

P2P Epidemics: Managing very large scale overlays

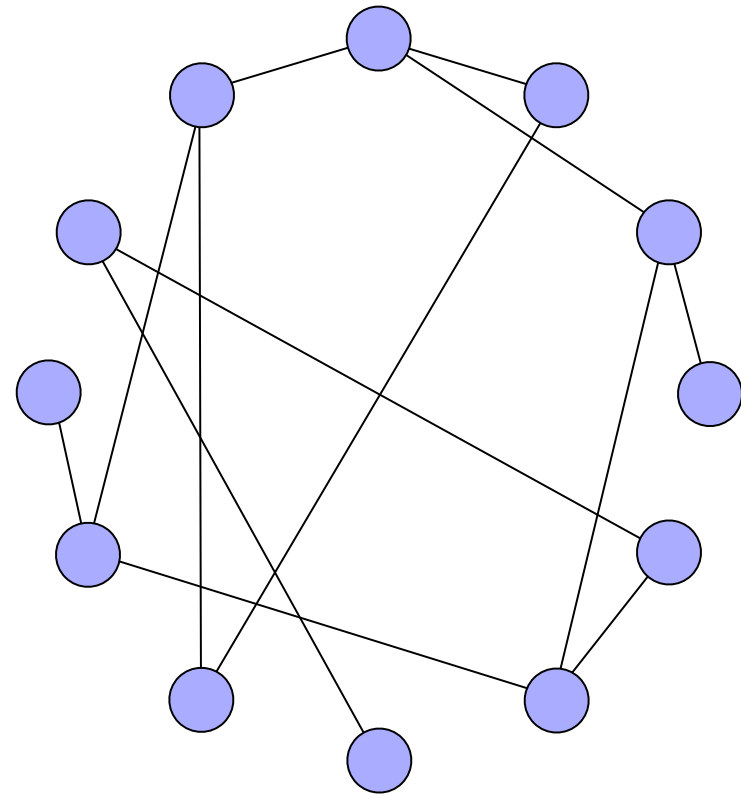
Spyros Voulgaris

vrije Universiteit amsterdam



Overlay

- Nodes jointly form an *overlay*
- Overlay consists of *logical links* between nodes

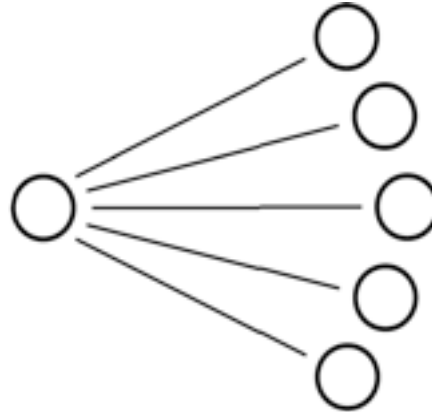


Overlay

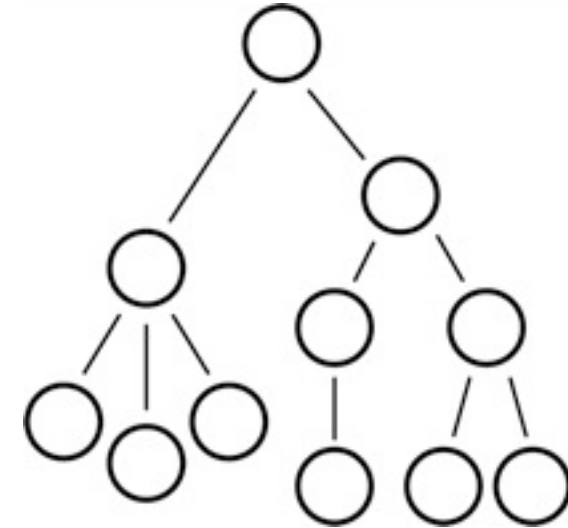
■ Unstructured

- No explicit topology
- *Observed rather than engineered*
- Examples: Gnutella, Freenet

Centralized

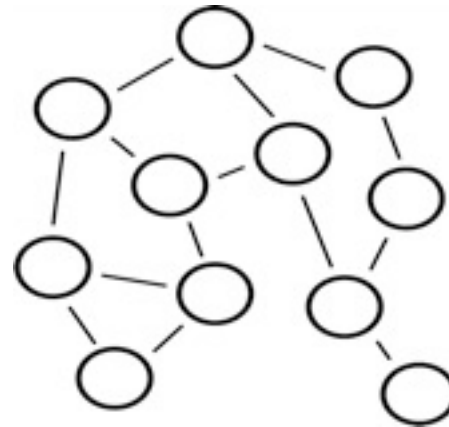


Hierarchical

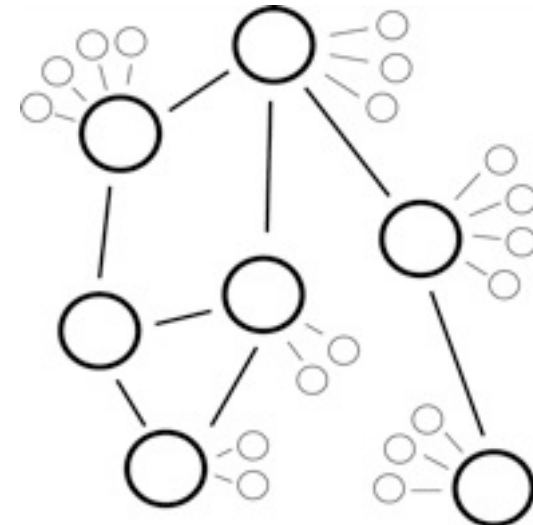


■ Structured

- *An explicit "shape" is maintained*
- Examples:
 - Rings, Trees, DHTs
 - But also random topologies



Decentralized



Hybrid

What we do

- We build overlays, with certain properties
- We are interested in
 - Large-scale (>10K nodes)
 - Highly dynamic
 - Robust connectivity & Survivability
 - Fast adaptivity
 - Certain properties, such as clustering, avg. path length, load balancing, etc.
- Our key philosophy is
 - LET CONTROL GO!
 - Obtain desired properties from *emergent* behaviour



Outline

- Peer Sampling Service
- Two different implementations
 - NEWSCAST
 - CYCLON
- Secure Peer Sampling

System model

- A dynamic collection of distributed nodes that want to participate in a common epidemic protocol
 - Node may join / leave
 - Node may crash at any time
 - Communication:
 - To communicate with node q , node p must know its address
 - Messages can be lost – high levels of message omissions can be tolerated
- } Churn

Peer Sampling Service

- Motivation

- To provide random nodes to epidemic protocols

- The input:

- The distributed collection of nodes

- The output:

- `getPeer()`

returns a peer as a result of an independent uniform random sampling among the collection

Epidemic Networks

Traditional

- Nodes have **full view** of the network
- Each node periodically “gossips” with a random node, out of the **whole set**



