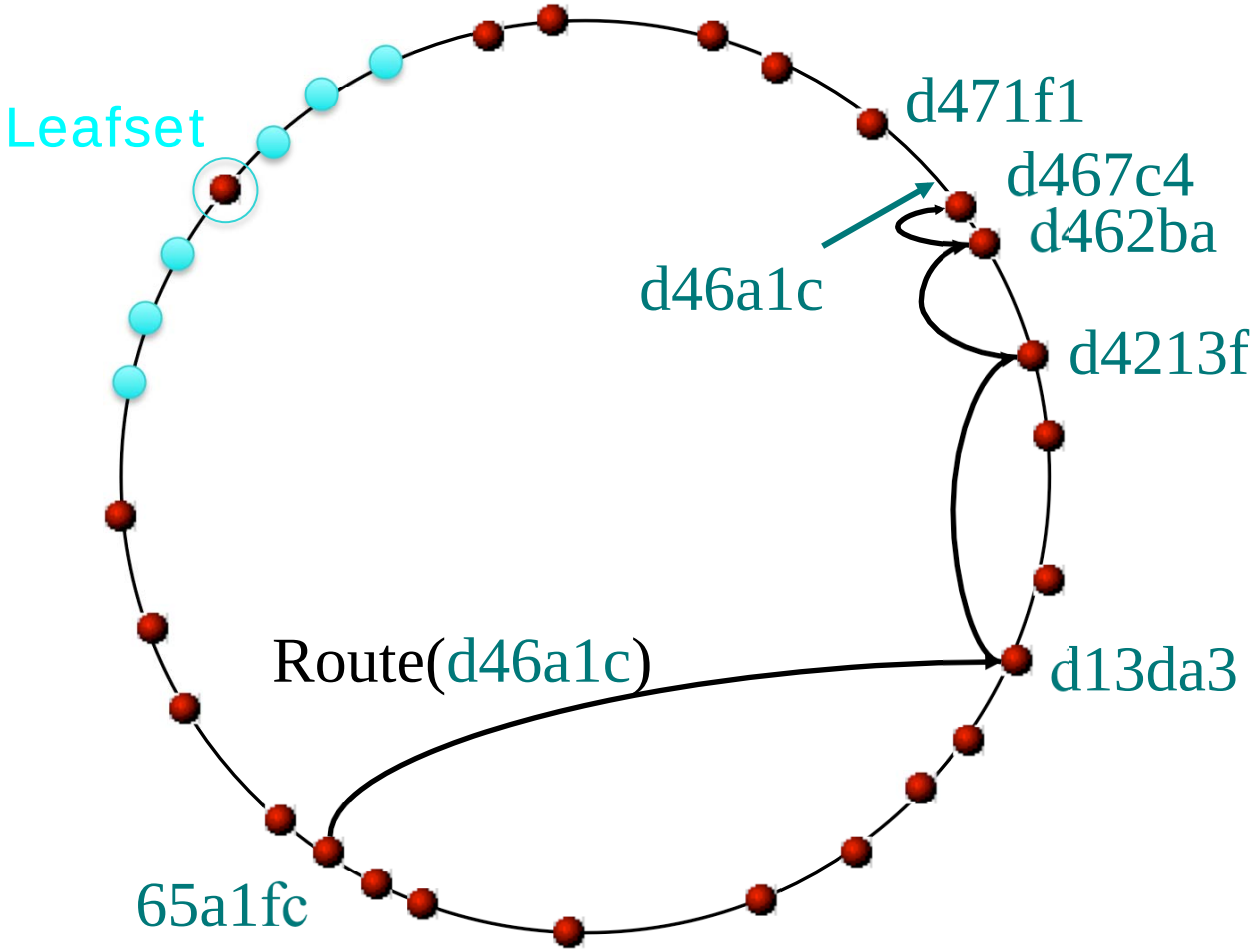
Pastry

- Naming space :
  - Ring of 128 bit integers
  - *nodeIds* chosen at random
- Key/node mapping
  - key associated to the node with the numerically closest node id
- Routing table:
  - Identifiers are a set of digits in base 16
  - Matrix of 128/4 lines et 16 columns
  - routeTable(i,j):
    - *nodeId* matching the current node identifier up to level I
    - with the next digit is j
- *Leaf set*
  - 8 or 16 closest numerical neighbours in the naming space

Pastry: Routing table(#65a1fcx)



Pastry: Routing



Properties

- $\log_{16}$ N hops
- Size of the state maintained (routing table): O(*log N*)
- Leaf set

Routing algorithm, notations

R_l^i : entry of the routing table R, $0 \le i \le 2^b$,

line l, $0 \le l \le \lfloor 128/b \rfloor$

L_i : ith closest nodeId in the leafset

D_l : value of the l digits of key D

$SHL(A,B)$: length of the shared prefix between A and B

Routing algorithm (on node A)

if $(L_{-\lfloor L/2 \rfloor} \le D \le L_{\lfloor L/2 \rfloor})$ {

// D belongs to the space covered by the leafset

forward to L_i, so that $|D\text{-}L_i|$ minimal }

else {

//use routing table entries

Let $l = SHL(D,A)$;

if $(R_l^{D_l} \neq null)$ forward to $R_l^{D_l}$;}

else {

// hole in the routing table

forward to $T \in \{L \cup R\}$ so that

$SHL(T,D) \ge l$

$|T\text{-}D| < |A\text{-}D|$ }

}

Node departure

- Explicit departure or failure
- Replacement of a node
- The leafset of the closest node in the leafset contains the closest new node, not yet in the leafset
- Update from the leafset information
- Update the application

Failure detection

- Detected when immediate neighbours in the name space (leafset) can no longer communicate
- Detected when a contact fails during the routing

    Routing uses an alternative route

Fixing the routing table of A

- Repair

R_l^d : entry of the routing table of A to repair

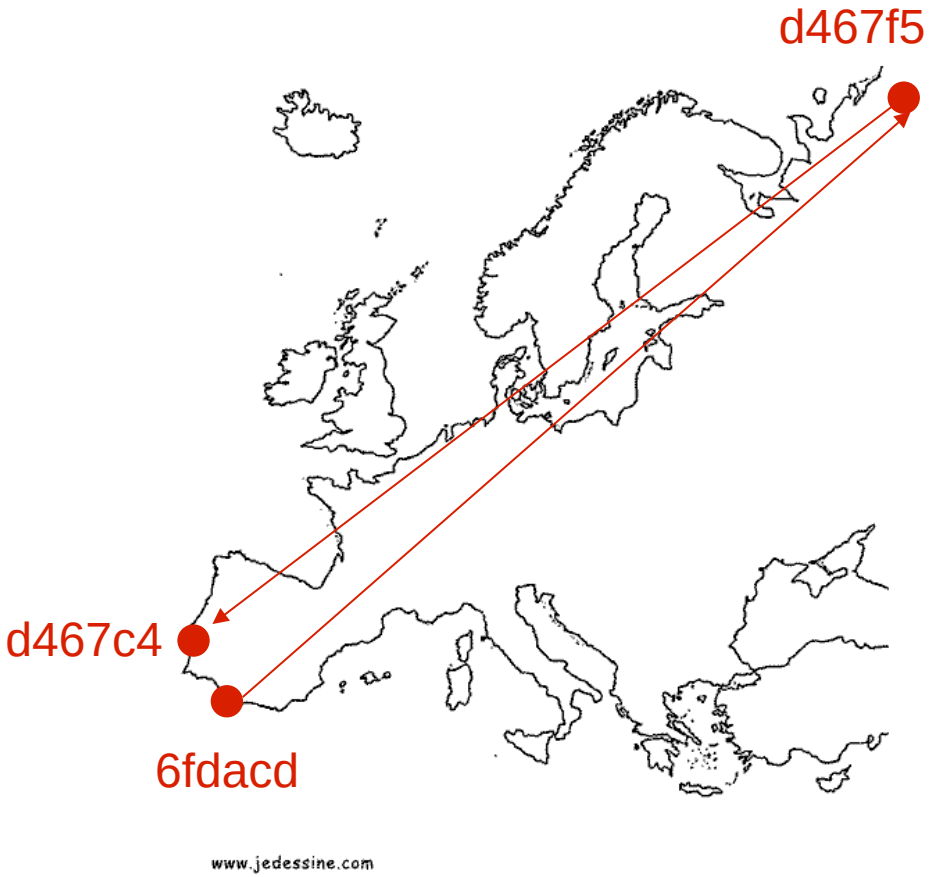
A contacts another entry (at random) R_l^i from the same line so that $(i \neq d)$ and asks for entry R_l^d, otherwise another entry from R_{l+1}^i $(i \neq d)$ if no node in line l answers the request.

State maintenance

- ***Leaf set***
  - **is aggressively monitored and fixed**
- **Routing table**
  - **is lazily repaired,** when a hole is detected during the routing
  - periodic gossip-based maintenance

Reducing latency

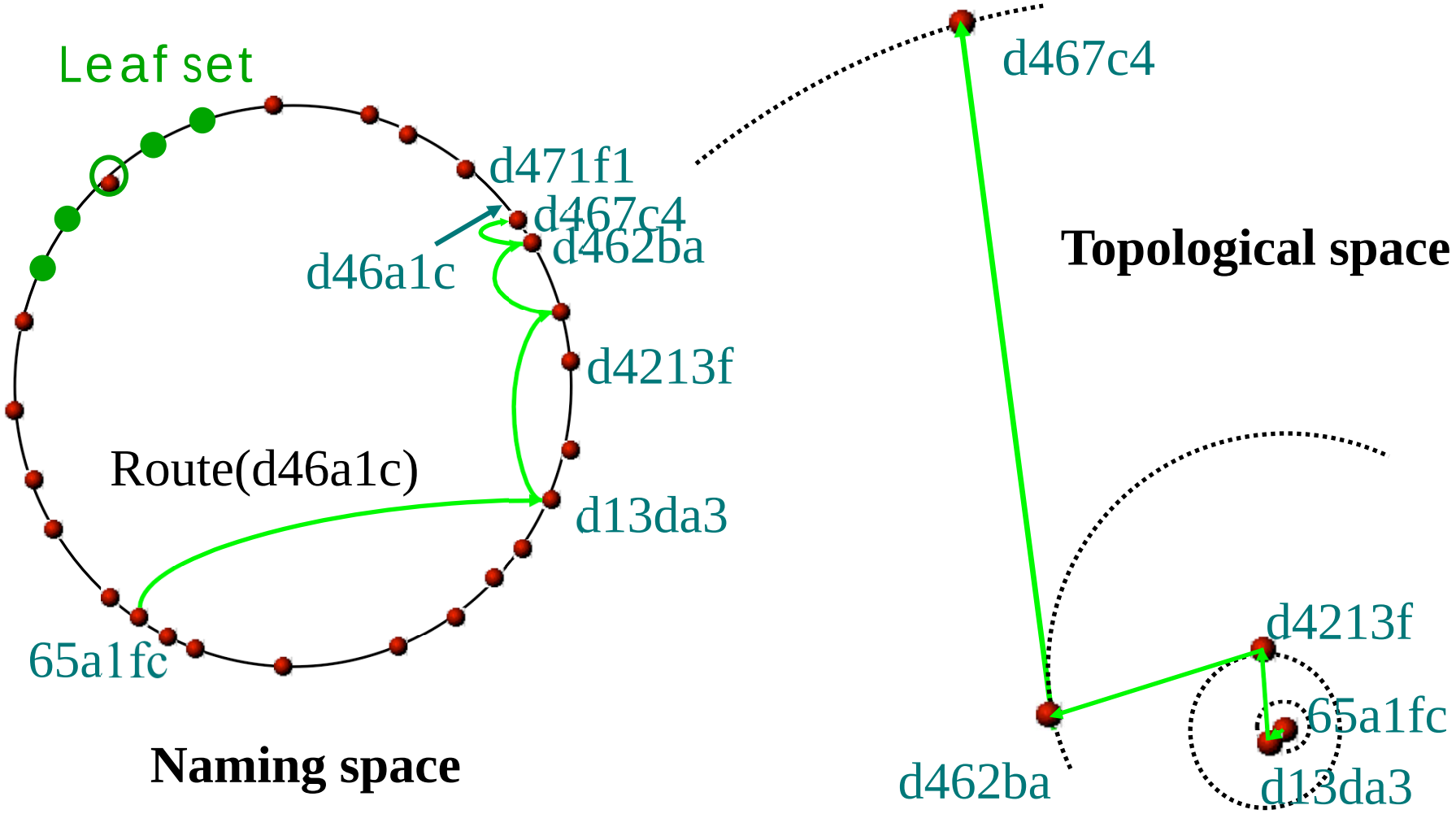
- **Random assignment of nodeId**: Nodes numerically close are geographically (topologically) distant
- **Objective**: fill the routing table with nodes so that routing hops are as short (latency wise) as possible
- **Topological Metric**: latency



Exploiting locality in Pastry

- Neighbour selected based of a network proximity metric:
  - Closest topological node
  - Satisfying the constraints of the routing table routeTable(i,j):
    - *nodeId* corresponding to the current *nodeId* courant up to level i
    - next digit = j
  - nodes are close at the top level of the routing table
  - random nodes at the bottom levels of the routing tables

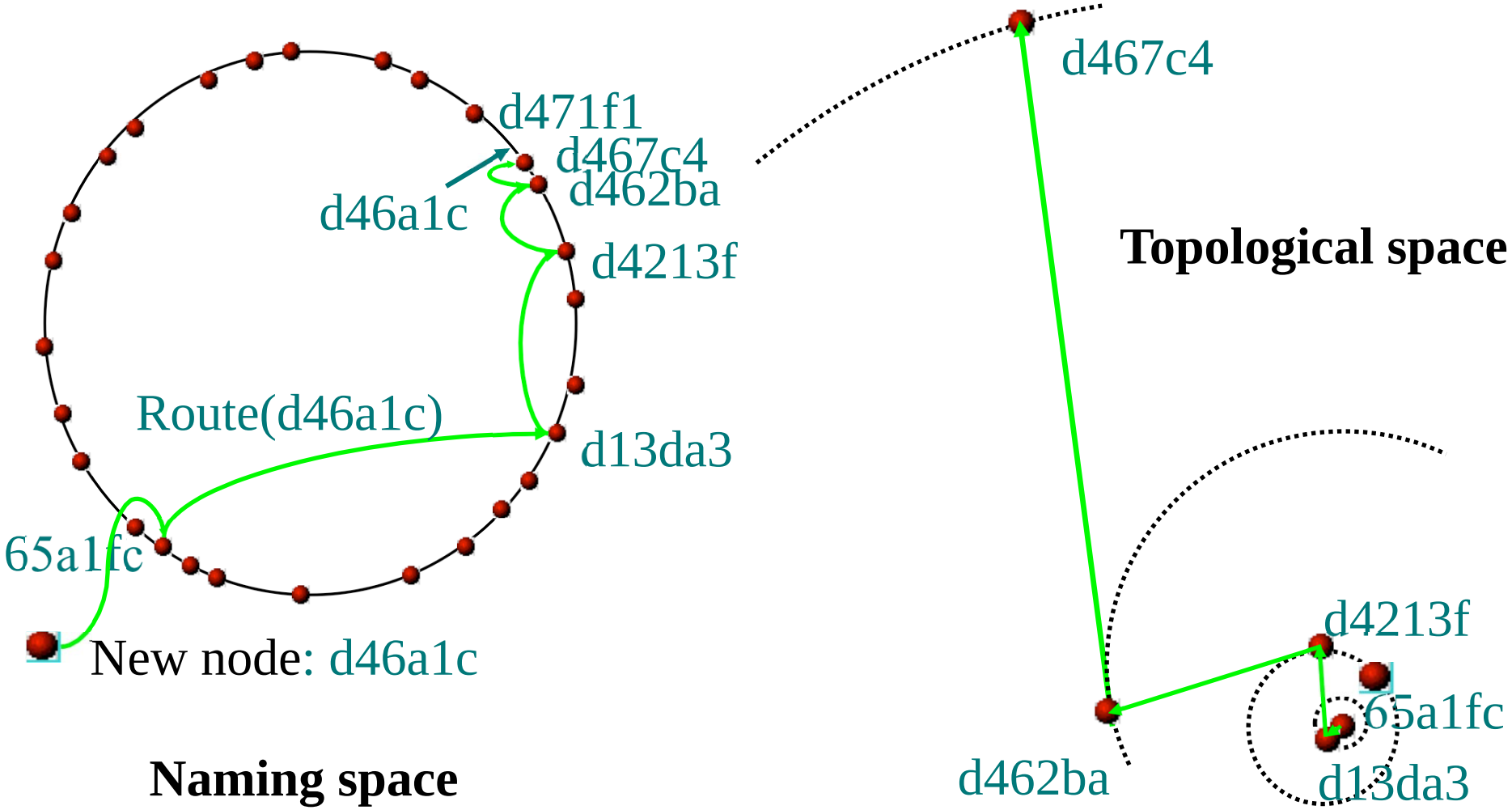
Proximity routing in Pastry



Locality

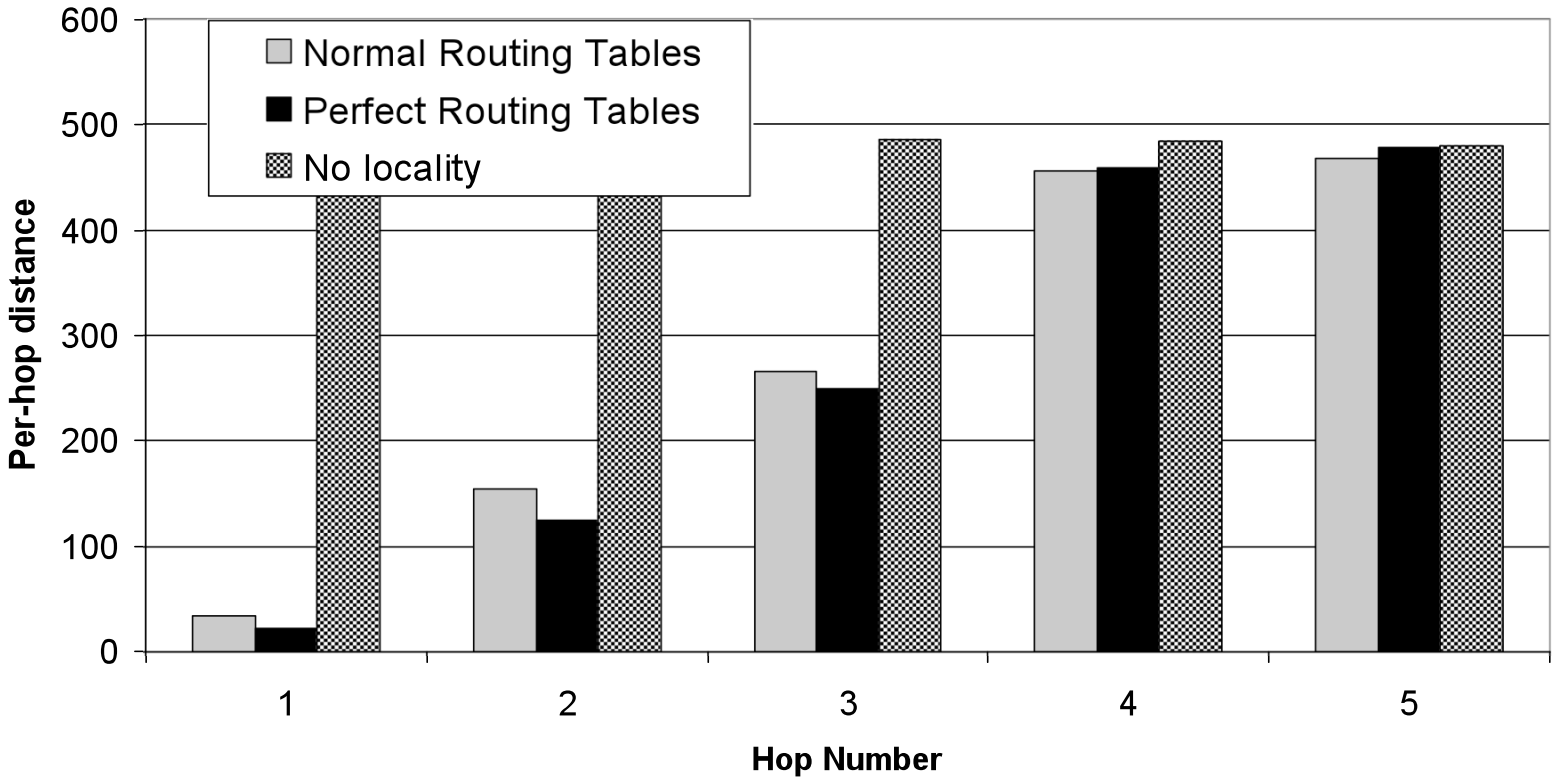
- Node X routes to node A
  - Path A,B,… -> Z
  - Z numerically closest to X
  - Initialisation of the line i of the routing table with the contents of line i of the routing table of the *ith* node encountered on the path
- Improving the quality of the routing table
  - X asks to each node of its routing table its own routing state and compare distances
  - Gossip-based update for each line (20mn)