Epidemic Networks

Traditional

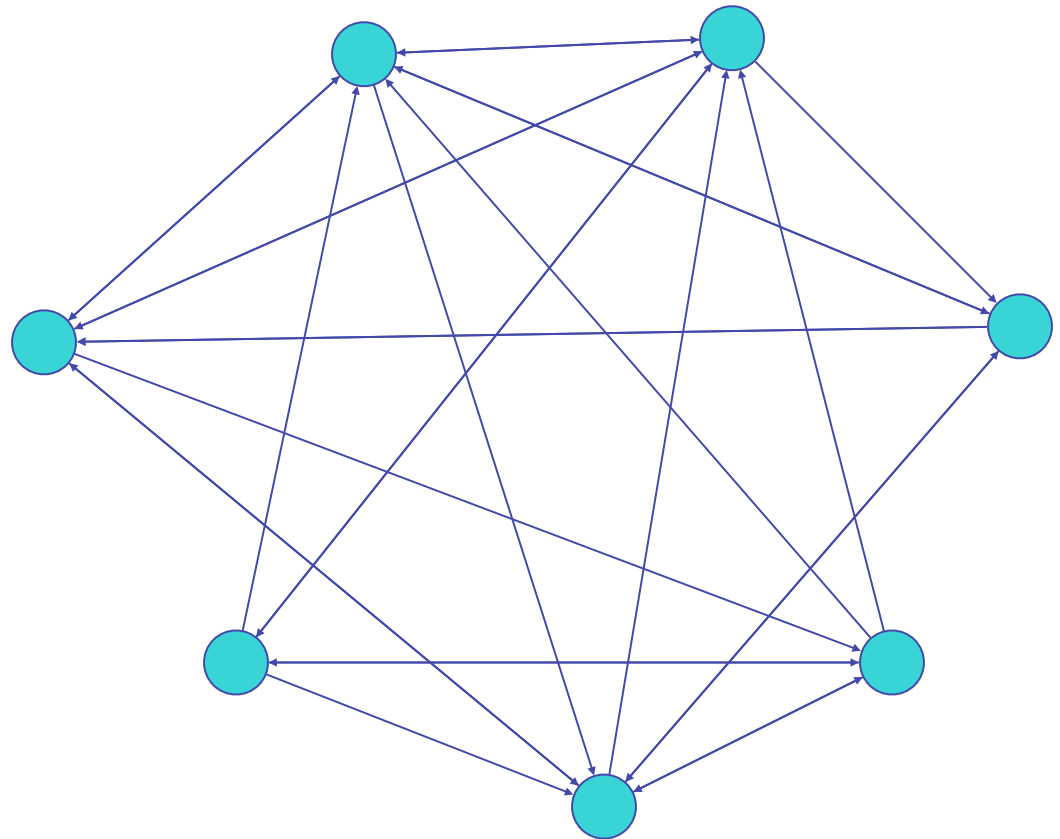
- Nodes have **full view** of the network
- Each node periodically “gossips” with a random node, out of the **whole set**

Reality

- Nodes have a **partial view** of the network (a set of “neighbors”)
- Each node periodically “gossips” with a random node, out of its **partial view**

Our Epidemic Overlays

- Each node has a *view* containing C neighbors ($C = \text{view size}$)
- Each node periodically contacts a neighbor
- They *gossip* and update their views





Neighbor descriptor

- Descriptors of node p contains
 - The address needed to communicate with p
 - Additional information that may be needed by different implementations of the peer sampling service
 - Additional information that may be needed by upper layers

A generic algorithm

// *view* is a collection of neighbors

Init: *view* = { *descriptor(s)* of a node already in the system }

// **active thread**

// executed by *p*

do once every
 δ time units

q = **selectNeighbor**(*view*)

msg_p = **extract**(*view*, *q*)

send *msg_p* to *q*

receive *msg_q* from *q*

view = **merge**(*view*, *msg_q*)

// **passive thread**

// executed by *p*

do forever

receive *msg_q* from *

msg_p = **extract**(*view*, *q*)

send *msg_p* to *q*

view = **merge**(*view*, *msg_q*)

A "cycle"
of length

δ



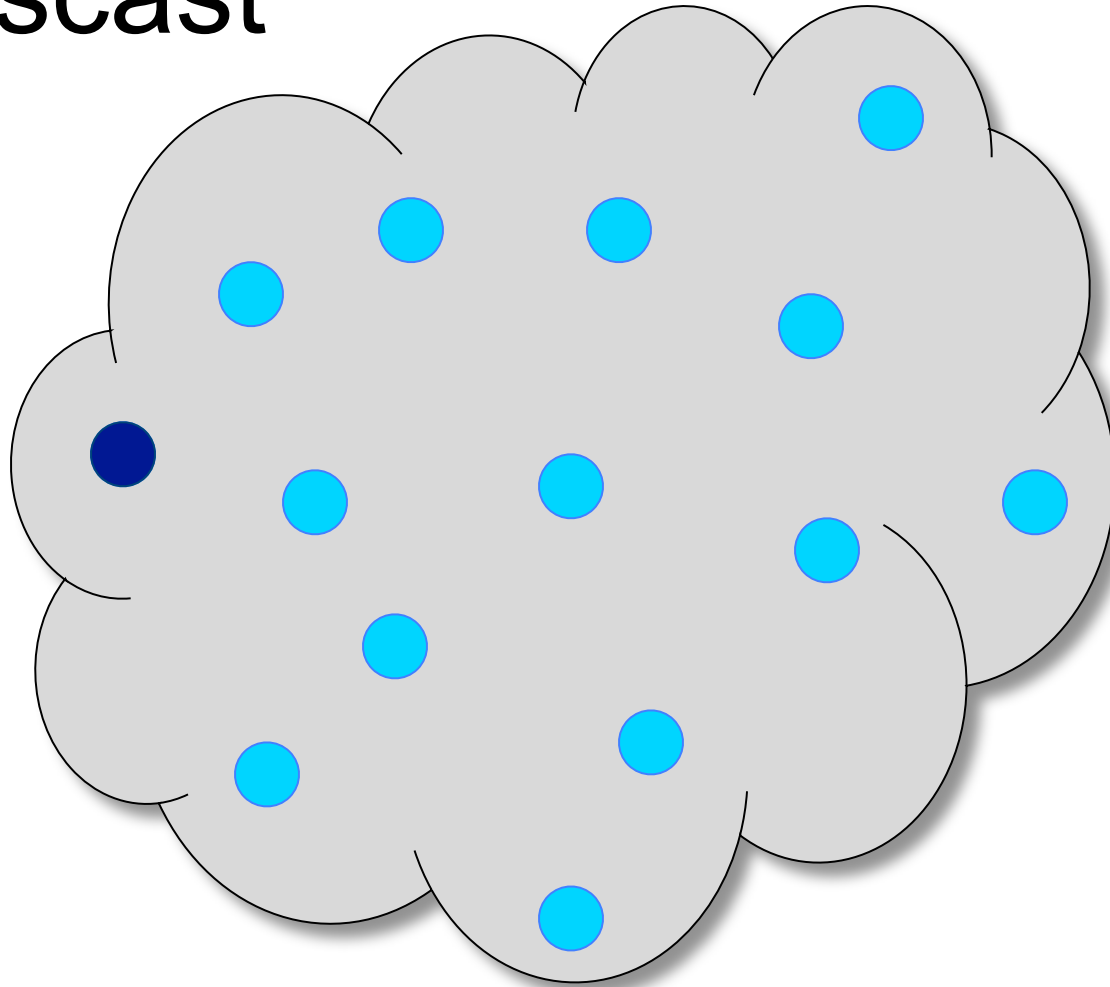
A generic algorithm

- `selectNeighbor()`
 - select one of the neighbor contained in the view
- `extract(view, q)`
 - returns a subset of the descriptors contained in the local view
 - may add other descriptors (e.g. its own)
- `merge(view, msgq)`
 - returns a subset of the descriptors contained in the union of the local view and the received view