Node insertion in Pastry



Performance

1.59 slower than IP on average

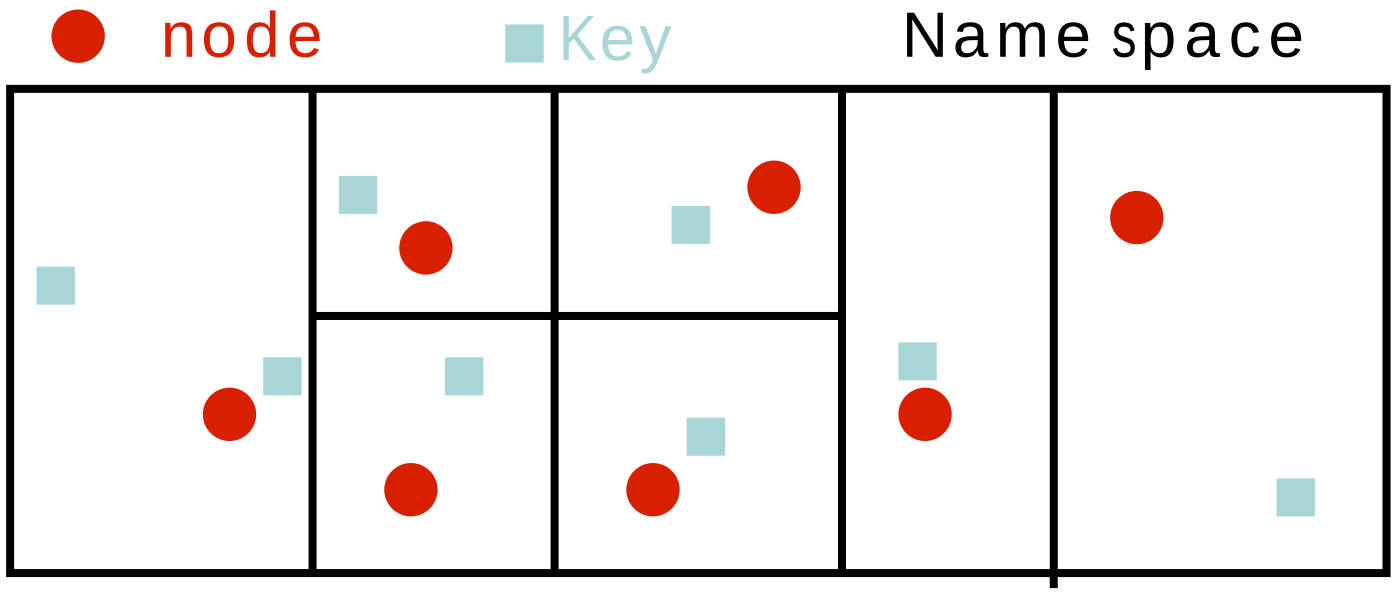


Content Adressable network (CAN)

Content Adressable Network -CAN

- UCB/ICIR
- Virtual coordinates Cartesian space
- The space is shared between peers
  - Each node is responsible for a part of the space (zone)
- Abstraction
  - CAN enables to store data at a given point in the space
  - CAN enable the routing from a point of the space to the other (DHT functionality)
  - A point is associated to the node owning the zone in which the point lies

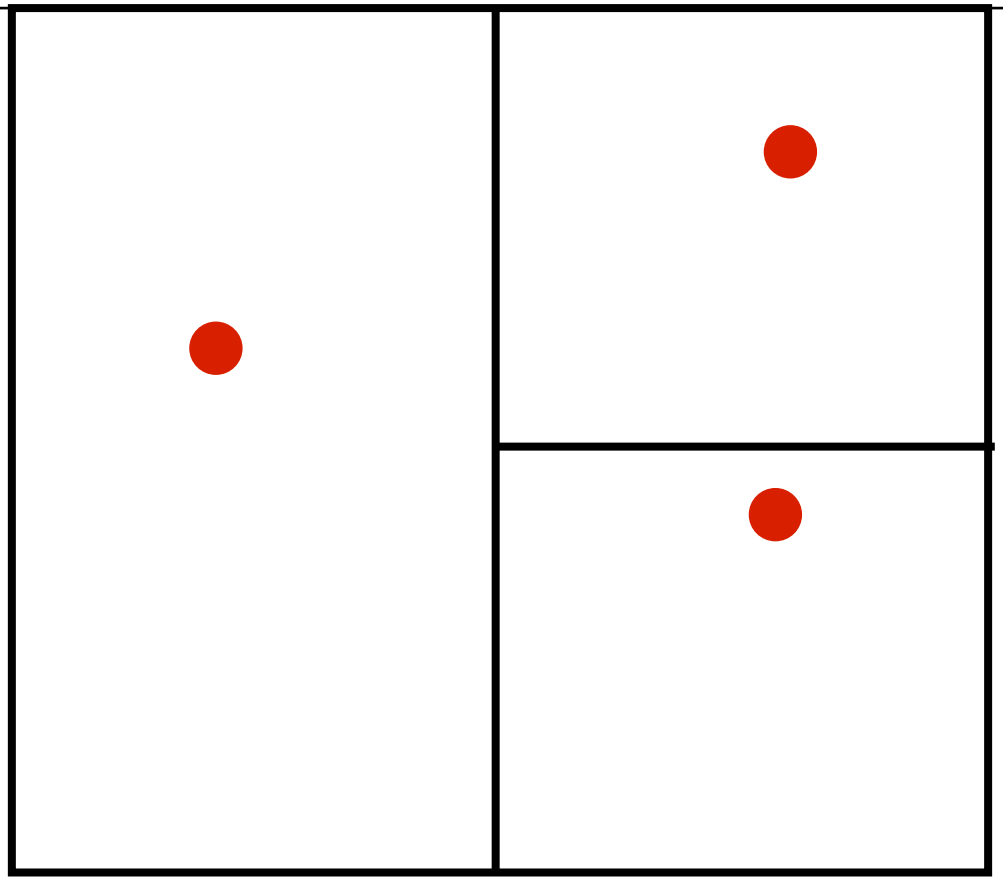
Space organisation in CAN



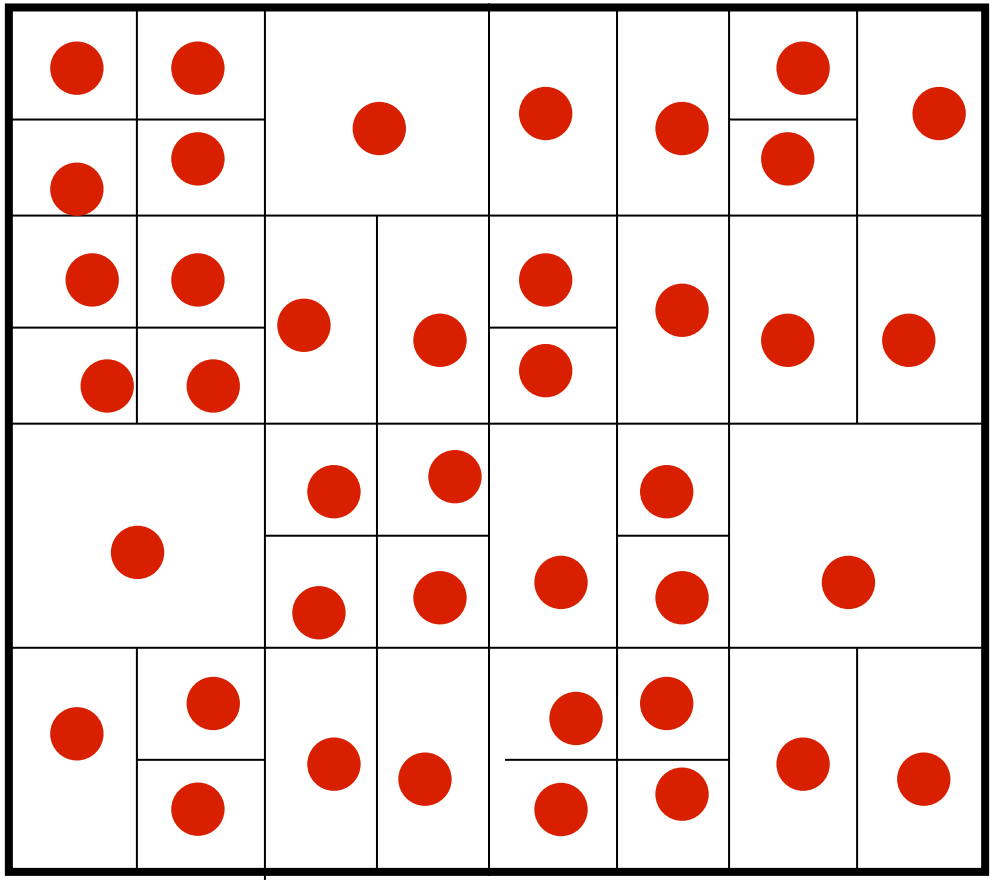
D-dimension space

Routing: progression within the space towards The destination

CAN: Example



CAN: exemple



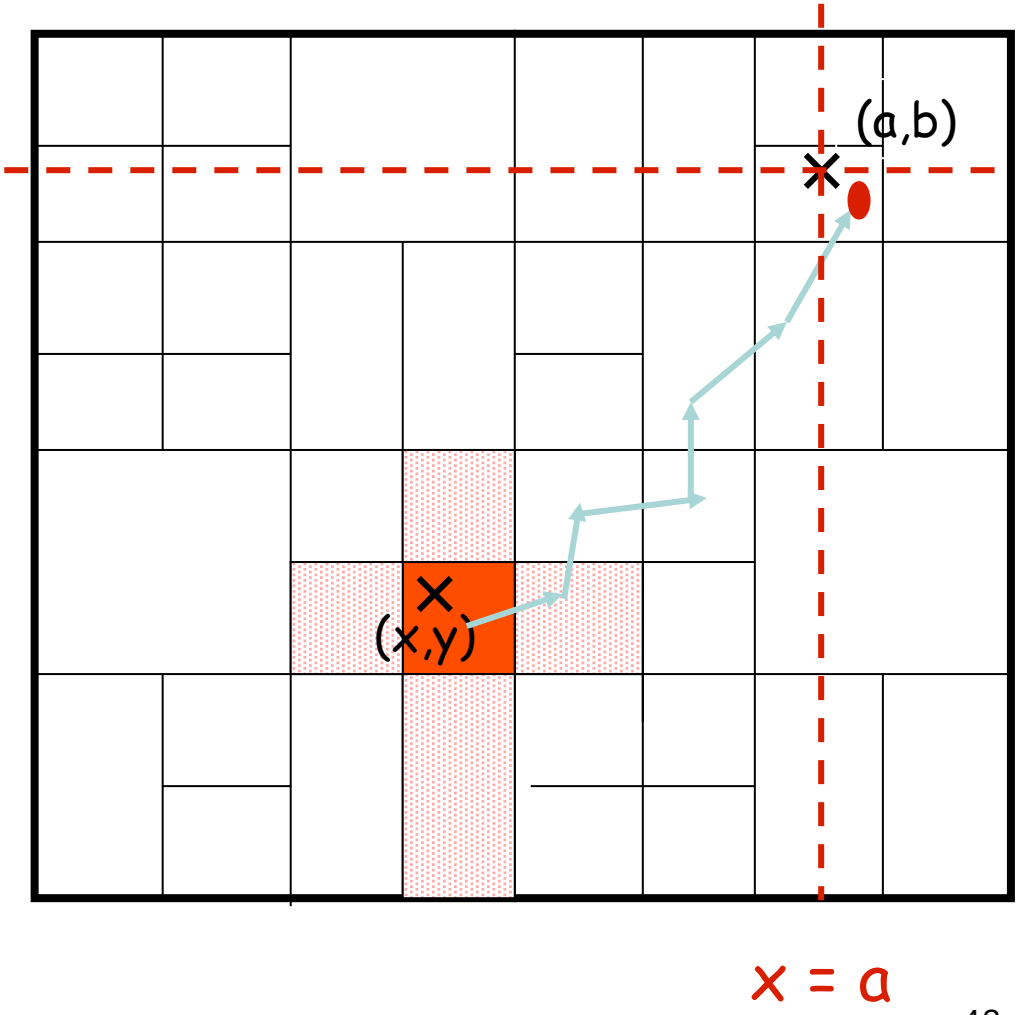
CAN: routing

node X ::insert(K,V)

(1) a = h_x(K)
b = h_y(K)

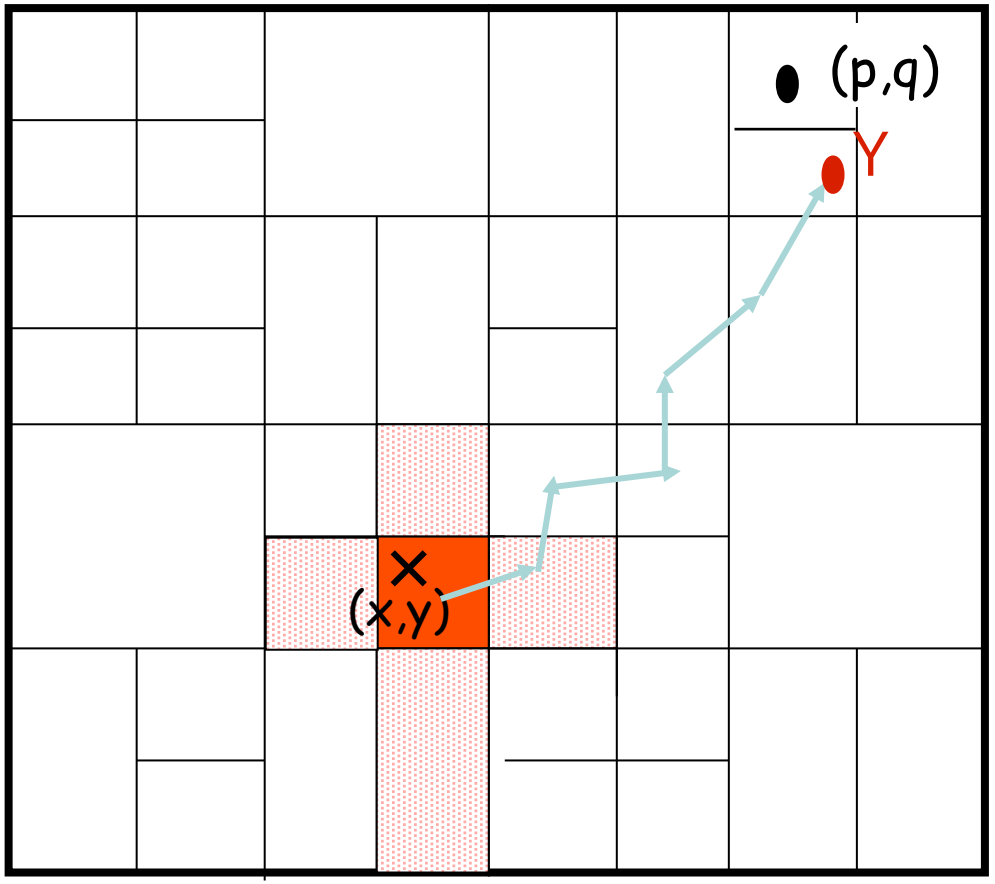
(2) route(K,V) -> (a,b)

(3) (a,b) stores (K,V)



CAN: node insertion

(1) Bootstrap : discovery of a contact node already participating to the CAN overlay network
(2) Selection of a random point (p,q) in the space
(3) Routing to (p,q) and discovery of node Y
(4) Zone splitting between Y and N



Insertion affects only Y and its immediate neighbours

Routing information

- The joining node gets the IP @ of its neighbours from the previous owner of the zone
- Set of neighbours of the joining node is a sub-set of neighbours of the previous owner
- The previous owner updates its own list of neighbours
- The neighbours of the joining node should also be updated

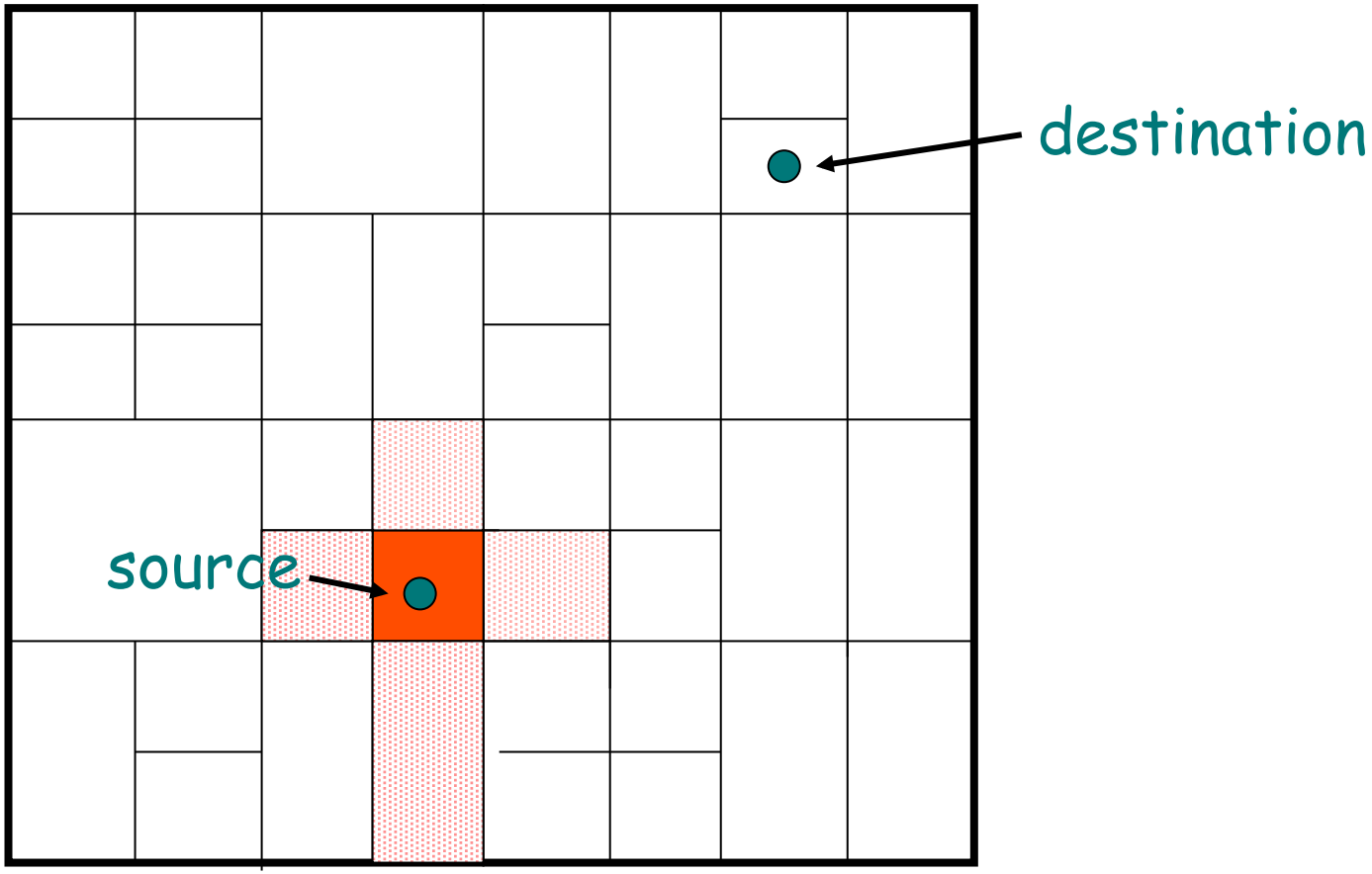
Departure, arrival, maintenance

- Node departure
  - The leaving node L must make sure that the zone is taken over
  - A leaving node hands over explicitly its own zone (and associated database) to one of its neighbours
  - Failure: detected by periodic messages from a node to its neighbours (hearteat) containing its own coordinates and coordinates of its neighbours

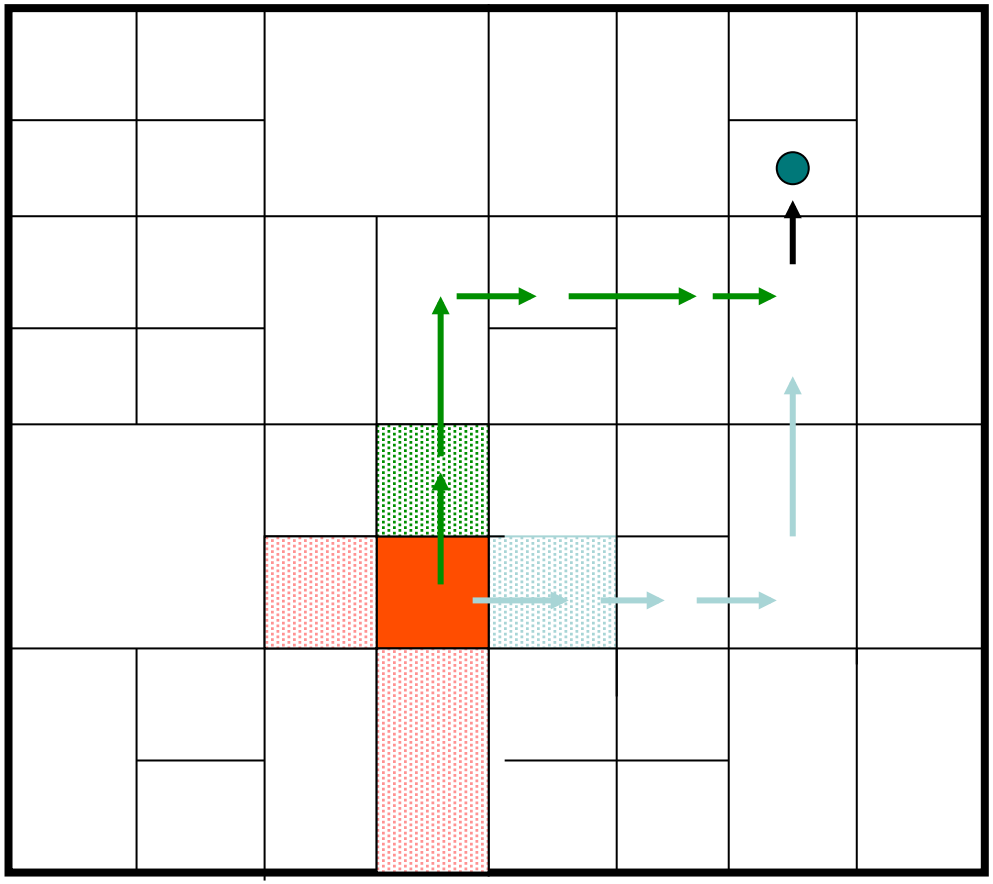
CAN: properties

- Each node maintains pointers to its immediate neighbours = 2d *O(d)*
- Routing in a N node network
  - Number of hops in a d-dimension space $O(d(n^{1/d}))$
  - In case of failure: selection of an alternative neighbour
- Optimizations
  - Multiple dimensions
  - Multiple reality
  - RTT Measures
  - Zone splitting
  - Locality awareness: *landmarks*

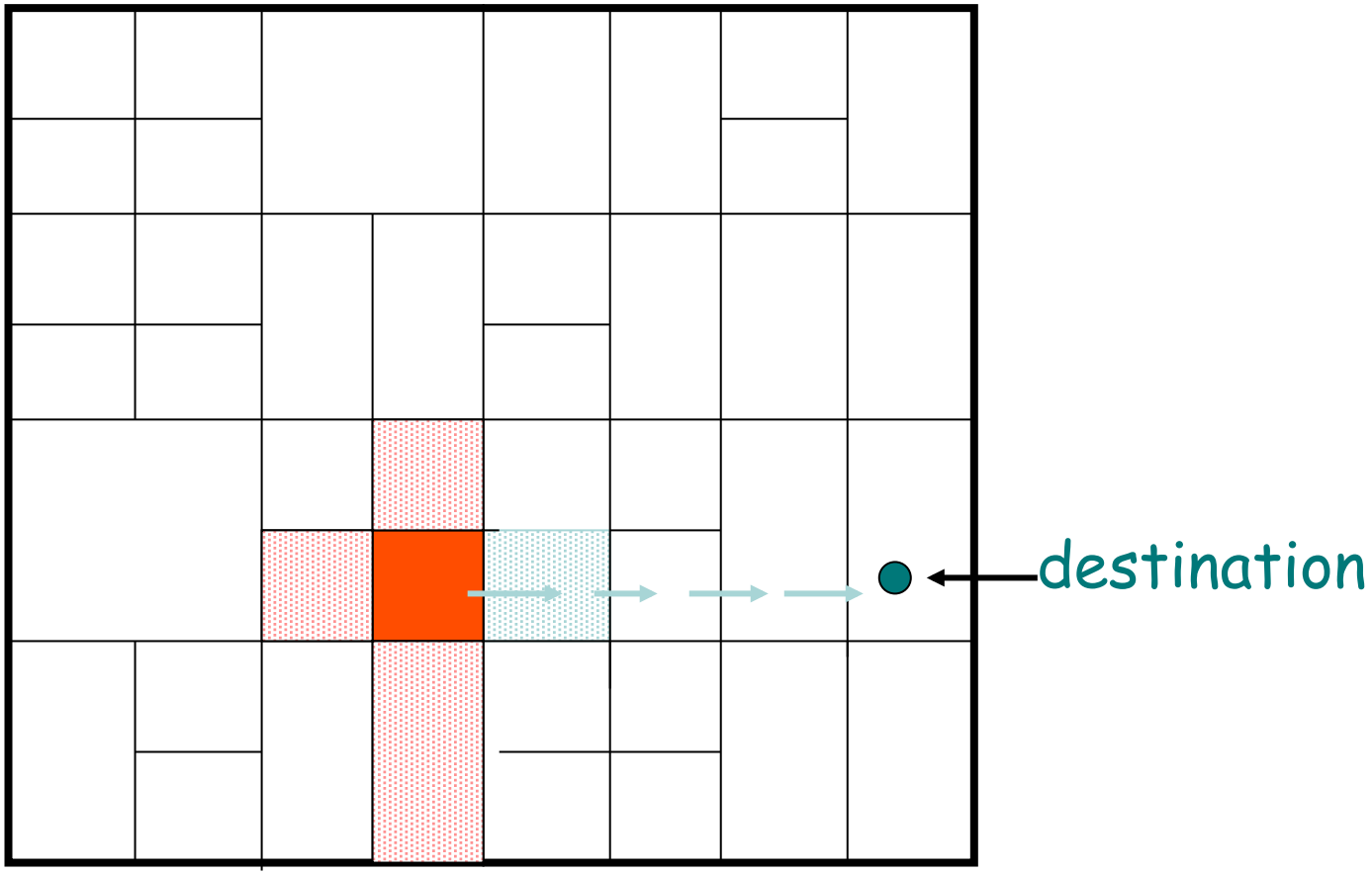
Failure resilience



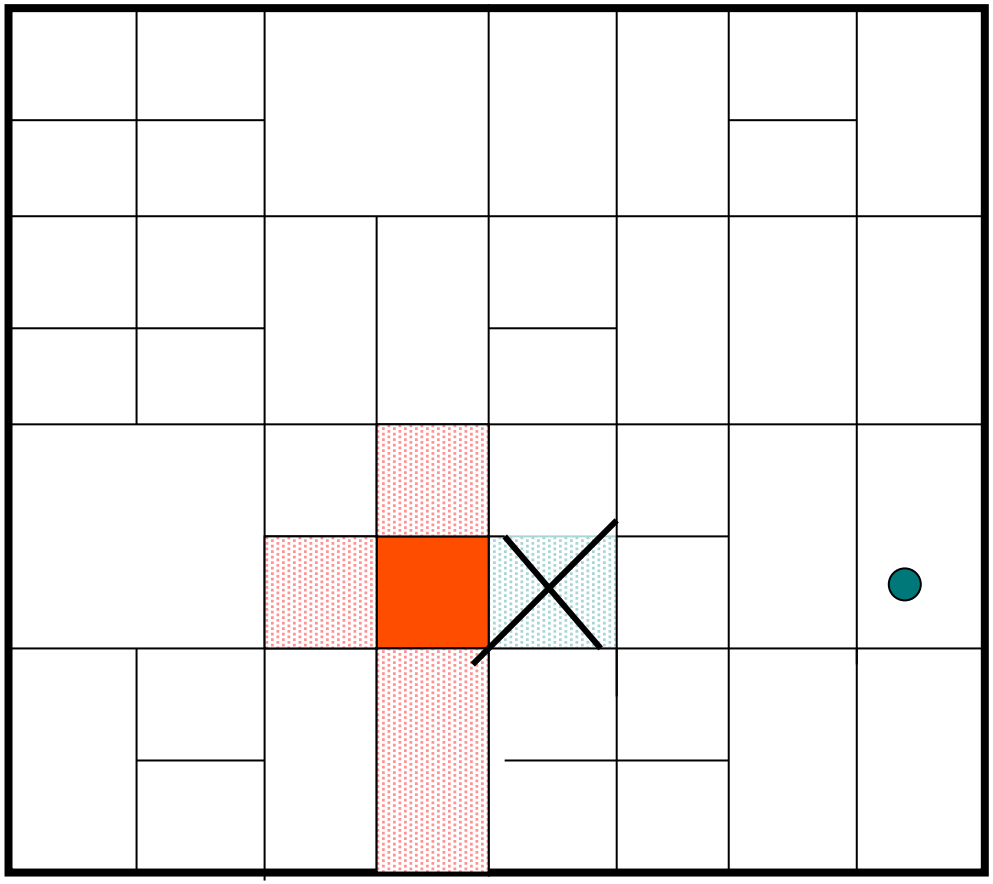
Failure resilience



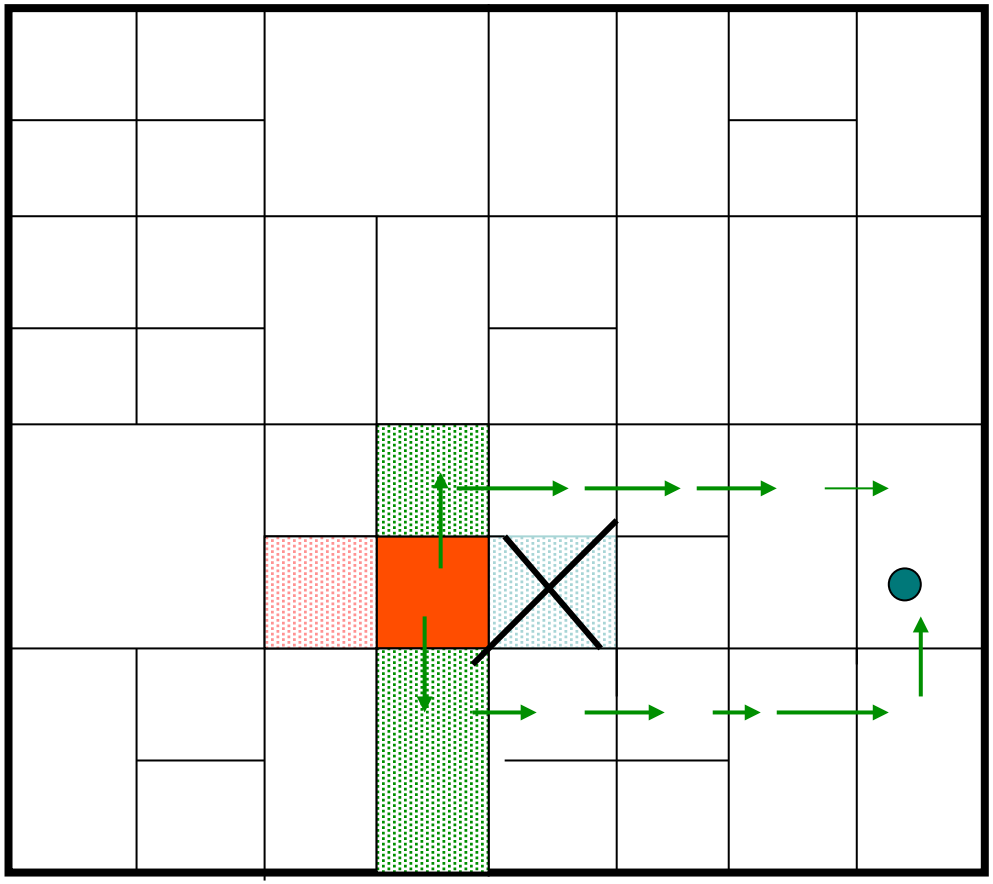
Failure resilience



Failure resilience



Routing resilience



Failure resilience

- Node X::route(D)

  If (X cannot progress directly towards D)
  - Check if one neighbour can progress towards the destination
  - If so, forward the message

Multiple dimensions

- Increasing the number of dimensions
  - The average path length is improved
  - The number of neighbours increases linearly with the dimension
  - Enhanced availability: potentially more nodes available

Reality

- Multiple independent coordinate spaces
- Each node is associated to a different zone in each space (reality): r sets of coordinates
- Enhanced availability
  - DHT content can be replicated across realities
    - Ex: a pointer to a file stored at (x,y,z) is stored on three nodes responsible of point (x,y,z) in 3 realities
  - Improves average path length as well: depending on the destination, the most relevant reality is chosen

RTT measures

- So far, the metric used to progress in the space in the path length in the Cartesian space
- Better criterion to take into account the underlying topology
- RTT to each neighbour
- Message forwarded to the neighbour for which the ratio progress/RTT is the best
  - Avoid long hops

Summary on structured overlay networks

- Chord, Pastry and Tapestry use a generalized hypercube routing: prefix matching
  - State maintained: O(Log(N))
  - Number of routing hops: O(Log(N))
  - Proximity routing in Pastry and Tapestry

- CAN uses progression in a multidimensional Cartesian space
  - State maintained: O(D)
  - Number of routing hops: $O(N^{1/D})$
  - Proximity routing more difficult to exploit

DHT Functionality=Exact match interface

References

- A. Rowstron and P. Druschel, "**Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems**", *Middleware'2001*, Germany, November 2001.
- **Scalable Content-Addressable Network (SIGCOMM 2001)** Sylvia Ratnasamy Paul Francis Mark Handley Richard Karp Scott Shenker

- Many more: Google P2P structured overlay networks

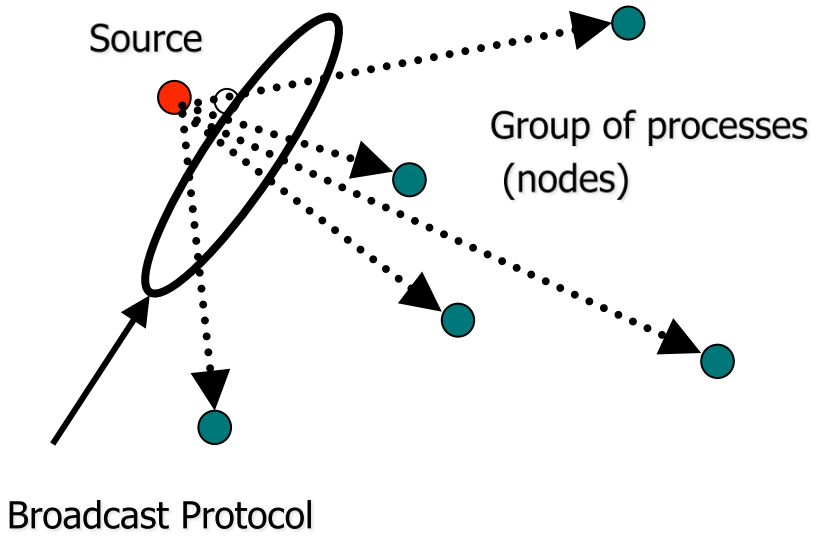
Application level multicast

Group Communication

- Common and useful communication paradigm
- Disseminating information within a group sharing interest
  - Consistency of replicated data
  - Publish/Subscribe systems
- Studied a lot in local area networks
  - Group management (join, leave, send)
- More scalability needed
  - Application-level multicast (for medium-size groups) not scalable
  - Network-level multicast not fully deployed