Newscast

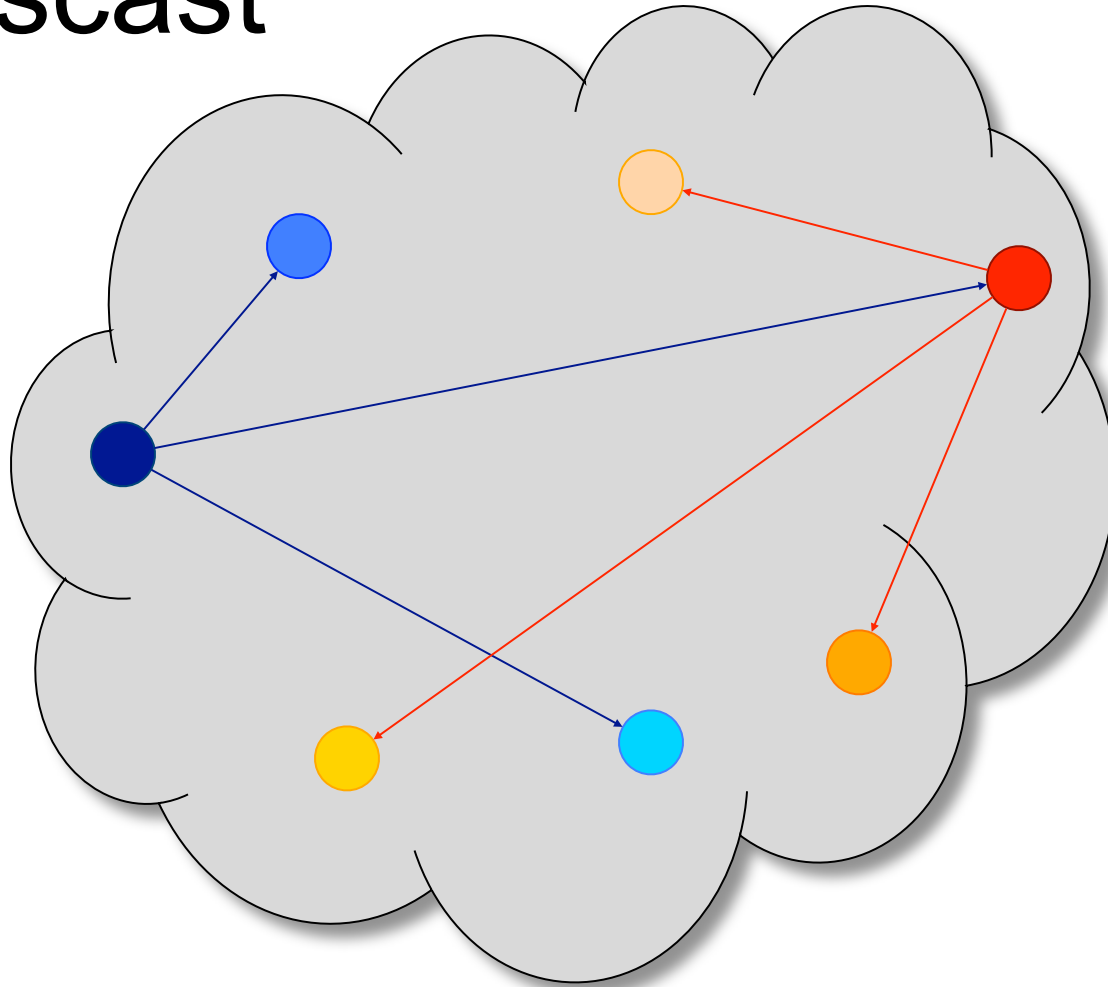
- Descriptor: address + timestamp
- `selectNeighbor()`
 - select one node at random
- `extract(view, q)`
 - returns the entire view + a local descriptor with a fresh timestamp
- `merge(view, msgq)`
 - returns the C freshest identifiers (w.r.t. timestamp) from the union of local view and message