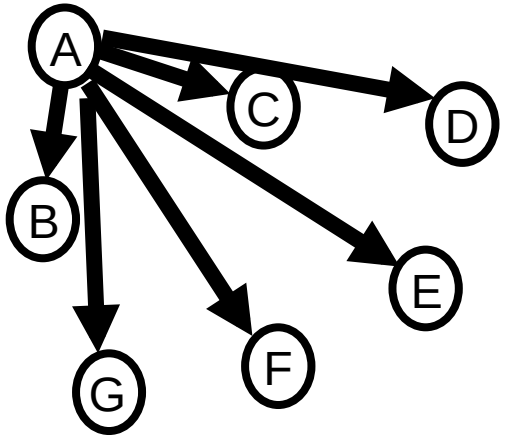
Group communication

- Important functionality of distributed systems
  - Failure detection
  - Membership management
  - Coherence management
  - Event notification systems
- **Scalability**
  - Group size
  - Geographical spreading



Broadcast protocols

- Centralized versus decentralized protocols
  - Load balancing
  - Performance
- Evaluation metrics
  - Delay from the source to the destination
  - Network traffic
  - Node load
  - Failure resilience



Application-level multicast (ALM)

1. Structured peer to peer networks
   - Flooding
   - Tree-based
2. Large Content dissemination

Structured overlay networks

- Scalability
  - O(logN) hops routing with a O(logN) state
  - Load balancing
- Self-* properties (organizing, healing, ...)
  - P2P overlay network automatically repaired upon peer joins and departures
  - Automatic load re-distribution

Attractive support for large-scale application-level multicast

ALM on structured overlay networks

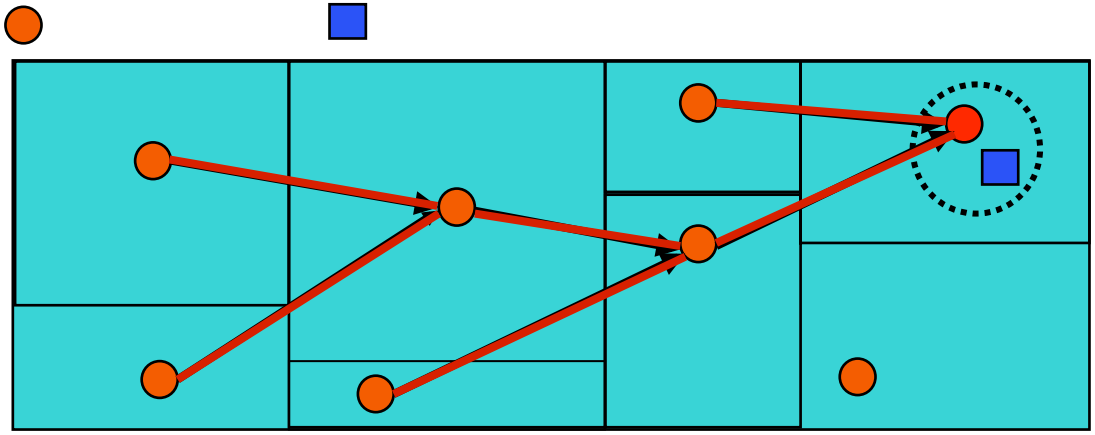
- Overlay network used for group naming and group localization
- Flooding-based multicast [CAN multicast]:
  - Creation of a specific network for each group
  - Message flooded along the overlay links
- Tree-based multicast [Bayeux, Scribe]
  - Creation of a tree per group
  - Flooding along the tree branches

Flooding-based multicast

- Group members join the network associated to a given group
- Messages sent over all links of the P2P overlay
- Specific mechanism to get rid of duplications
- Example : message *m* in Pastry
  - on receiving <*flood*, *m*, *i*>
  - for each routing table row *i'* (*i'* greater than *i*)
    - send <*flood*, *m*, *i'*> to nodes in row
  - *i=0* for original message sender

Tree-based multicast

- Creation of a tree per group
  - The tree root is the peer hosting the key associated to that group
  - The tree is formed as the union of routes from every member to the root

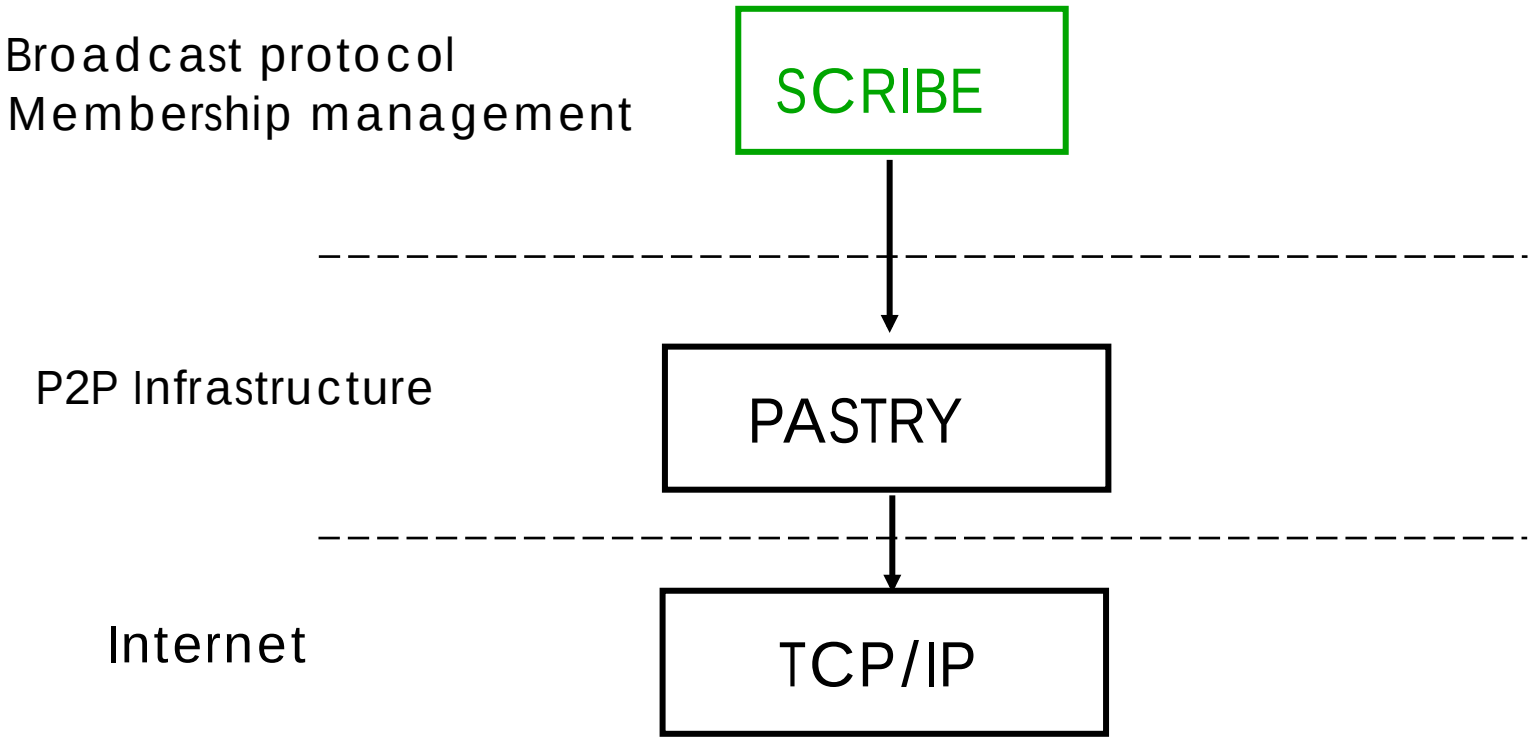


id space

Scribe

- Support multiple groups on a p2p prefix-matching infrastructure (Pastry, Tapestry,...)
- Support various applications (size-wise) on a single infrastructure potentially
  - Instant Messaging
  - Information dissemination (stock alerts)
  - Diffusion lists (Windows updates)
- Properties
  - Scalability
  - Efficient: low latency, low network link stress, low node load
  - Reliability: application-specific

Scribe


Broadcast protocol
Membership management
SCRIBE
P2P Infrastructure
PASTRY
Internet
TCP/IP

Scribe: design

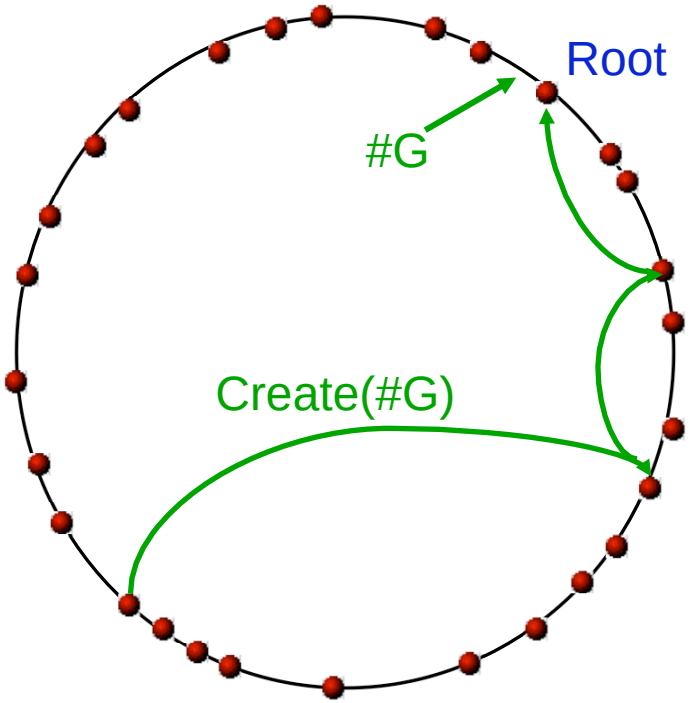
- Goals
  - Group creation
  - Membership maintenance
  - Messages dissemination within a group
- Construction of a multicast tree on top of a Pastry-like infrastructure
  - *Reverse path forwarding*
  - Messages flooded along the tree branches

Scribe Interface

- Create(group)
- Join(group)
- Leave(group)
- Multicast(group,m)

The P2P infrastructure is used for group creation and join protocol

Scribe: group creation

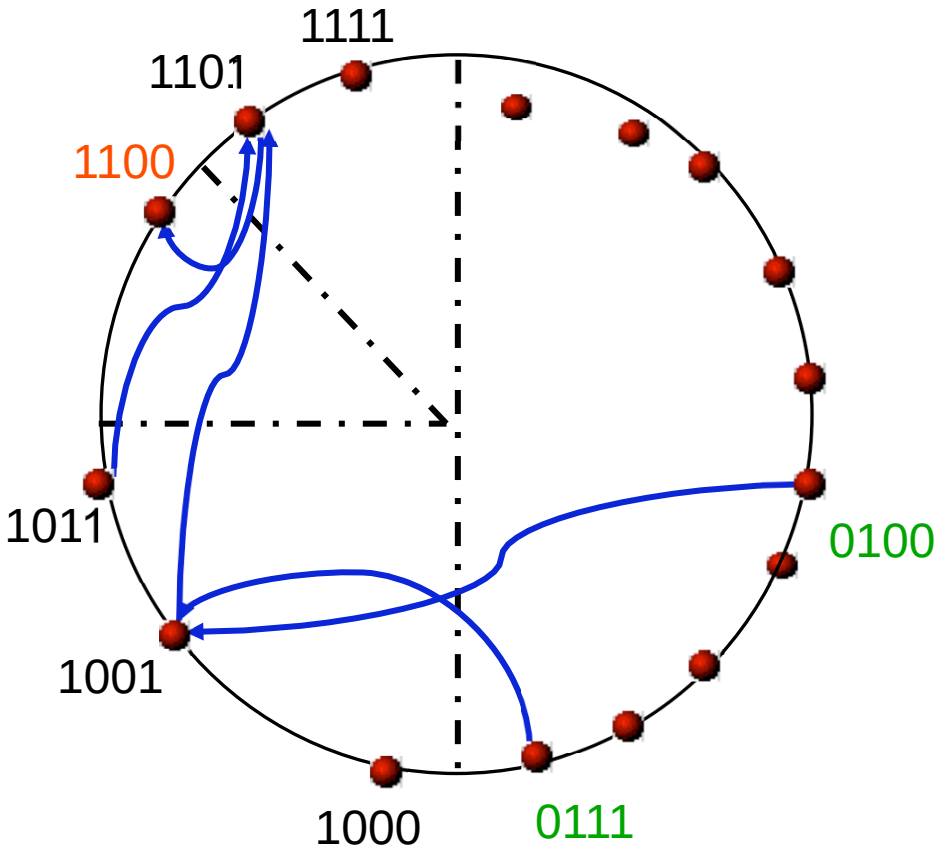


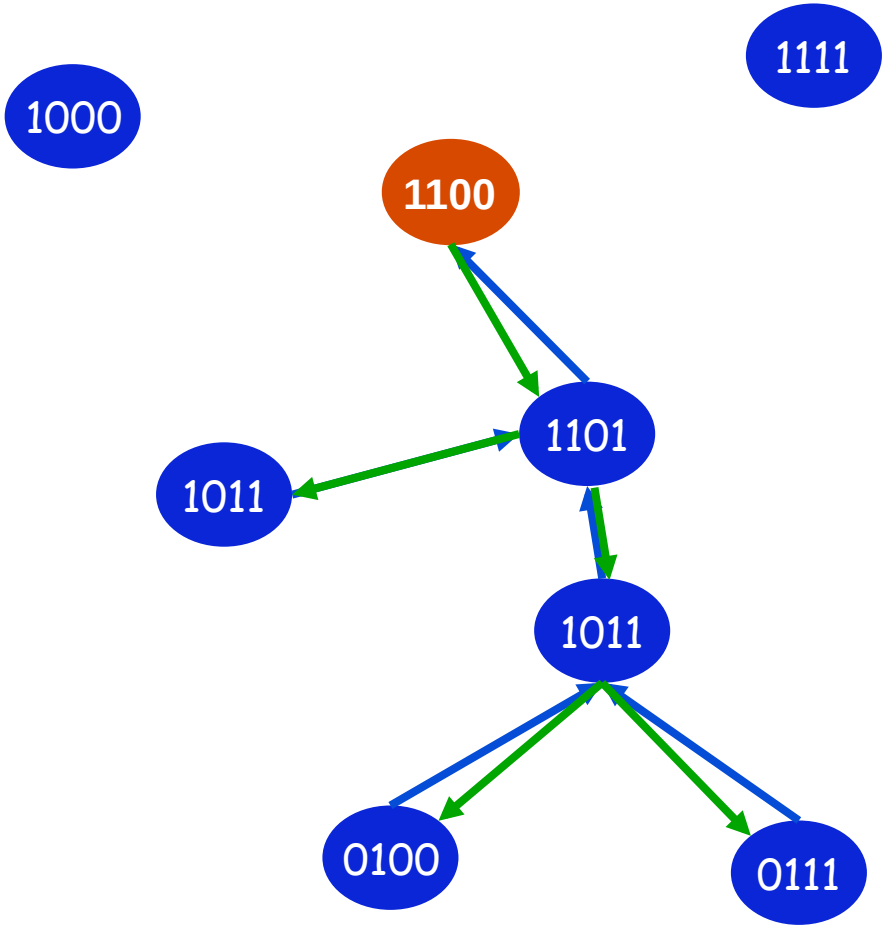
- Each group is assigned an identifier *groupId = Hash(name)*
- Multicast tree root : node which nodeId is the numerically closest to the groupId
- **Create(group):** P2P routing using the *groupeId* as the key

Scribe: tree creation

- **join(group)** : message sent through Pastry using *groupeId* as the key
- **Multicast tree** : union of Pastry routes from the root to each group
  - Low latency: leverage Pastry proximity routing
  - Low network link stress: most packets are replicated low in the tree

Scribe : join(group)

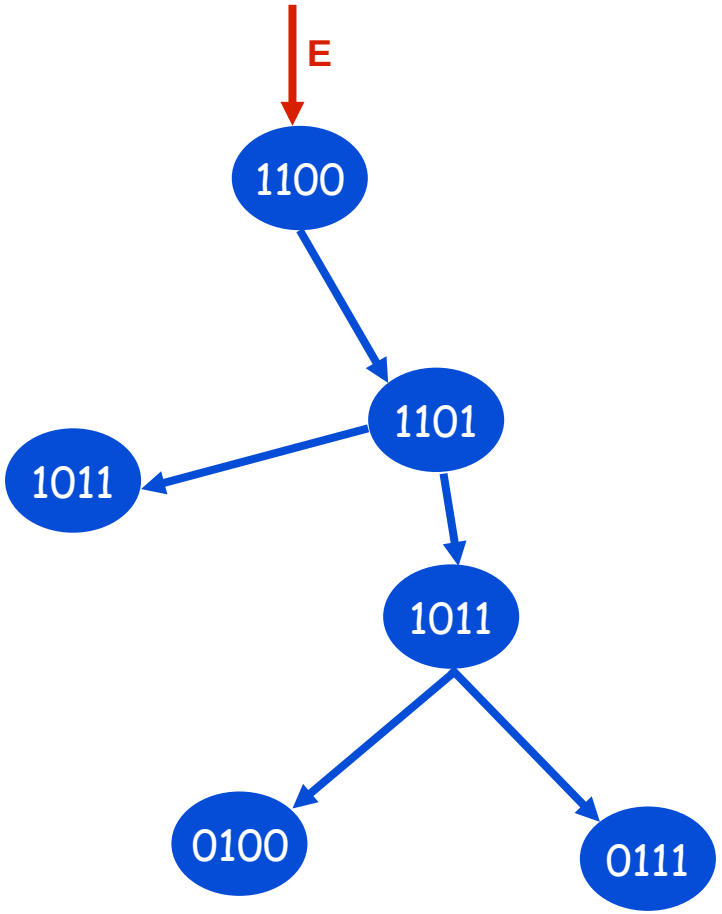




Scribe: message dissemination

Multicast(group, m)

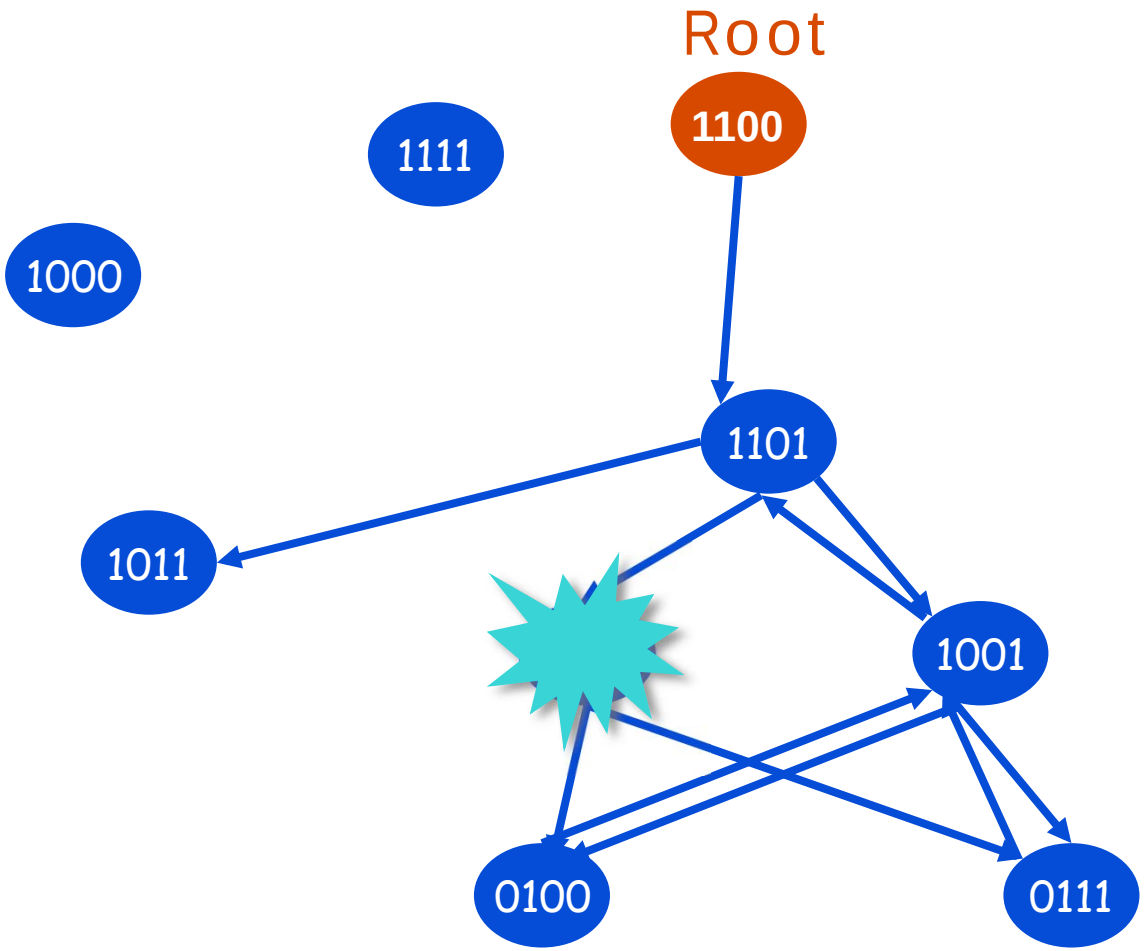
- Routing through Pastry to the root key=*groupeId*
- Flooding along the tree branches from the root to the leaves



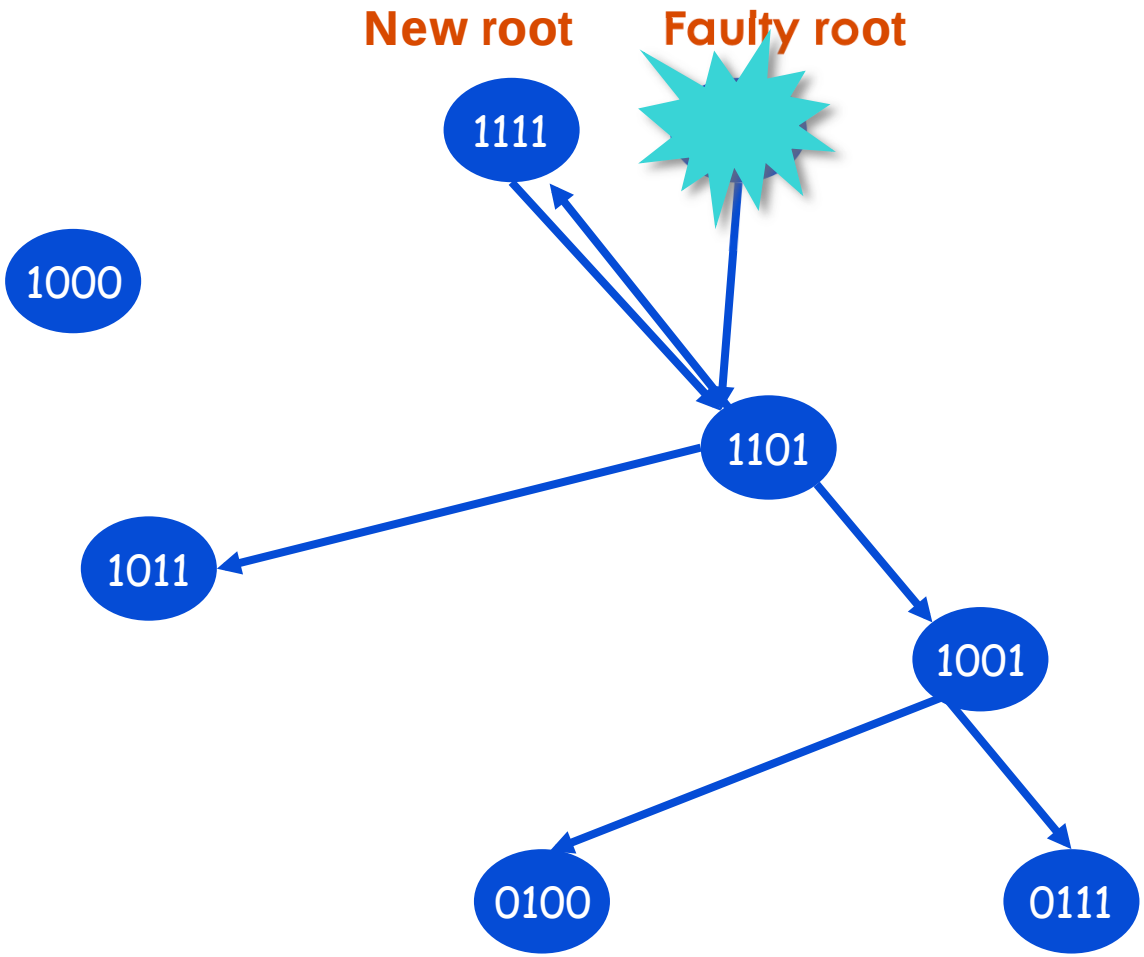
Reliability

- « best effort » reliability guarantee
  - Tree maintenance when failures are detected
  - Stronger guarantee may also be implemented
- Node failure
  - Parents periodically send heartbeat messages to their descendants in the tree
  - When such messages are missed, nodes join the group again
- Local reconfiguration
- Pastry routes around failures

Tree maintenance



Tree maintenance



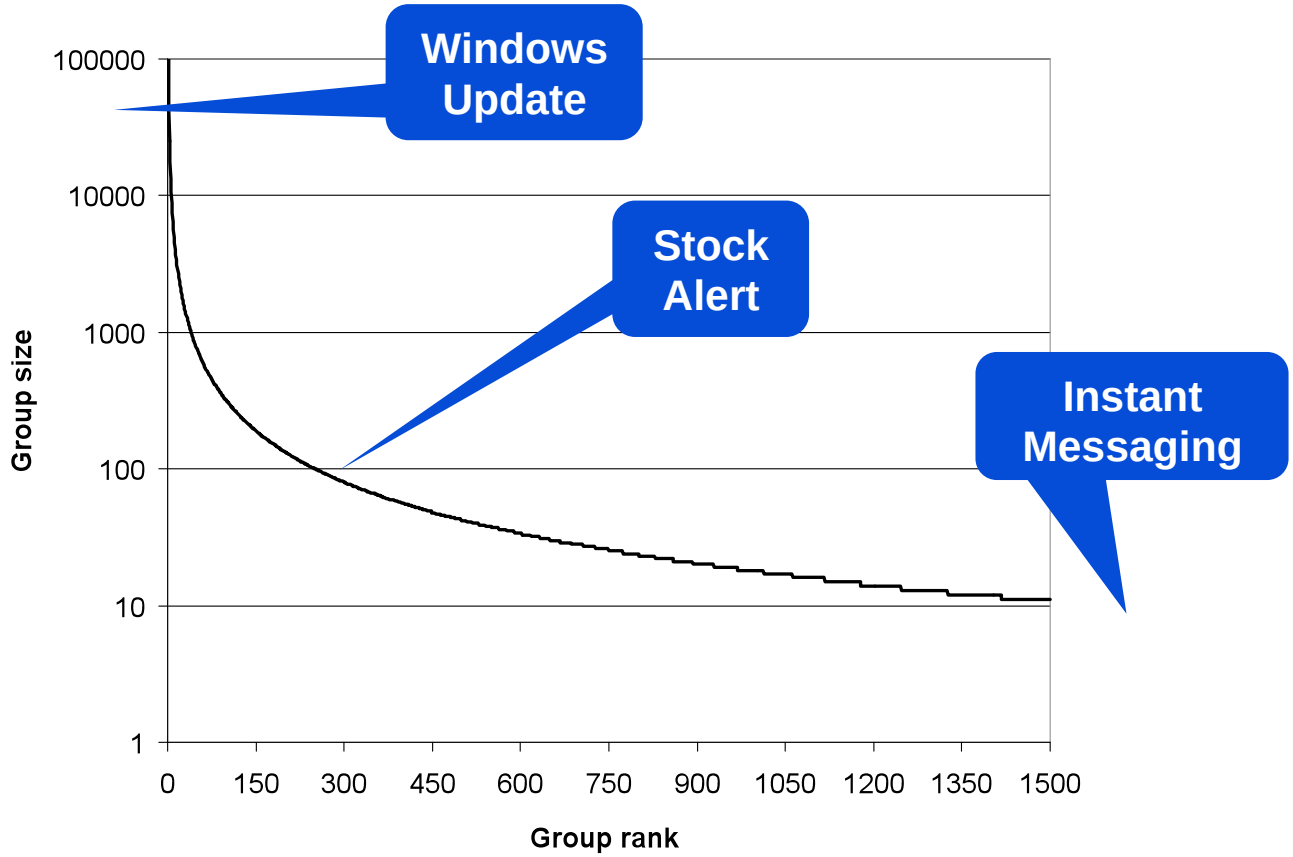
Load balancing

- Specific algorithm to limit the load on each node
  - Size of forwarding tables
- Specific algorithm to remove the forwarders-only peers from the tree
  - small-size groups

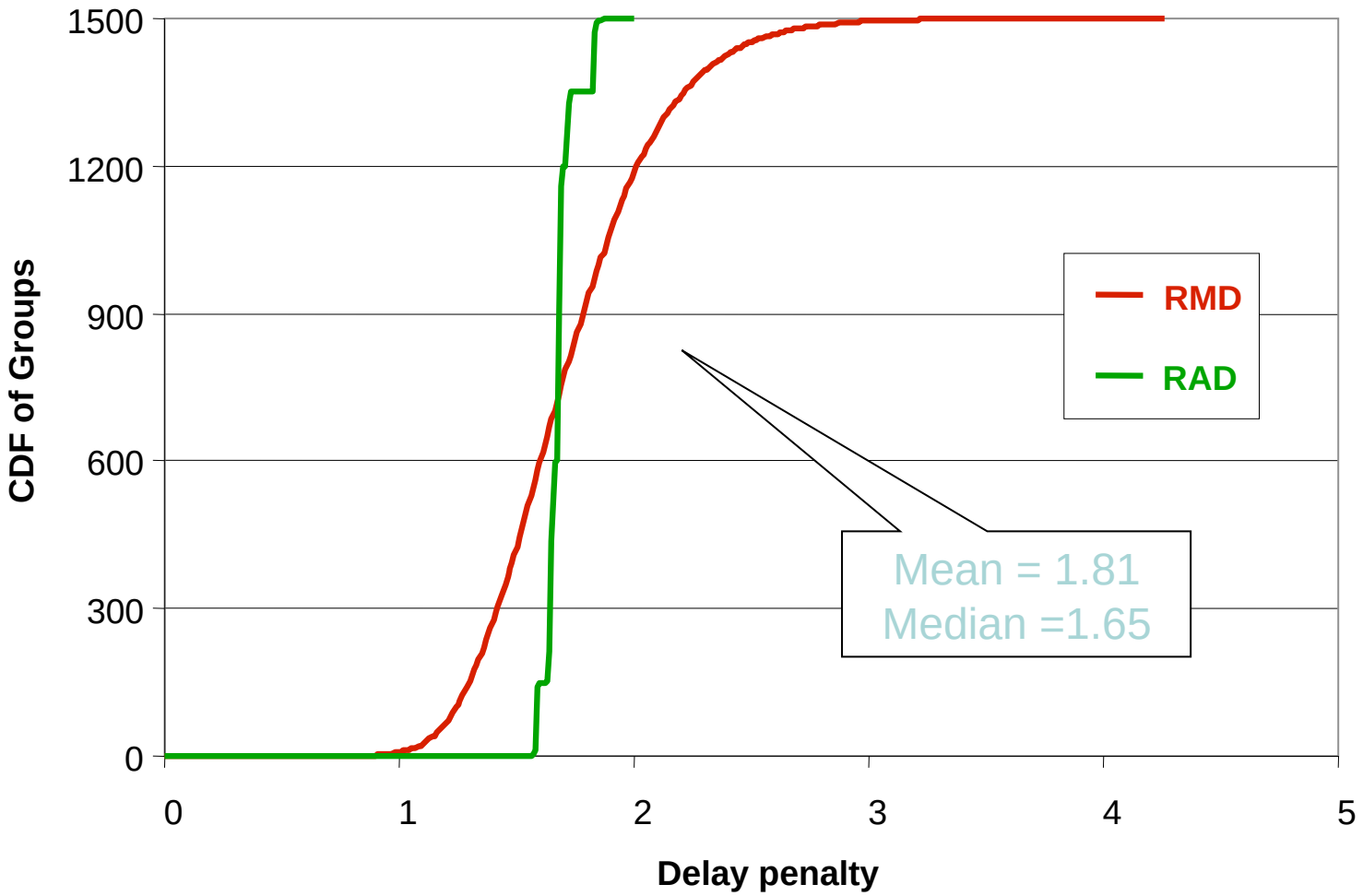
Scribe performance

- Discrete event simulator
- Evaluation metrics
  - Relative delay penalty
    - RMD: max delay$_{\text{app-mcast}}$ / max delay$_{\text{ip-mcast}}$
    - RAD: avg delay$_{\text{app-mcast}}$ / avg delay$_{\text{ip-mcast}}$
  - Stress on each network link
  - Load on each node
    - Number of entries in the routing table
    - Number of entries in the forwarding tables
- Experimental set-up
  - Georgia Tech *Transit-stub* model *(5050 core routers*)
  - 100 000 nodes chosen at random among 500 000
  - Zipf distribution for 1500 groups
  - Bandwidth not modeled