Newscast






Newscast

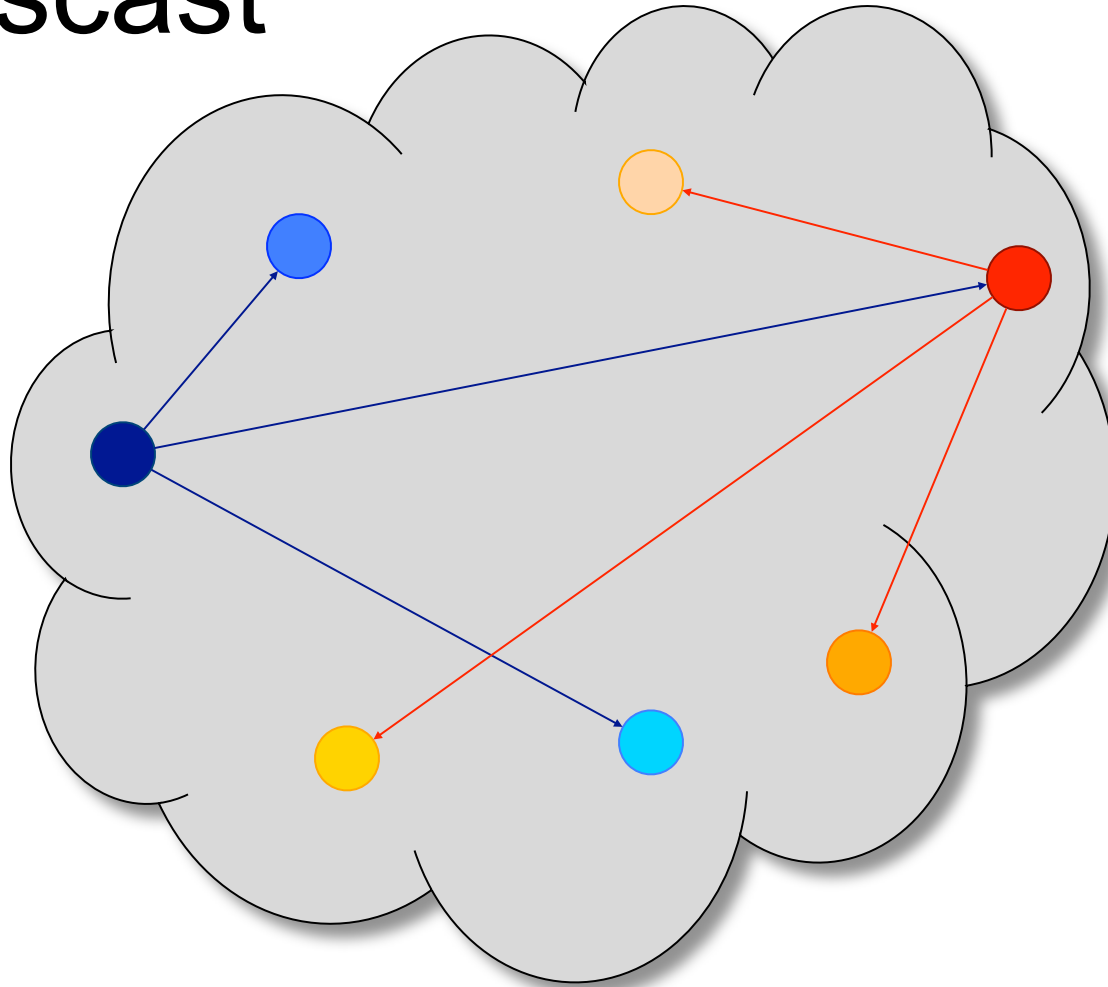
ID & Address	Time stamp
	9
	12
	16






ID & Address	Time stamp
	7
	10
	14

Newscast




ID & Address	Time stamp
	9
	12
	16

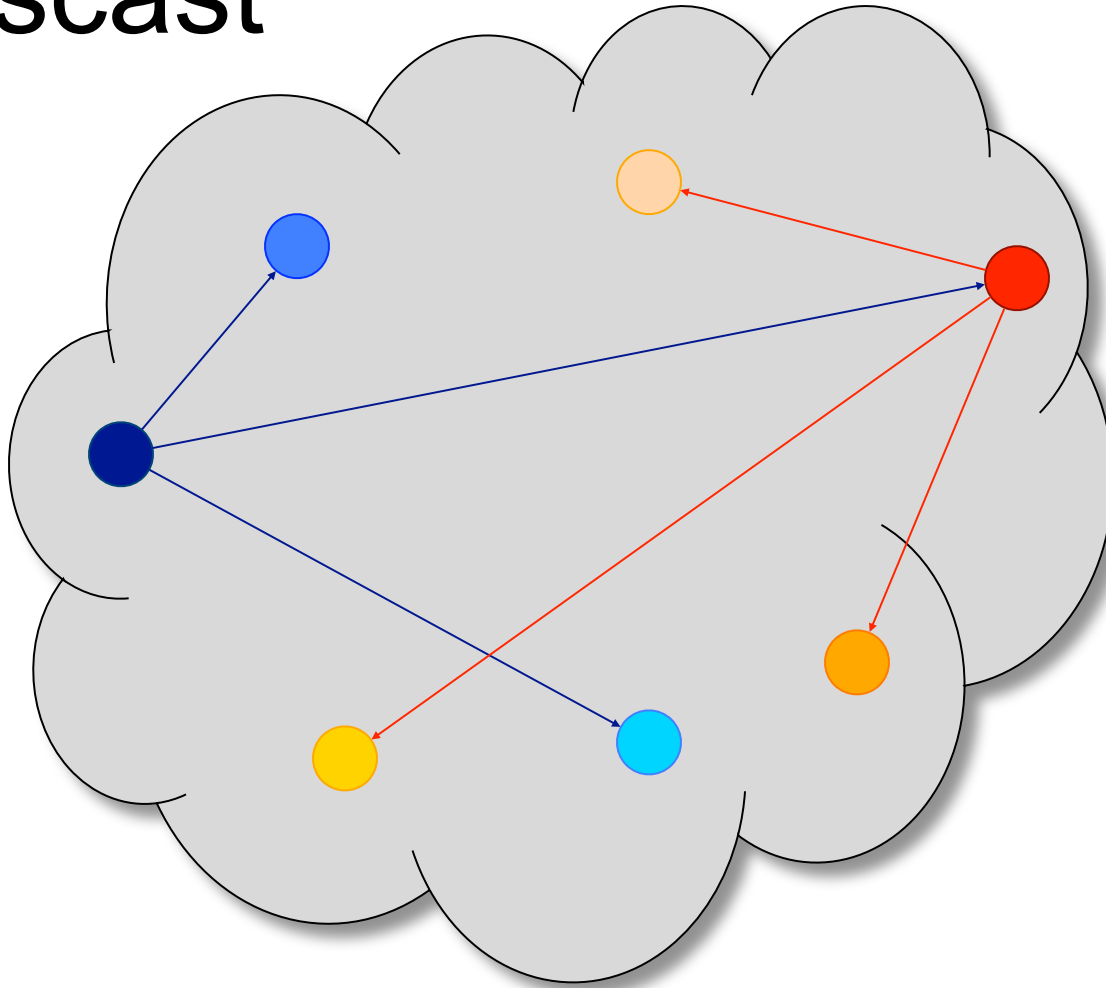





ID & Address	Time stamp
	7
	10
	14

1. Pick random peer from my view

Newscast




ID & Address	Time stamp
	9
	12
	16

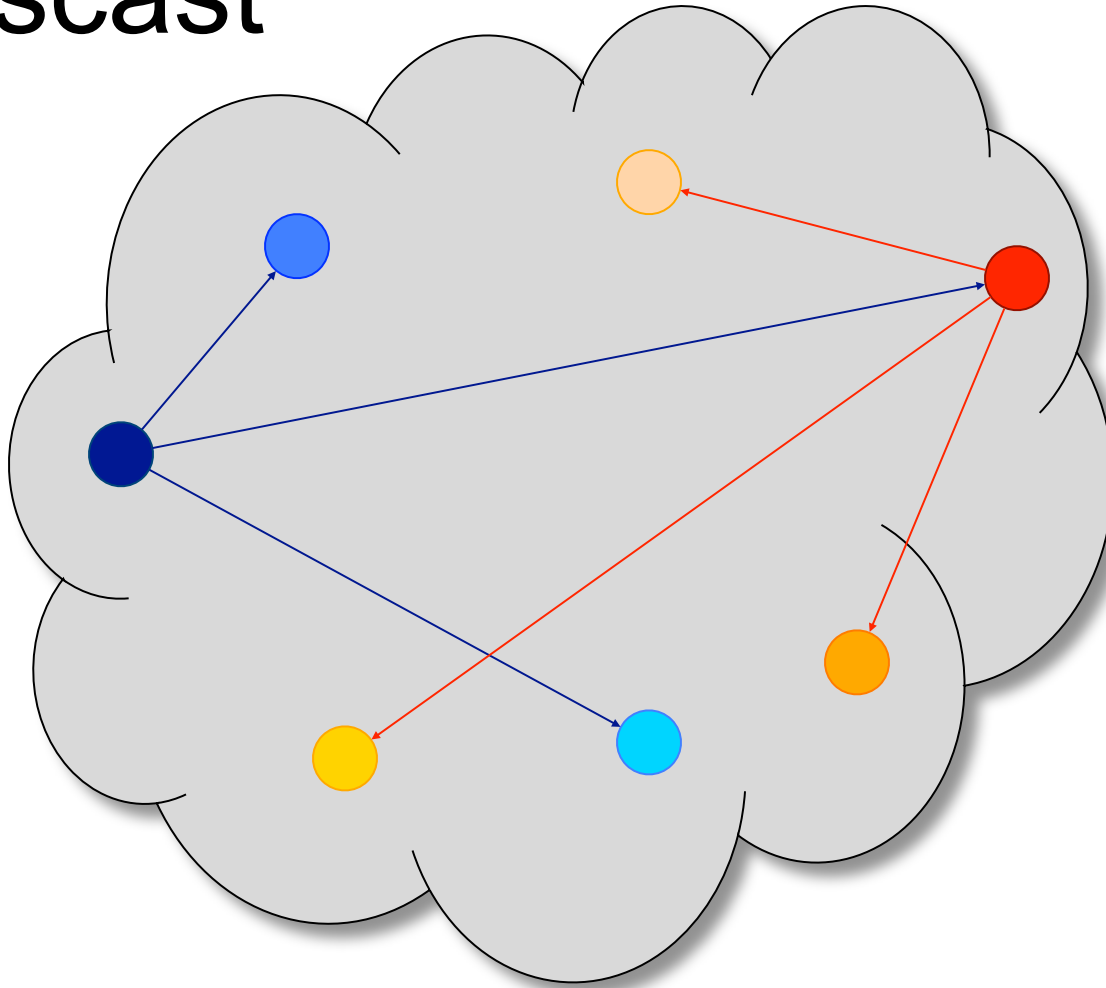





ID & Address	Time stamp
	7
	10
	14

1. Pick random peer from my view