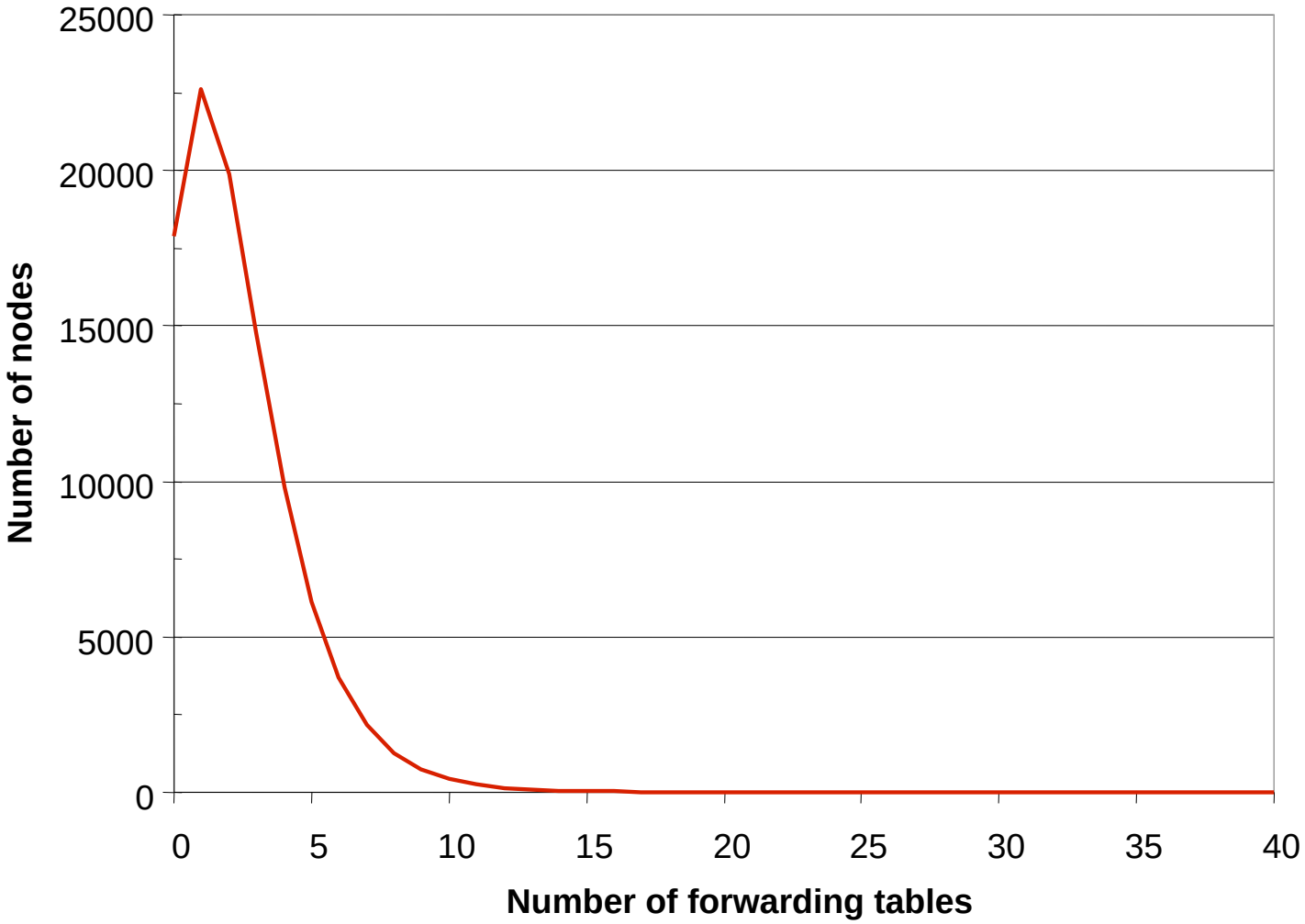
Group distribution



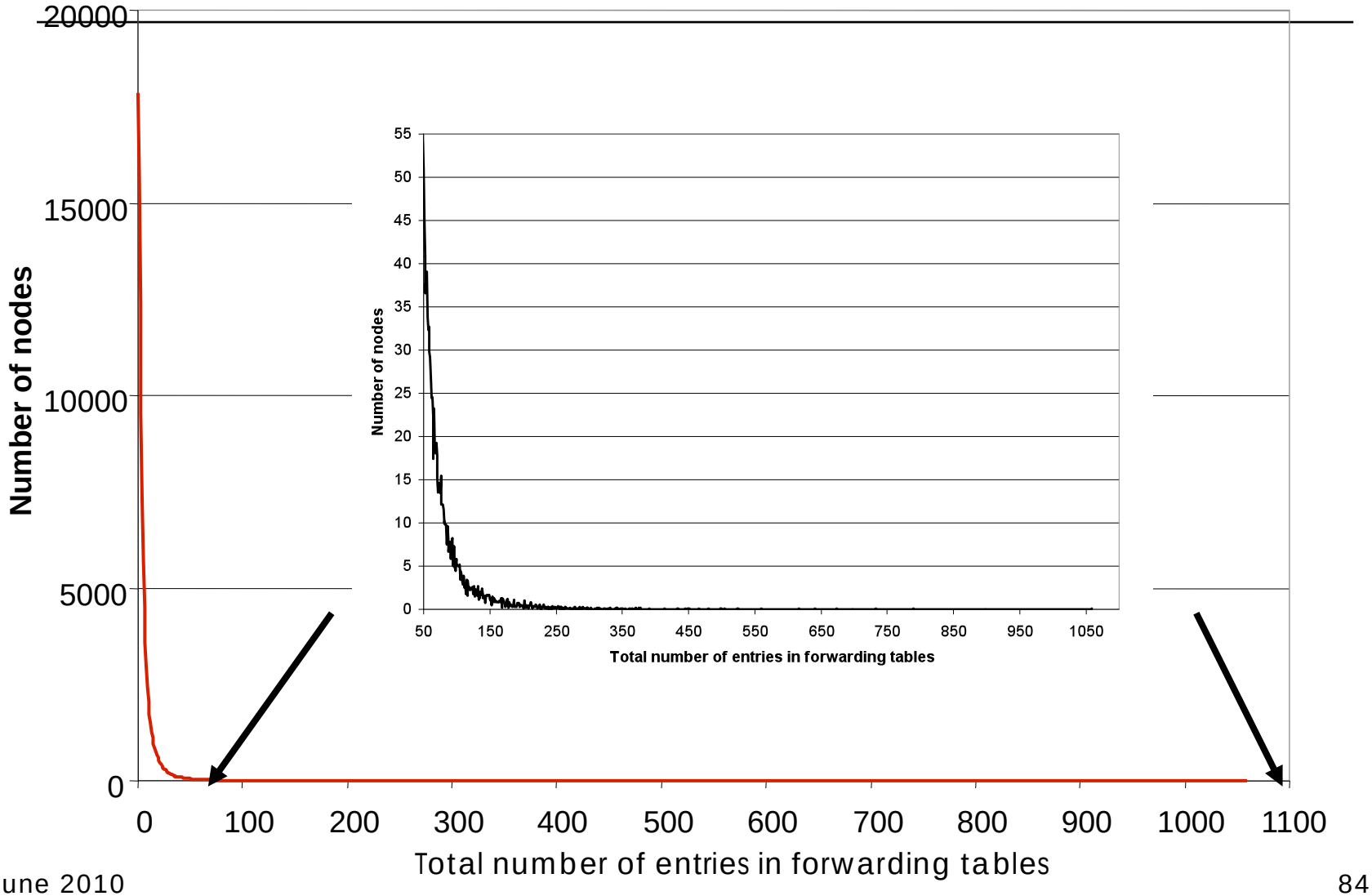
Delay/IP


1500
1200
900
600
300
0
CDF of Groups
0
1
2
3
4
5
Delay penalty
RMD
RAD
Mean = 1.81
Median =1.65

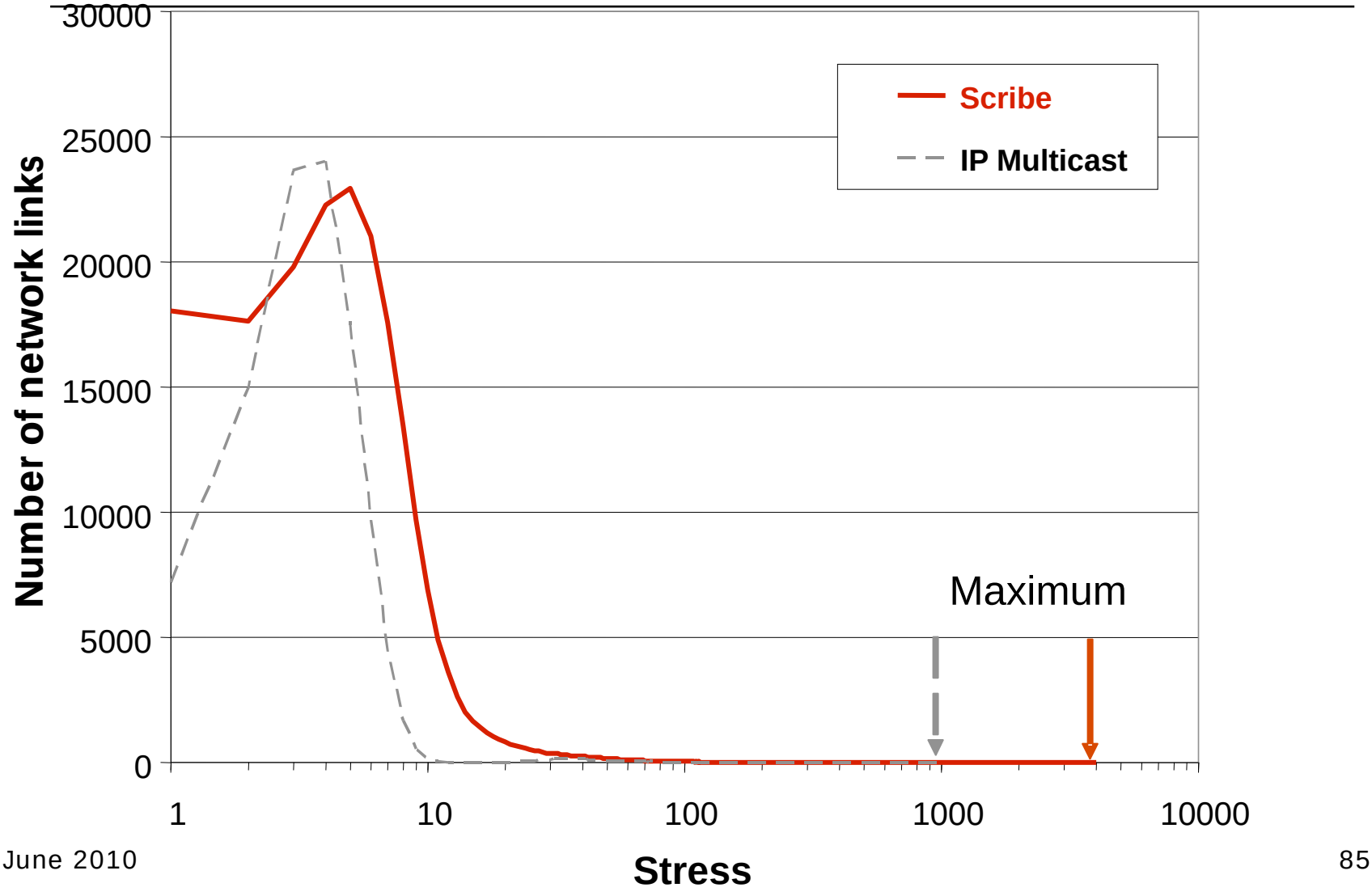
Load balancing



Load balancing



Network load



Summary

- Generic P2P infrastructures
  - Good support for large-scale distributed applications
  - ALM Infrastructure
- Scribe exhibits good performances/IP multicast
  - Large size groups
  - Large number of groups
  - Good load-balancing properties

CAN Multicast

- Flooding in a CAN network
  - Either
    - All CAN members are group members
  - Or
    - Mini CAN overlay creation/groupe

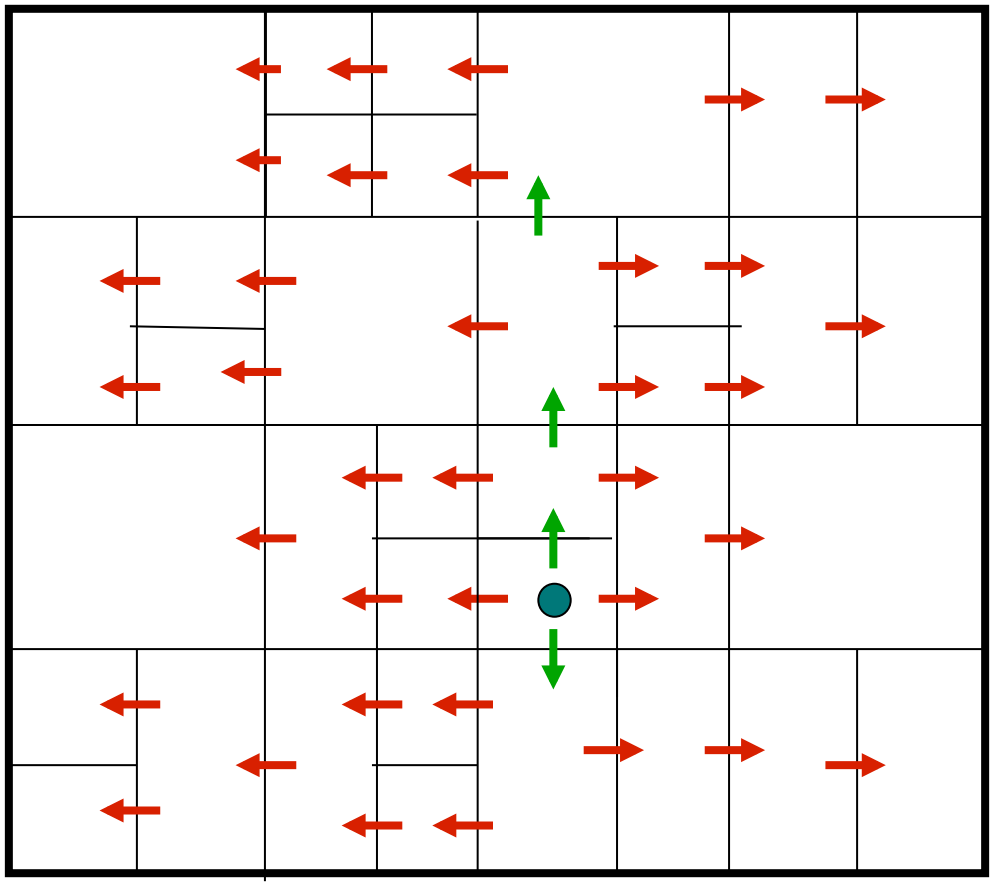
CAN multicast: group formation

- Subset of CAN network members forms a mini-CAN
- Group identifier associated to a point *(x,y)* in the CAN space.
- (x,y) is the bootstrap node for the mini-CAN
- Group join = mini-CAN join
- Iterations on the CAN join protocol

CAN multicast : message diffusion

- CAN network of dimension *d: 1….d*
- Each node maintains at least *2d neighbours*
- Diffusion
  - Source node sends the message to all its neighbours
  - A node receiving a message from dimension i
    - Forwards the message to its neighbours along the dimensions 1…(i-1)
    - Forwards the message to neighbours of dimension i in in the opposite direction (from the one it receives the message)
  - A node does not forward the message along a given dimension if the message has already traversed half of that dimension
  - A node does not forward an already received message

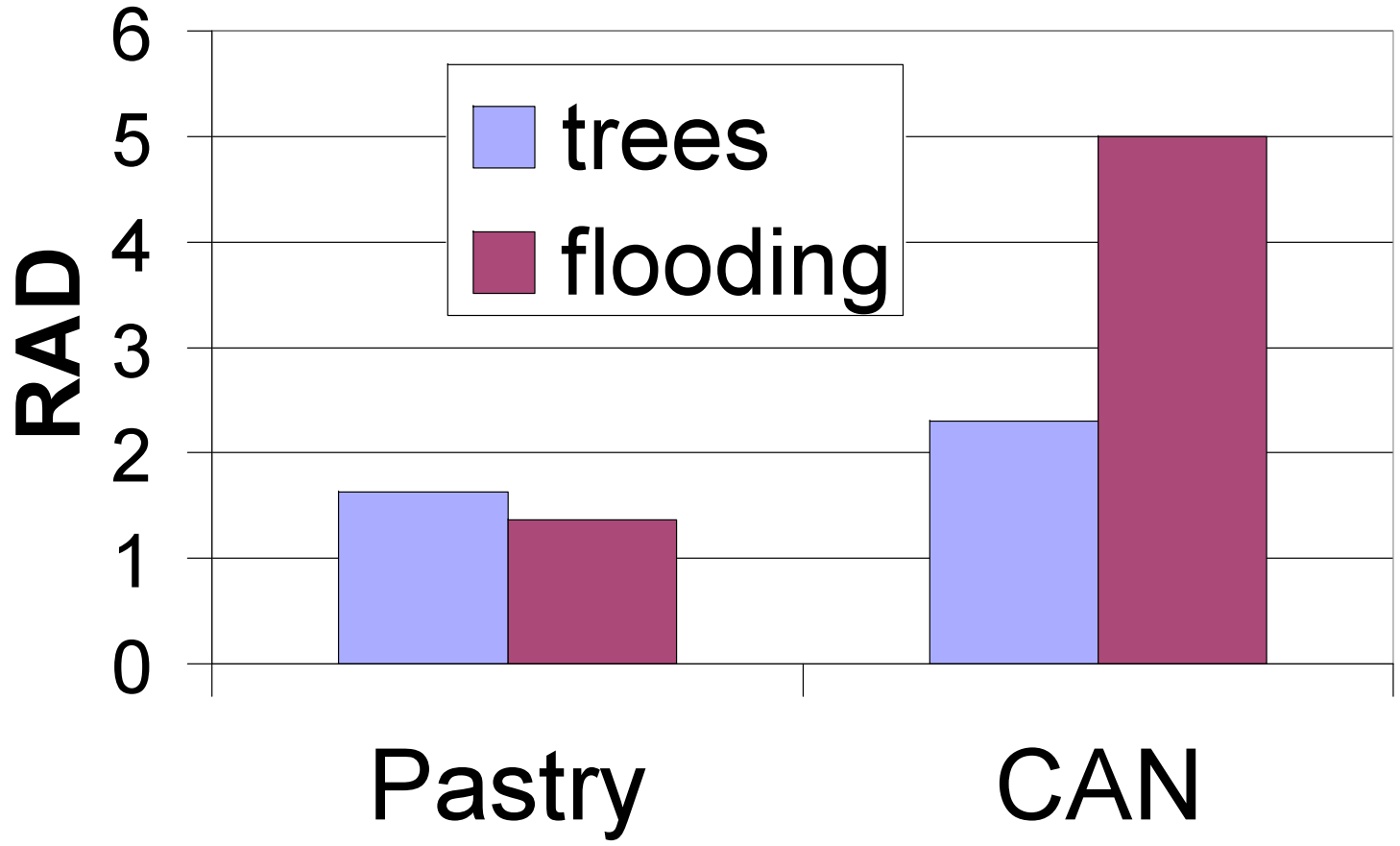
Example



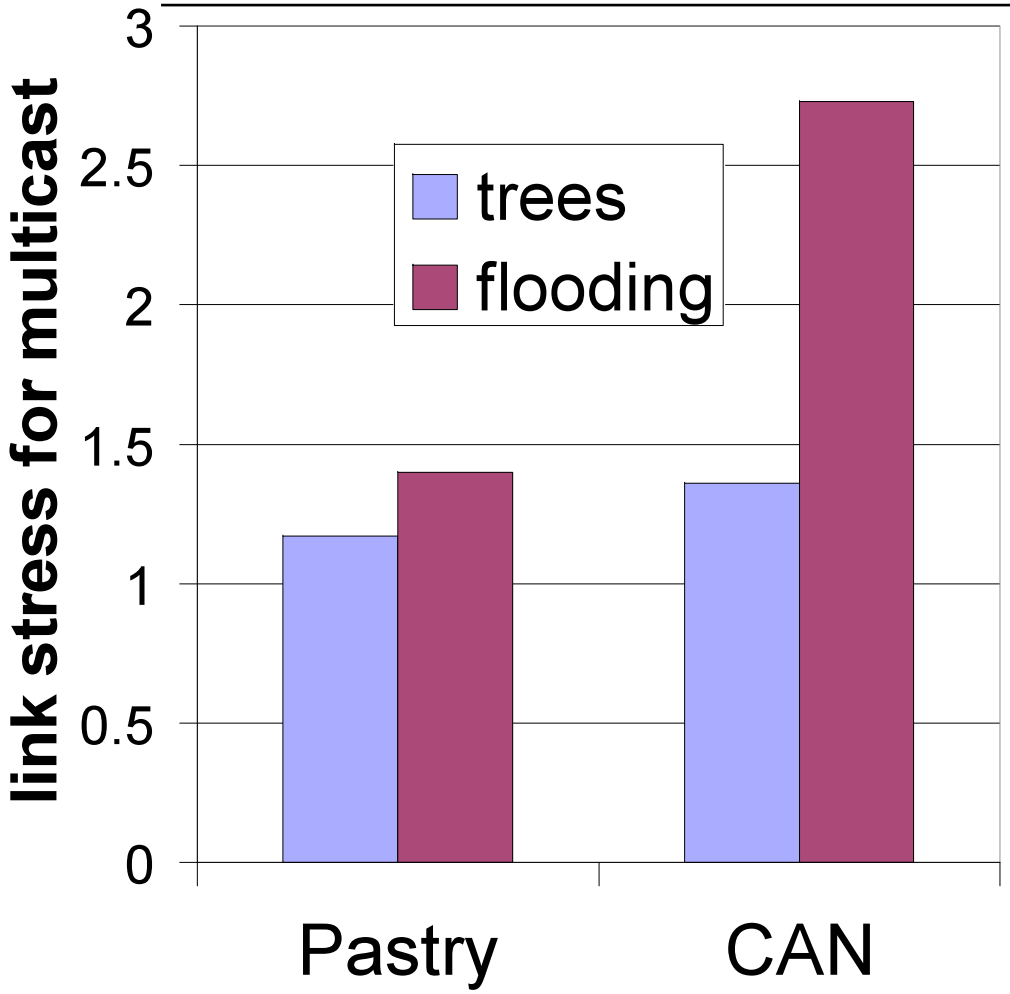
Can multicast : Performance

- CAN: 6 dimensions, group of 8192 nodes, transit-stub topology
- Relative delay penalty (RDP)
  - 5-6 for the majority of group members
- More details in the comparison