Newscast




ID & Address	Time stamp
	9
	12
	16

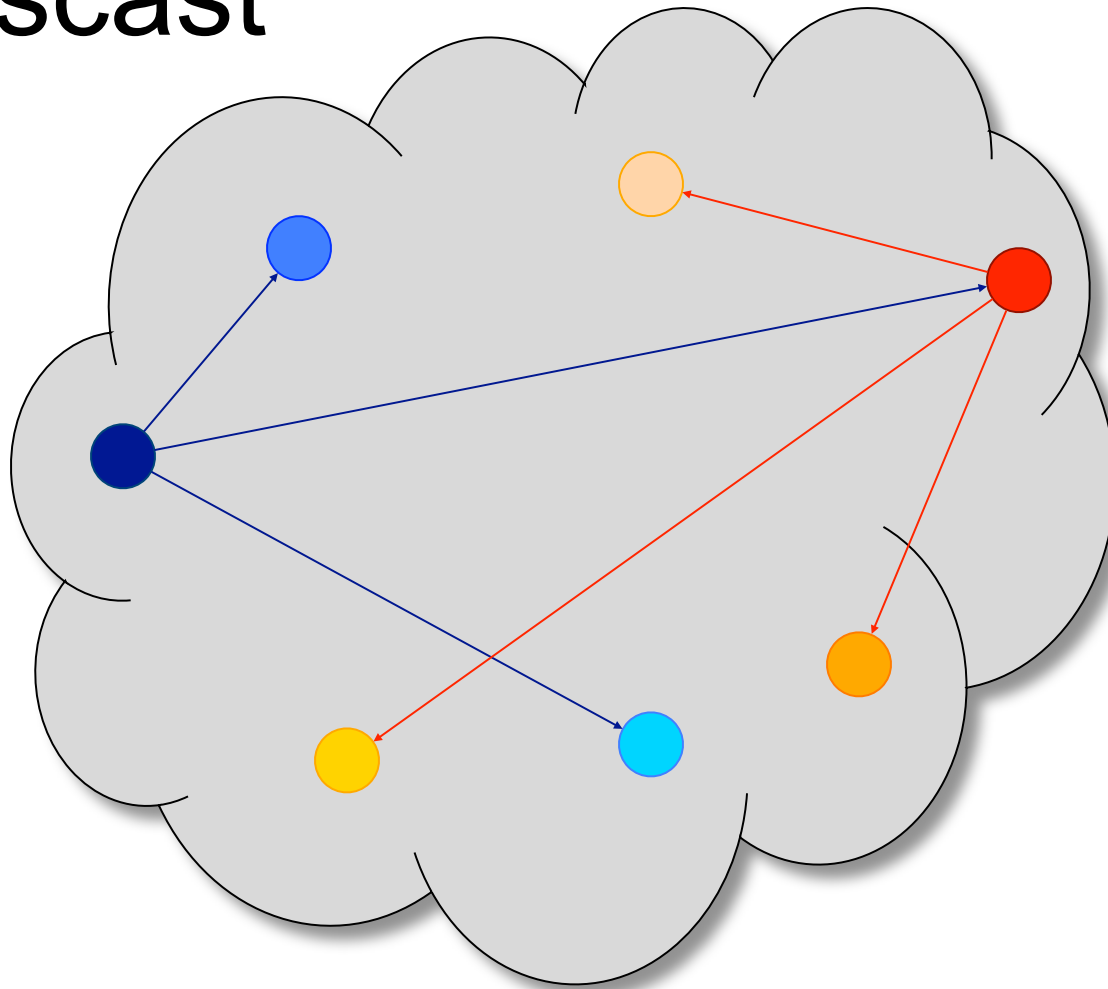





ID & Address	Time stamp
	7
	10
	14




1. Pick random peer from my view
2. Send each other view + own fresh link


Newscast

ID & Address	Time stamp
	9
	12
	16






ID & Address	Time stamp
	7
	10
	14




	9
	12
	16


	20
--	----

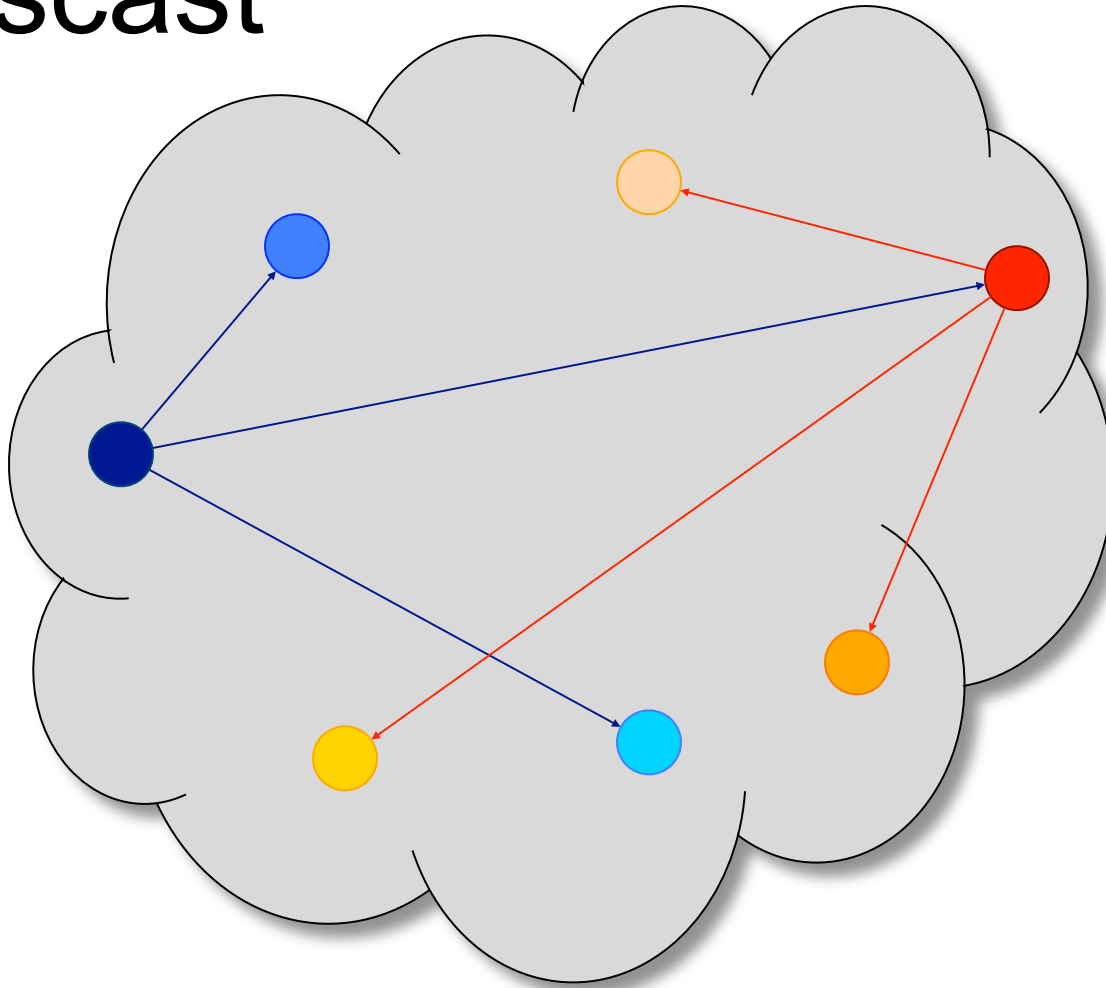
1. Pick random peer from my view
2. Send each other view + own fresh link