Comparison: delay penalty/IP



Comparison: average (physical) link stress



link stress for joining:

- identical for trees
- much larger for flooding
  - example: 281 on CAN

Trees versus flooding

- Tree-based multicast is more efficient
  - Lower delay and network stress during the multicast
  - Huge difference in the network trafic during group creation
  - Main drawback: some peers may be forwarders-only

Large-scale broadcast/multicast

Application-level multicast (ALM)

1. Structured peer to peer networks
   - Flooding
   - Tree-based
2. Content streaming
   - SplitStream

P2P ALM

- Tree-based protocols
  - Load unbalance: majority of nodes are leaves
  - Internal node failures
- Epidemic-based protocol
  - Redundancy by default
  - Potentially high network traffic
- The drawbacks are even more important when it comes to intensive contents
  - Load balancing
  - Network load

SplitStream approach

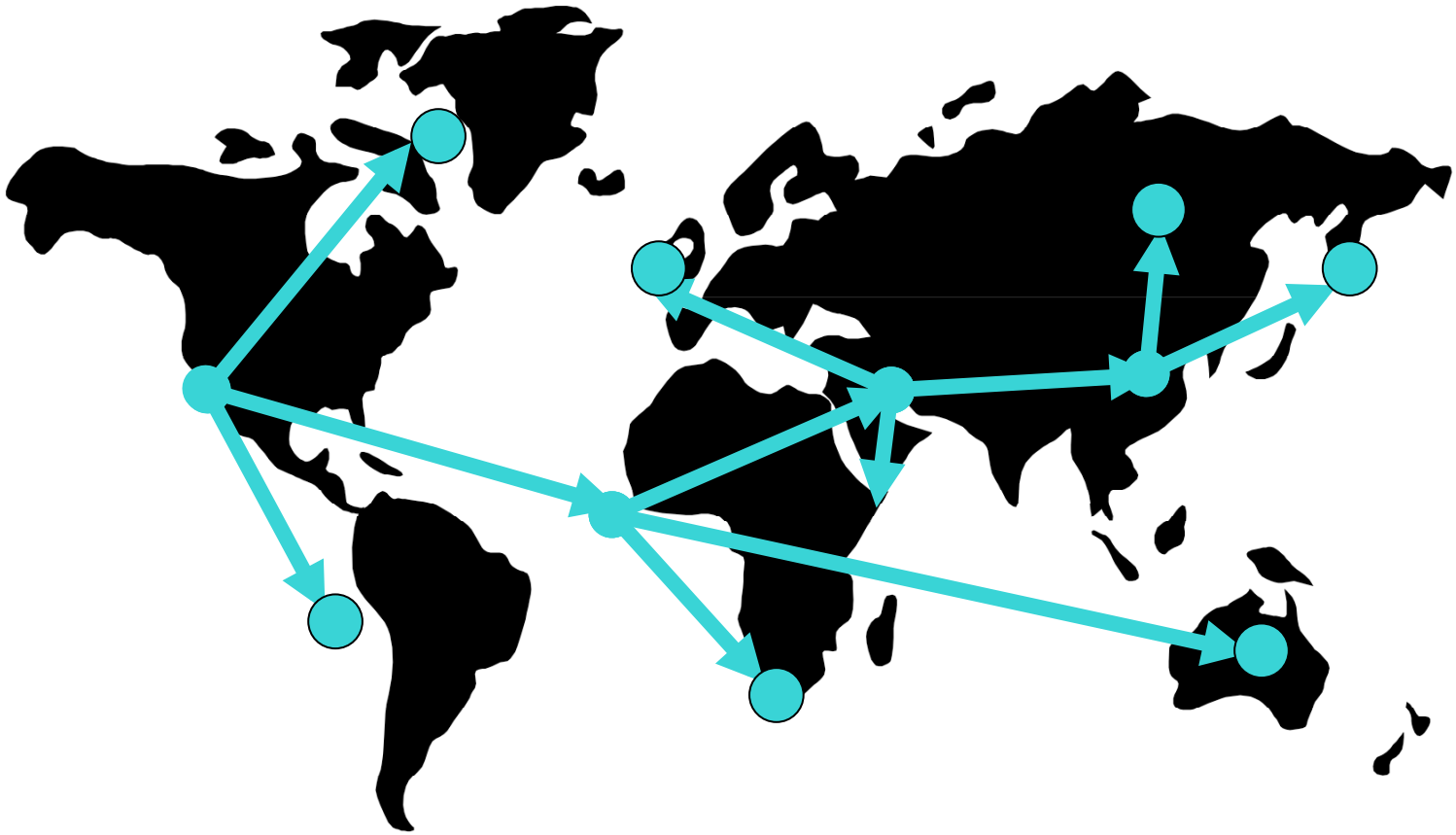
Content divided in *stripes*

Each stripe is distributed on an independent tree

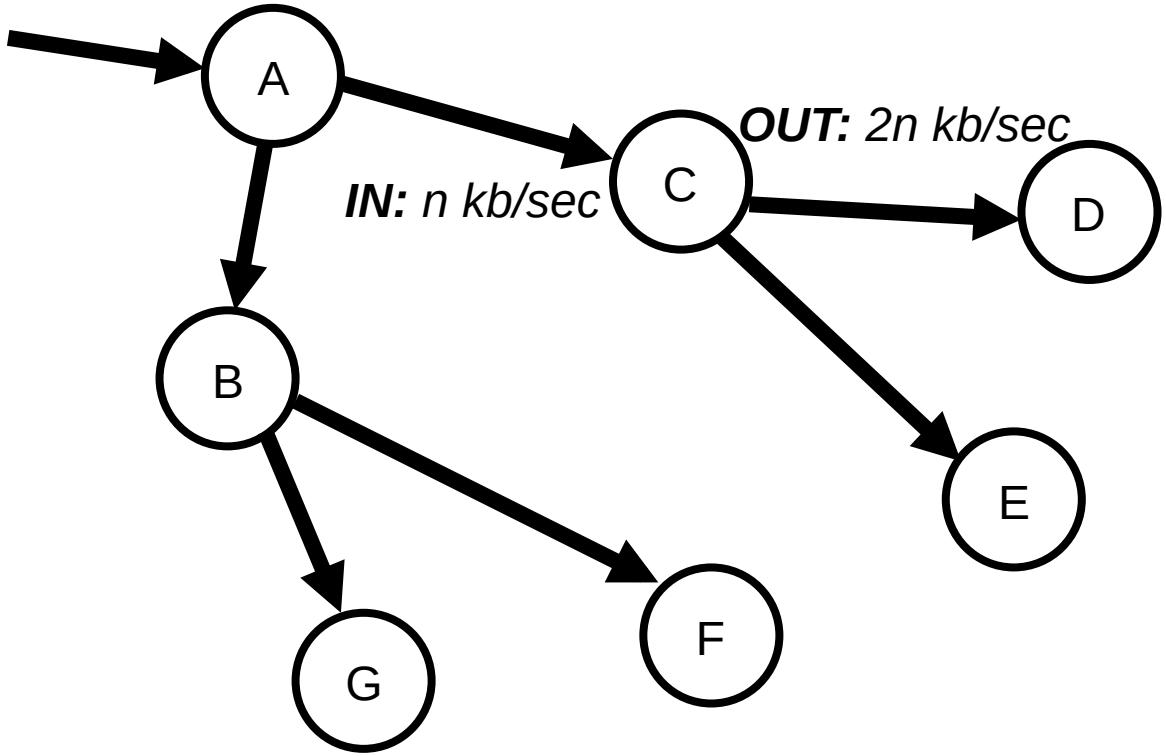
- Load balancing
  - Internal nodes in one tree are leaves in others
- Reliability
  - Failure of one load leads to unavailability of x stripes if parents are independent and using appropriate coding protocols

[SOSP 2003 « *SplitStream: High-Bandwidth Multicast in Cooperative Environment* »]