Newscast




ID & Address	Time stamp
	9
	12
	16


	7
	10
	14

	20
---	----






ID & Address	Time stamp
	7
	10
	14




	9
	12
	16


	20
--	----

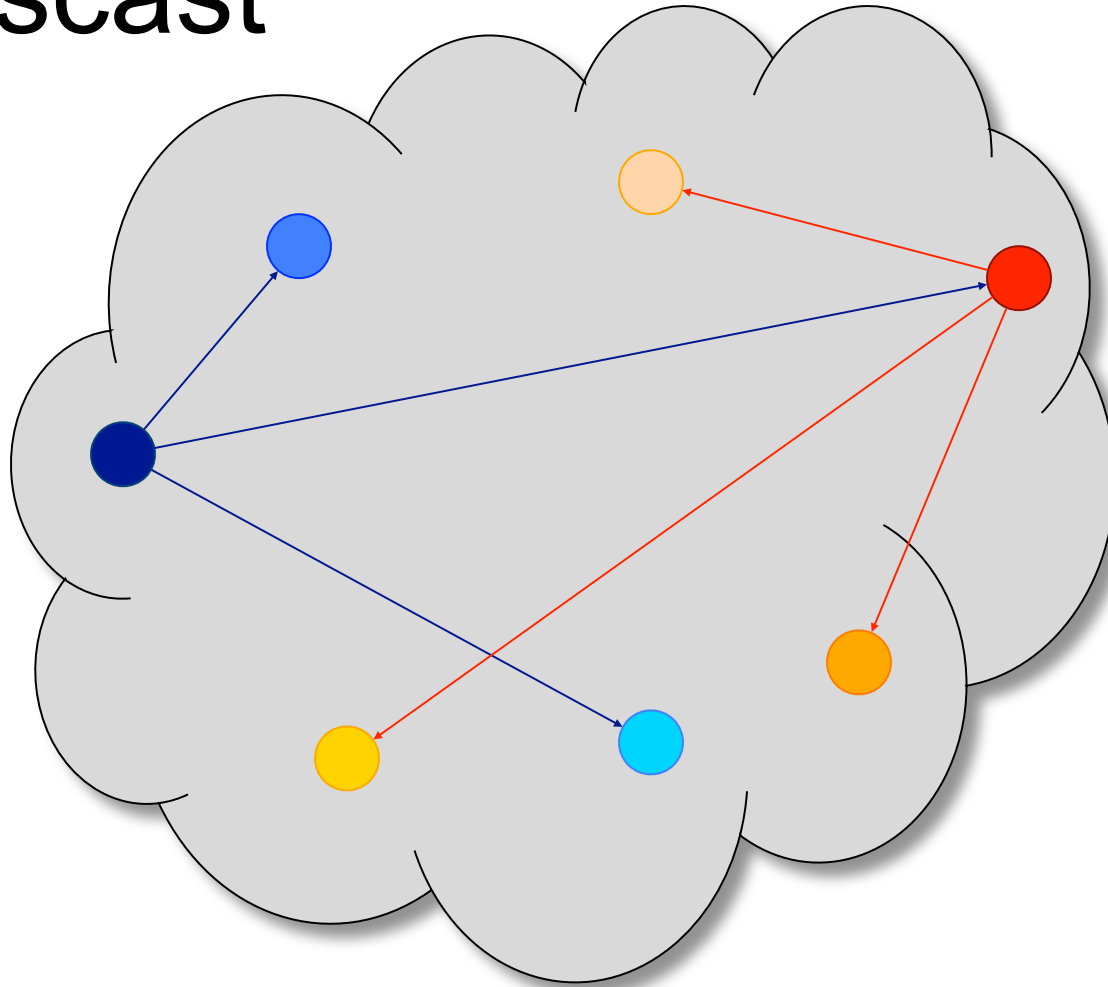
1. Pick random peer from my view
2. Send each other view + own fresh link




Newscast




ID & Address	Time stamp
	9
	12
	16


	7
	10
	14

	20
---	----



ID & Address	Time stamp
	7
	10
	14

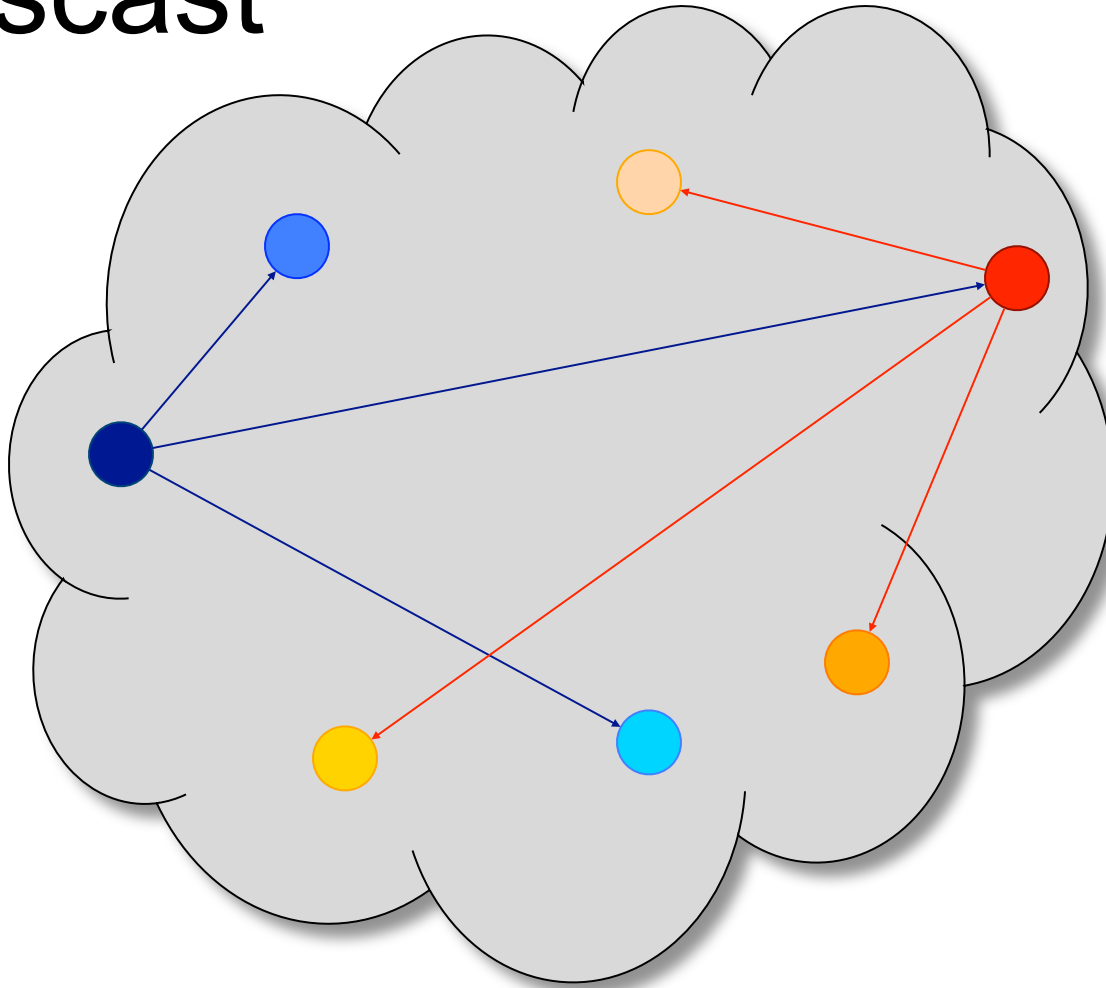
	9
	12
	16

	20
--	----

1. Pick random peer from my view
2. Send each other view + own fresh link
3. Keep c freshest links (remove own info, duplicates)

Newscast




ID & Address	Time stamp
Dark Blue	9
Dark Red	12
Cyan	16
Brown	7
Olive	10
Yellow	14
Red	20

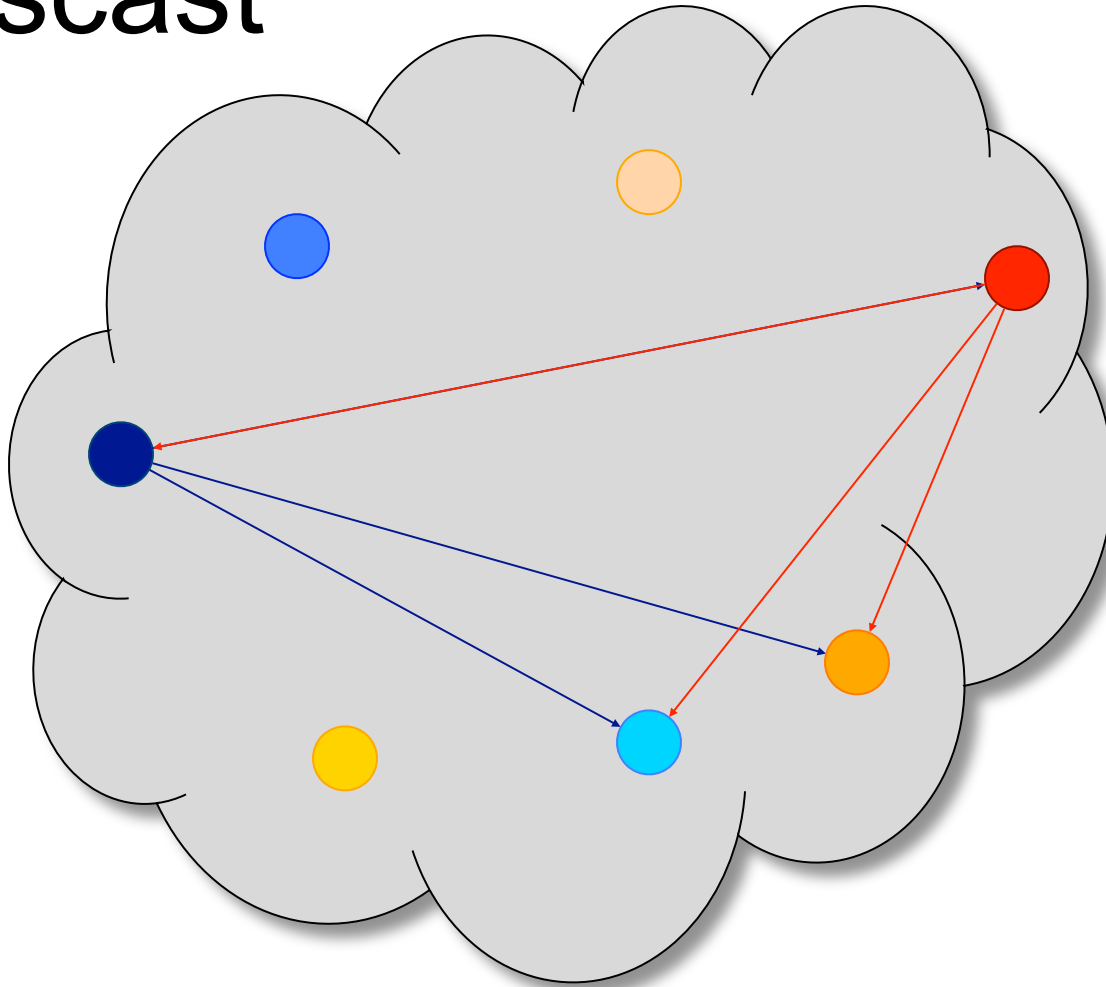





ID & Address	Time stamp
Brown	7
Olive	10
Yellow	14
Dark Blue	9
Dark Red	12
Cyan	16
Dark Blue	20

1. Pick random peer from my view
2. Send each other view + own fresh link
3. Keep c freshest links (remove own info, duplicates)

Newscast

ID & Address	Time stamp
	14
	16
	20



ID & Address	Time stamp
	14
	16
	20

1. Pick random peer from my view
2. Send each other view + own fresh link
3. Keep c freshest link (remove own info, duplicates)



Newscast

Experiments

- 100,000 nodes
- $C = 20$ neighbors per node



Evaluation framework

- Average path length
 - The average of shortest path lengths over all pairs of nodes in the graph
- In epidemic broadcast protocols
 - A measure of the time needed to diffuse information from a node to another

Evaluation framework

■ Clustering coefficient

- The clustering coefficient of a node p is defined as the # of edges between the neighbors of p divided by the # of all possible edges between those neighbors
- Intuitively, indicates the extent to which the neighbors of p know each other.
- The clustering coefficient of the graph is the average of the clustering coefficients of all nodes
- Examples
 - for a complete graph it is 1
 - for a tree it is 0

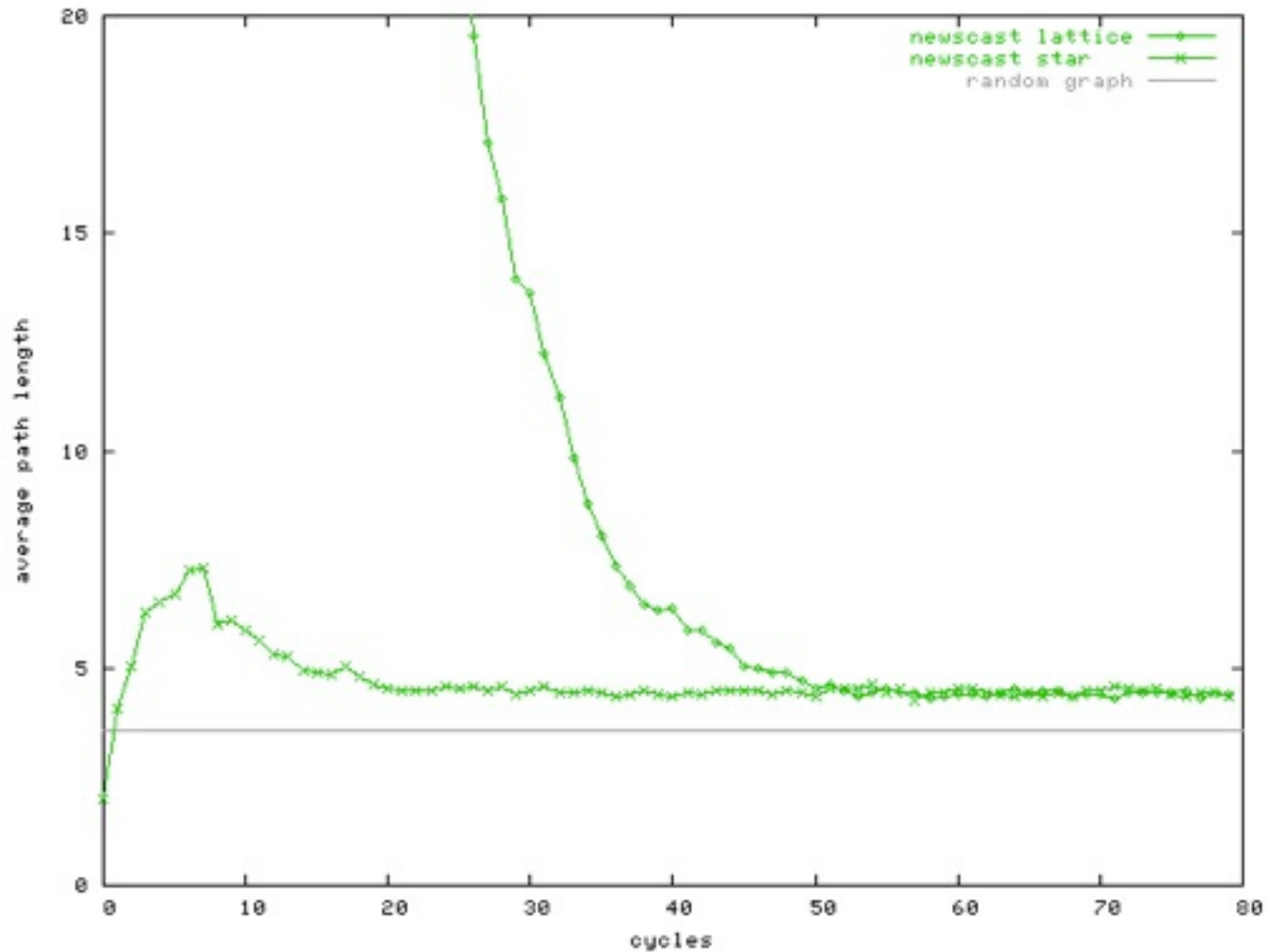


Evaluation framework

- High clustering coefficient
 - Several redundant messages are sent when an epidemic protocol is used