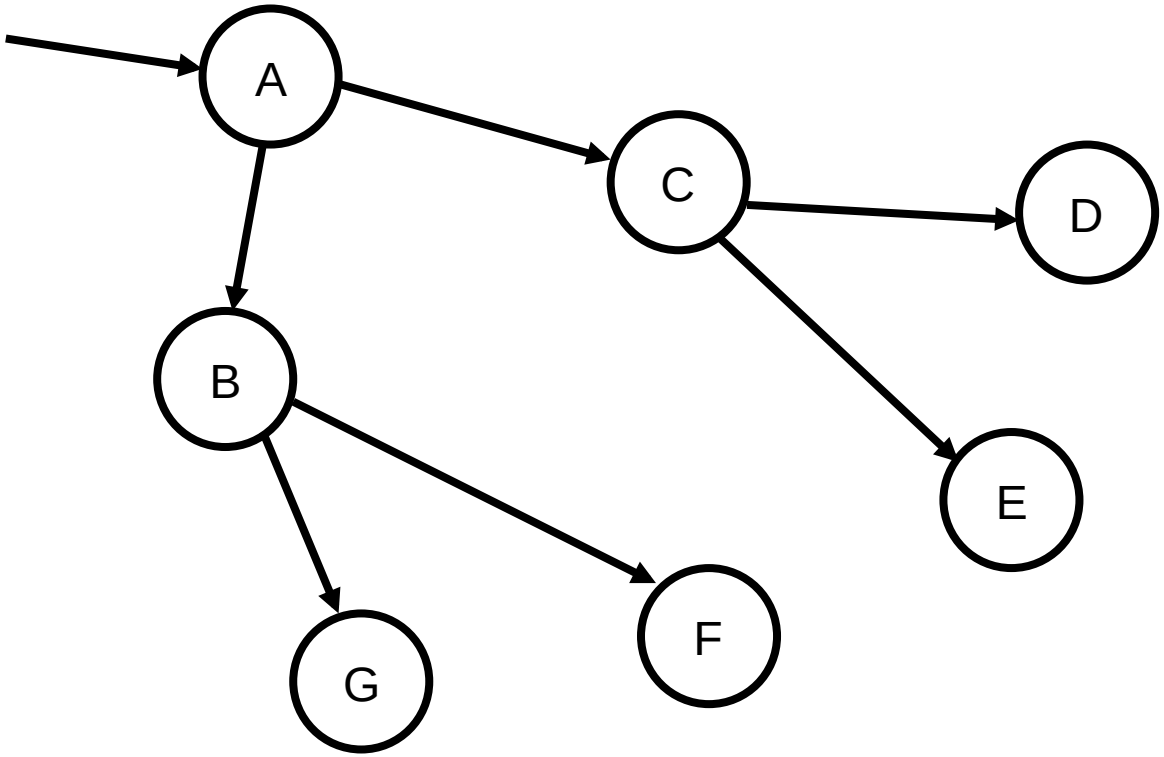
Tree-based ALM



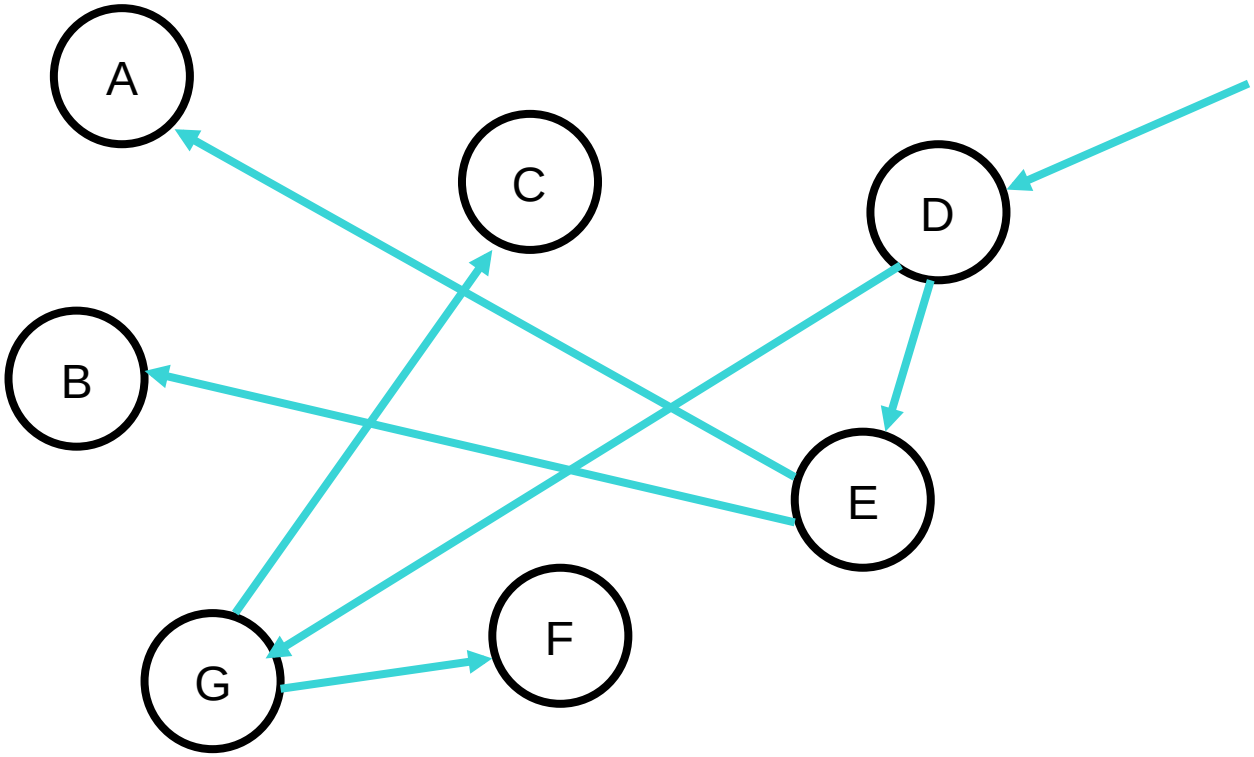
Tree-based ALM: unbalance



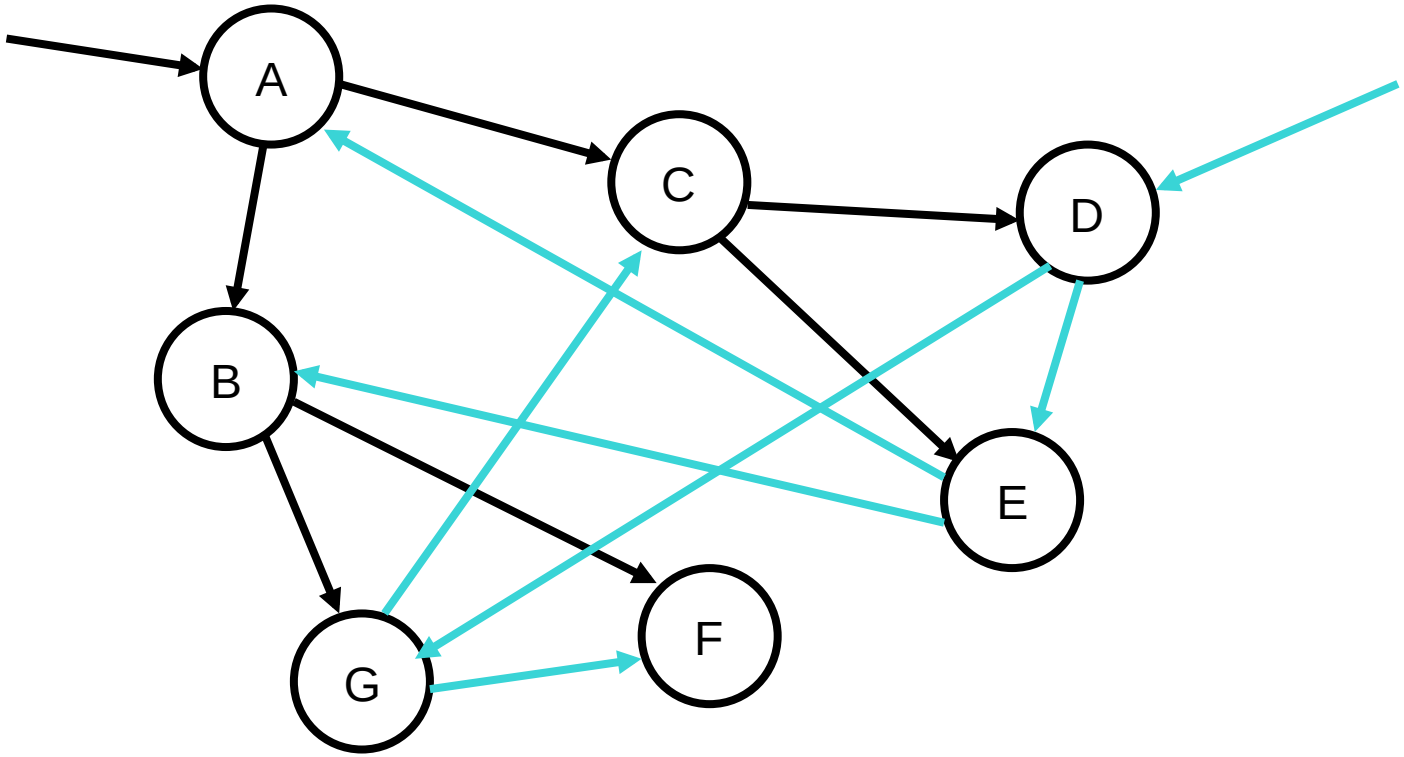
The SplitStream forest



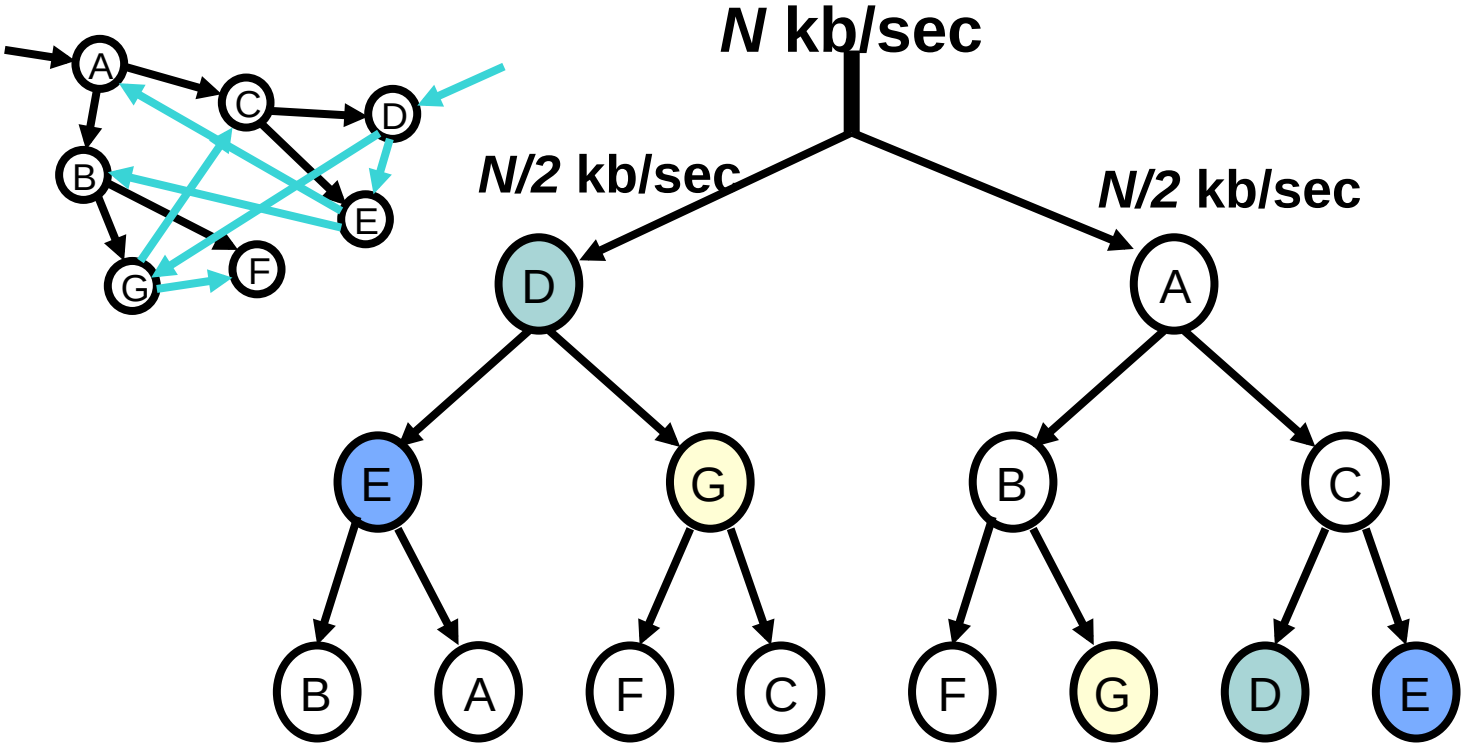
The SplitStream forest



The SplitStream forest

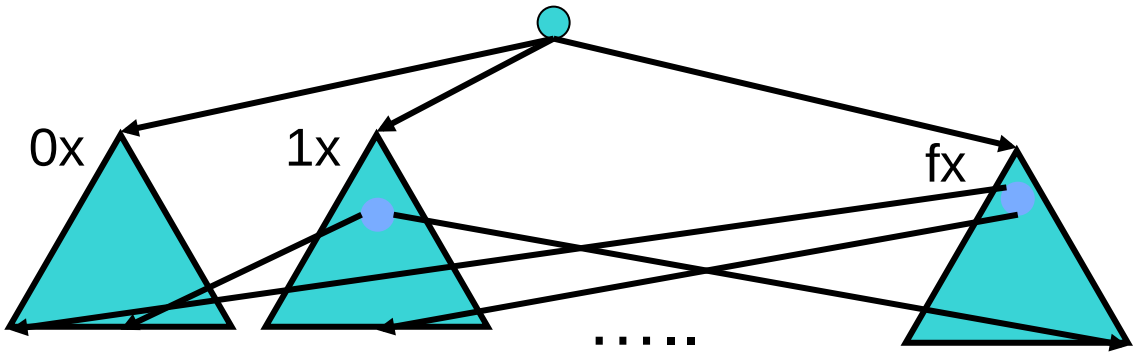


The SplitStream forest



SplitStream

- Construction of one tree/group per data stripe
- Each stripe identifier starts with a different digit (independence up to 16 stripes)



SplitStream

Main issue: build and maintain multiple multicast tree in a fully decentralized and reliable way so that

- Each client receives the desired number of stripes
- Independent trees
- Control upon bandwidth allocation
- Reasonable latency and network load

Leverage Scribe/Pastry

- Pastry: P2P routing infrastructure (structured, efficient, reliable)
- Scribe: decentralized and efficient tree-based protocol

SplitStream: forest managements

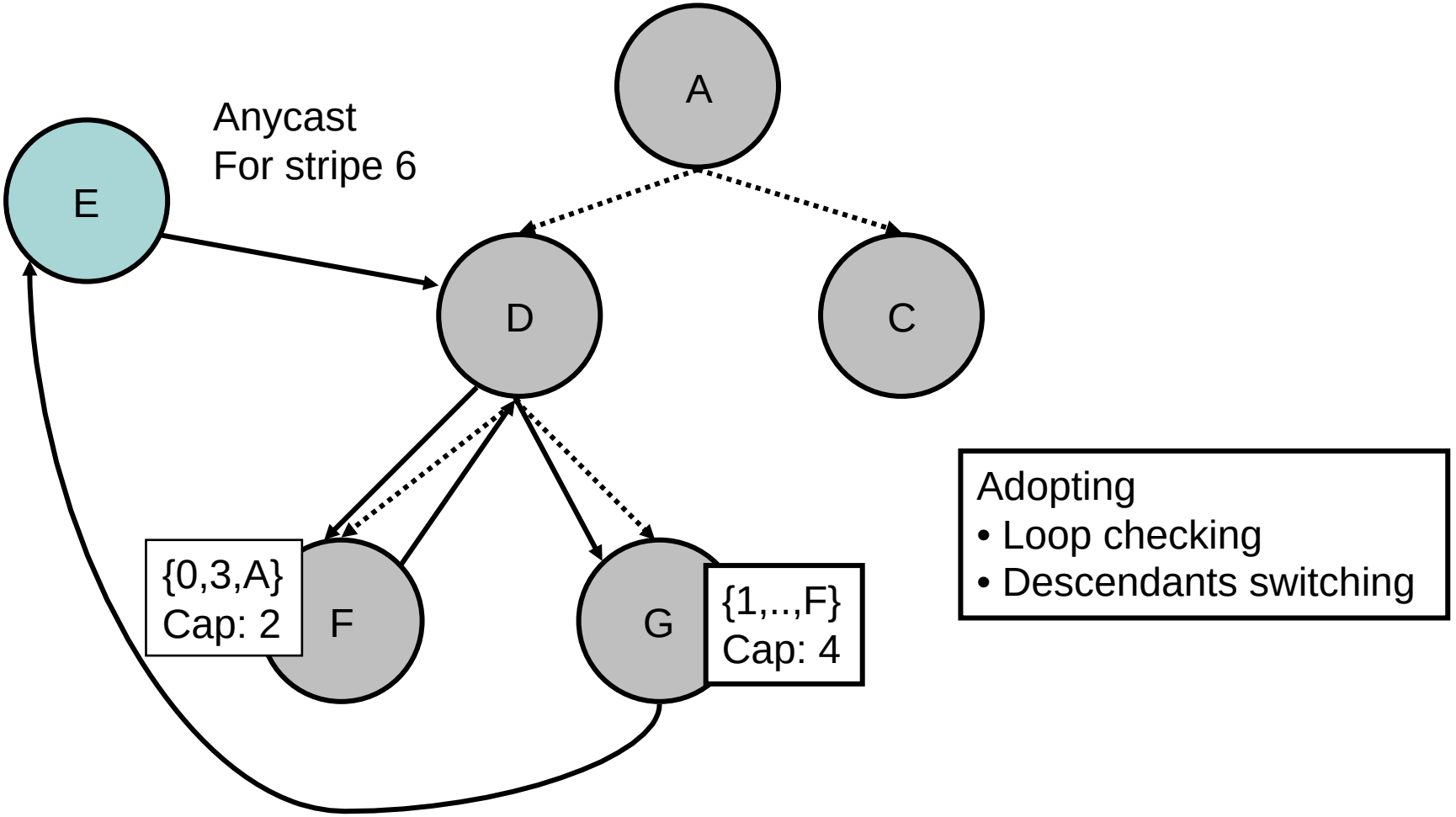
Constraints

- Limited out-degree potentially increases the tree depth
- Load balancing to ensure within trees and between trees
- Failure independence of trees .

Solution: spare capacity tree

- Overloaded nodes push descendents down (Scribe)
- Underloaded nodes join the spare capacity tree
- Overloaded nodes give up descendents
- Orphans *anycast* to the spare capacity tree to discover new parents

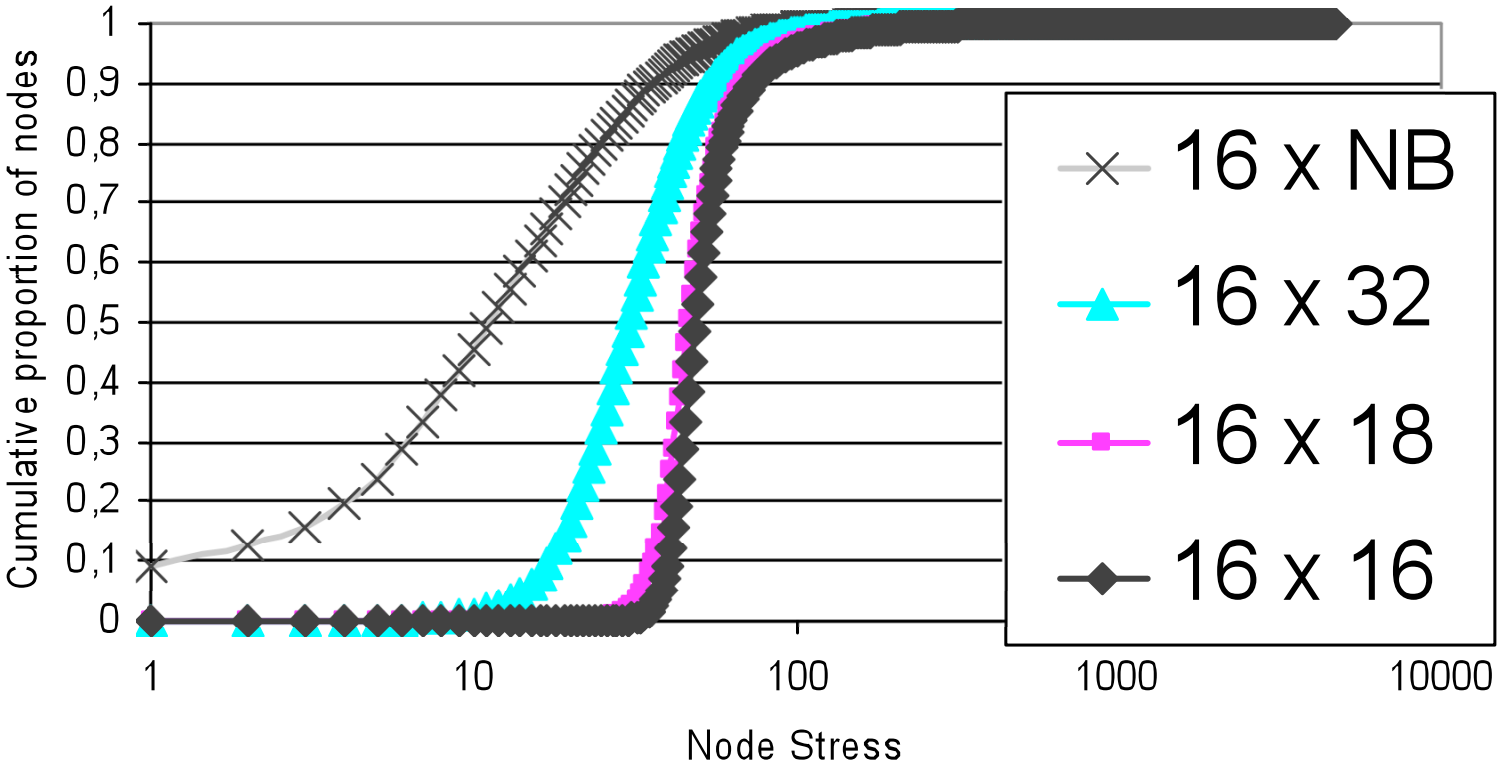
Spare capacity tree



Experiments

- Simulations (average on 10 runs)
  - Topologies **GT**, Mercator, MS Corp.
  - 40000 nodes
- Pastry (b=4, leafset = 16)
- SplitStream : 16 stripes
- Configurations in-degree x out-degree
  - Impact of spare capacity 16x16, 16x18, 16x32 and 16xNB
  - Impact of capacity/needs (Gnutella)
- Failure resilience
  - Path diversity
  - Catastrophic failures (25% of faulty nodes) in a 10,000 node system
- Results
  - Forest construction
  - Multicast performance

Forest construction: load on each node



Forest construction: load on each node

Configuration	16x16	16x18	16x32	16xNB
Max	2971	1089	663	472
Mean	57.2	52.6	35.3	16.9
Med	49.9	47.4	30.9	12

Load decreases as the spare capacity increases

16xNB: no *pushdown* nor orphans

•16x16: each node contacts the spare capacity tree for 8 stripes on average

• Nodes with id close to the spare capacity tree get the highest load

Forest construction: network load

Measured as the number of msg on physical links

Configuration	16x16	16x18	16x32	16xNB
Max	5893	4285	2876	1804
Mean	74.1	65.2	43.6	21.2
Med	52.6	48.8	30.8	17

Load decreases as the spare capacity increases
Maximum approx. 7 times < centralized system

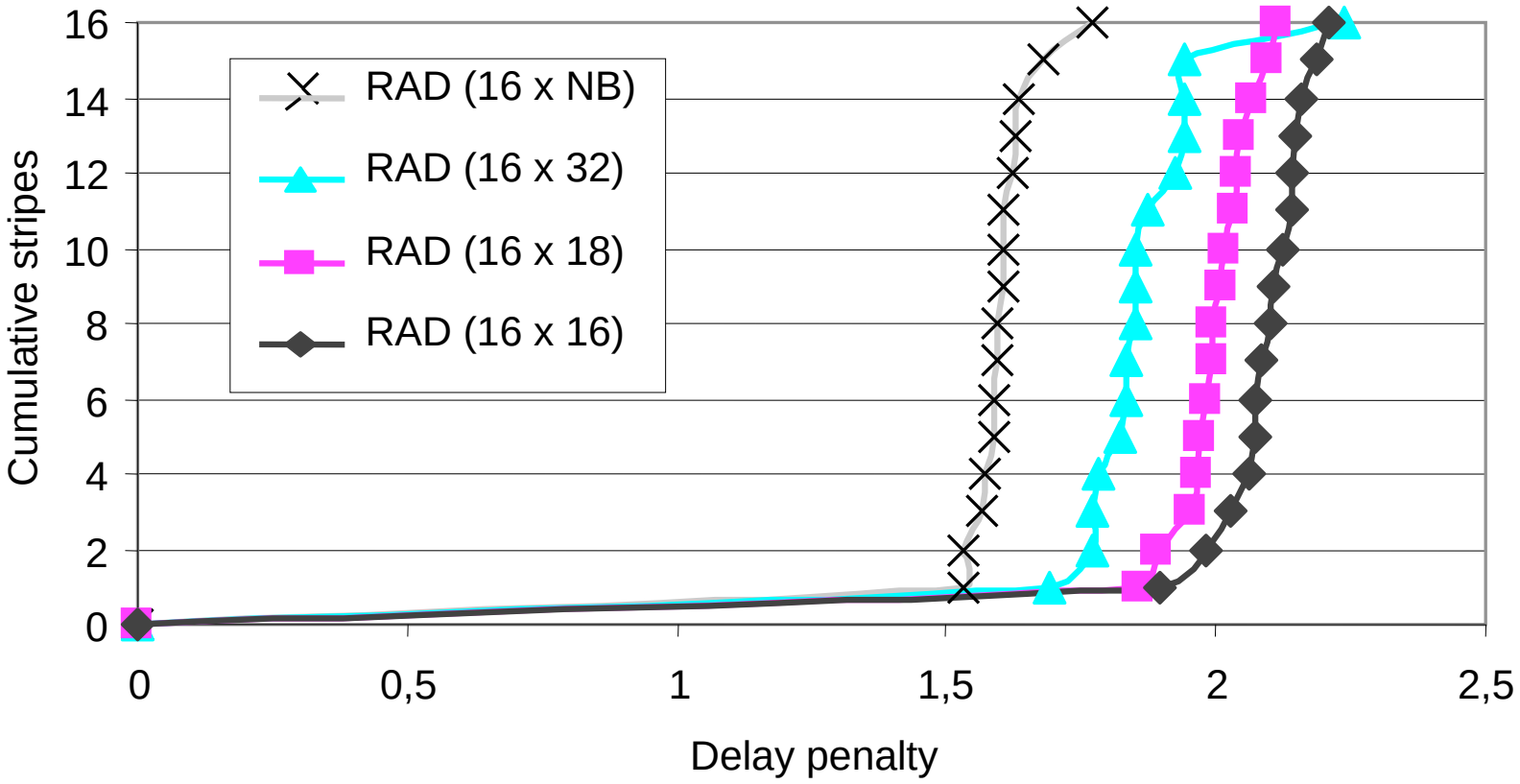
Multicast: link stress

One message/stripe, no failure

Configuration	Centralized (0.43)	Scribe (0.47)	IP (0.43)	16x16 (0.98)	16x18	16x32	16xNB
Max	639984	3990	16	1411	1124	886	1616
Mean	128.9	39.6	16	20	19	19	20
Med	16	16	16	16	16	16	16

- 16xNB : absence of forwarding bounds causes contention on a small
- Set of links
- Splitstream uses a larger fraction of links but load them less

Delay penalty during multicast



Path diversity

•Number of lost stripes (at most) on each node when the most significant ancestor is faulty (worst case scenario)

Configuration	16x16	16x32	16xNB
Max	6.8	6.6	1
Mean	2.1	1.7	1
Med	2	2	1

Summary

- SplitStream: robust and efficient protocol for large-scale content streaming
  - Forest of independent trees / unique tree
  - Spare capacity tree for maintenance
  - Decentralized and scalable management relying on Scribe and Pastry
  - Robust in dynamic environments

References

- M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "SCRIBE: A large-scale and decentralised application-level multicast infrastructure", IEEE Journal on Selected Areas in Communication (JSAC), Vol. 20, No, 8, October 2002.
- M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment", SOSP'03, Lake Bolton, New York, October, 2003.
- M. Castro, M. B. Jones, A-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang and A. Wolman, "*An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer overlays*", Infocom 2003, San Francisco, CA, April, 2003.
- Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy Katz John Kubiatowicz « *Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination* »Eleventh International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2001)
- Sylvia Ratnasamy, Mark Handley, Richard Karp, Scott Shenker « Application-level Multicast using Content-Addressable Networks » (2001) Lecture Notes in Computer Science, NGC 2001 London.
- D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. « *Bullet: High bandwidth data dissemination using an overlay mesh* ». In 19th ACM Symposium on Operating Systems Principles, October 2003.
- Suman Banerjee, Christopher Kommareddy, Koushik Kar, Bobby Bhattacharjee, Samir Khuller « *Construction of an Efficient Overlay Multicast Infrastructure for Real-Time Applications* » (INFOCOM 2003)