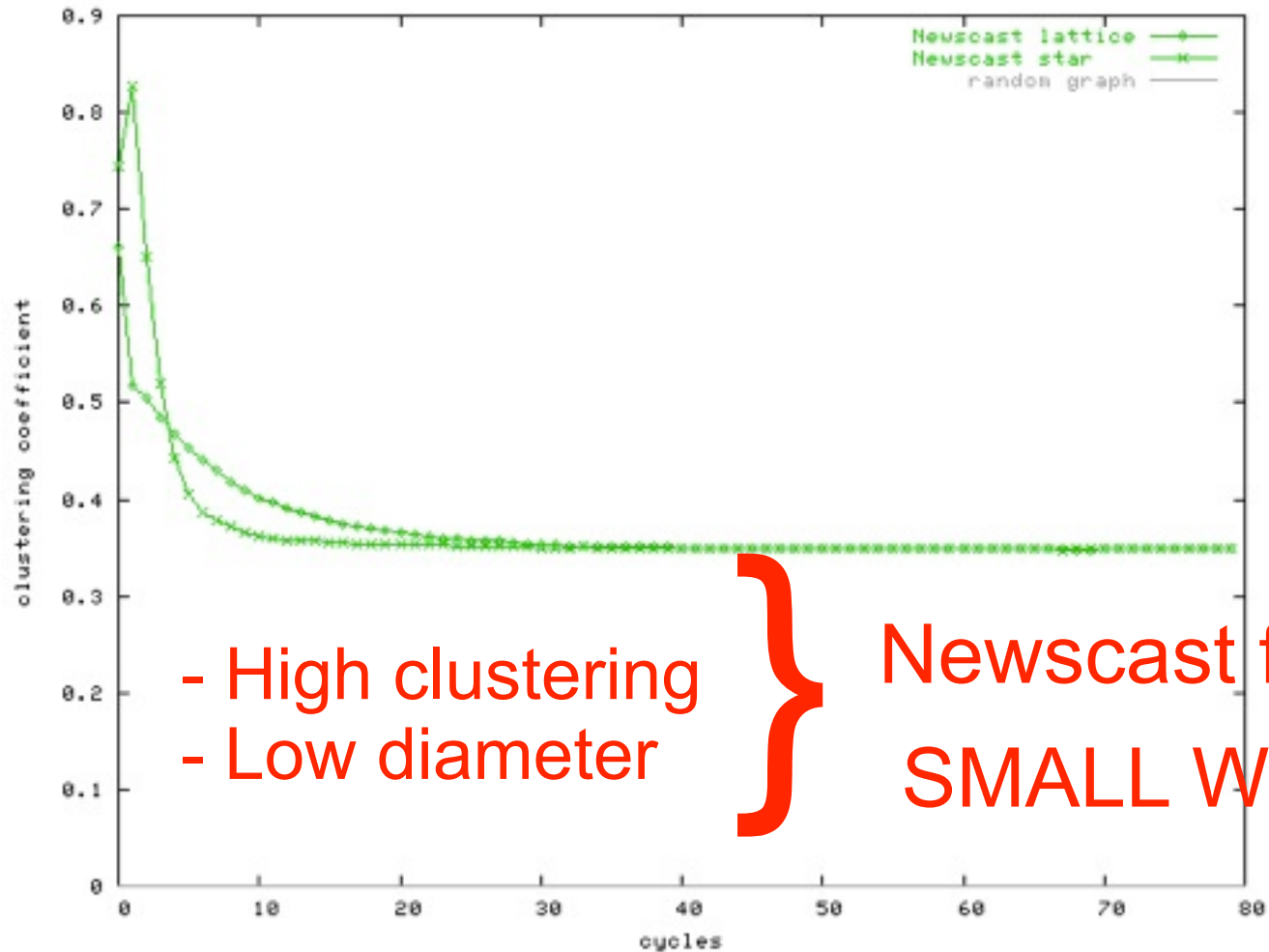
Average path length

- Indication of the *time* and *cost* to flood the network



Clustering

- High clustering is bad for:
 - **Flooding**: It results in many redundant messages
 - **Self-healing**: Strongly connected cluster → weakly connected to the rest of the network



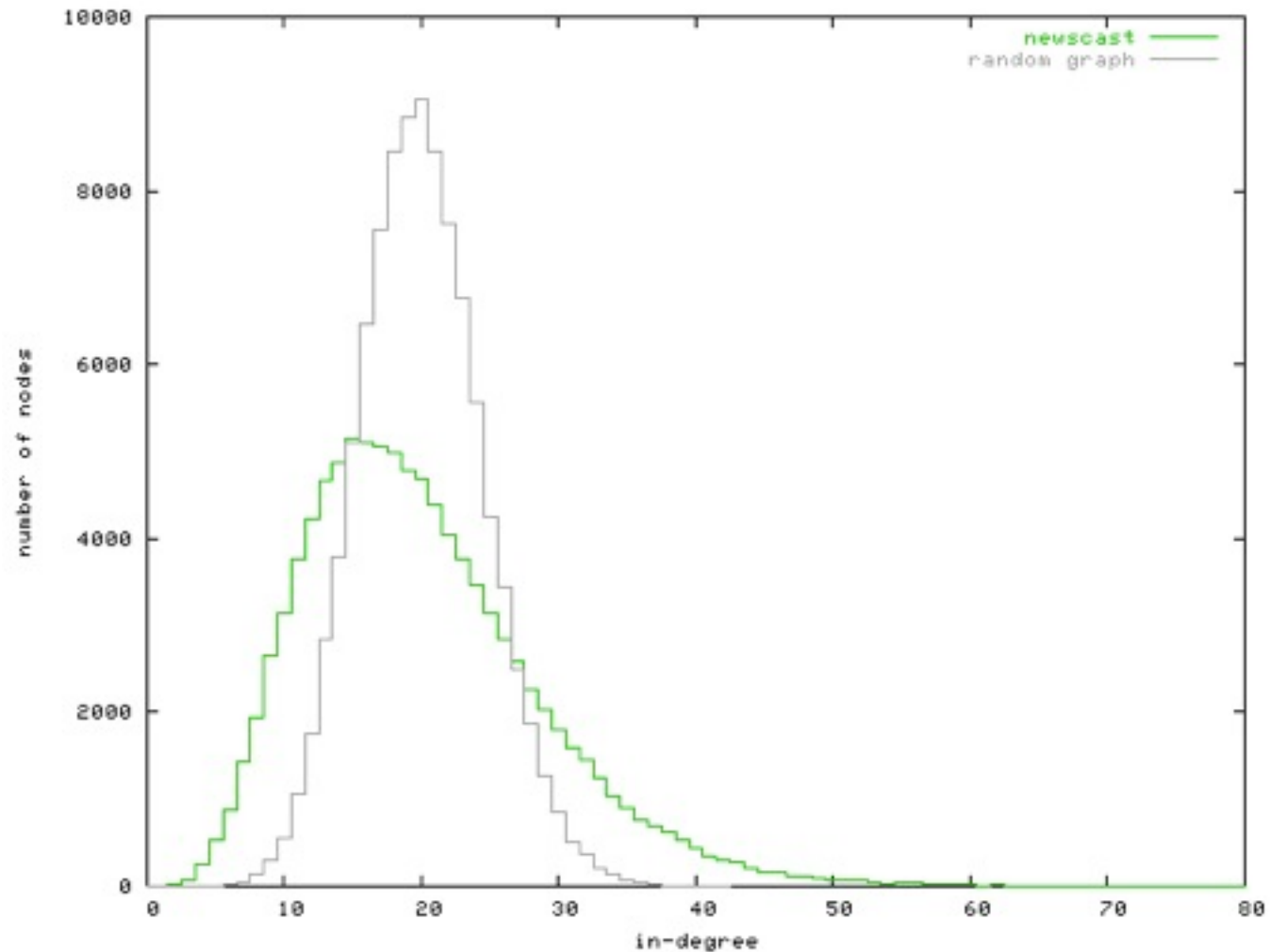
- High clustering
- Low diameter

} Newscast forms a
SMALL WORLD

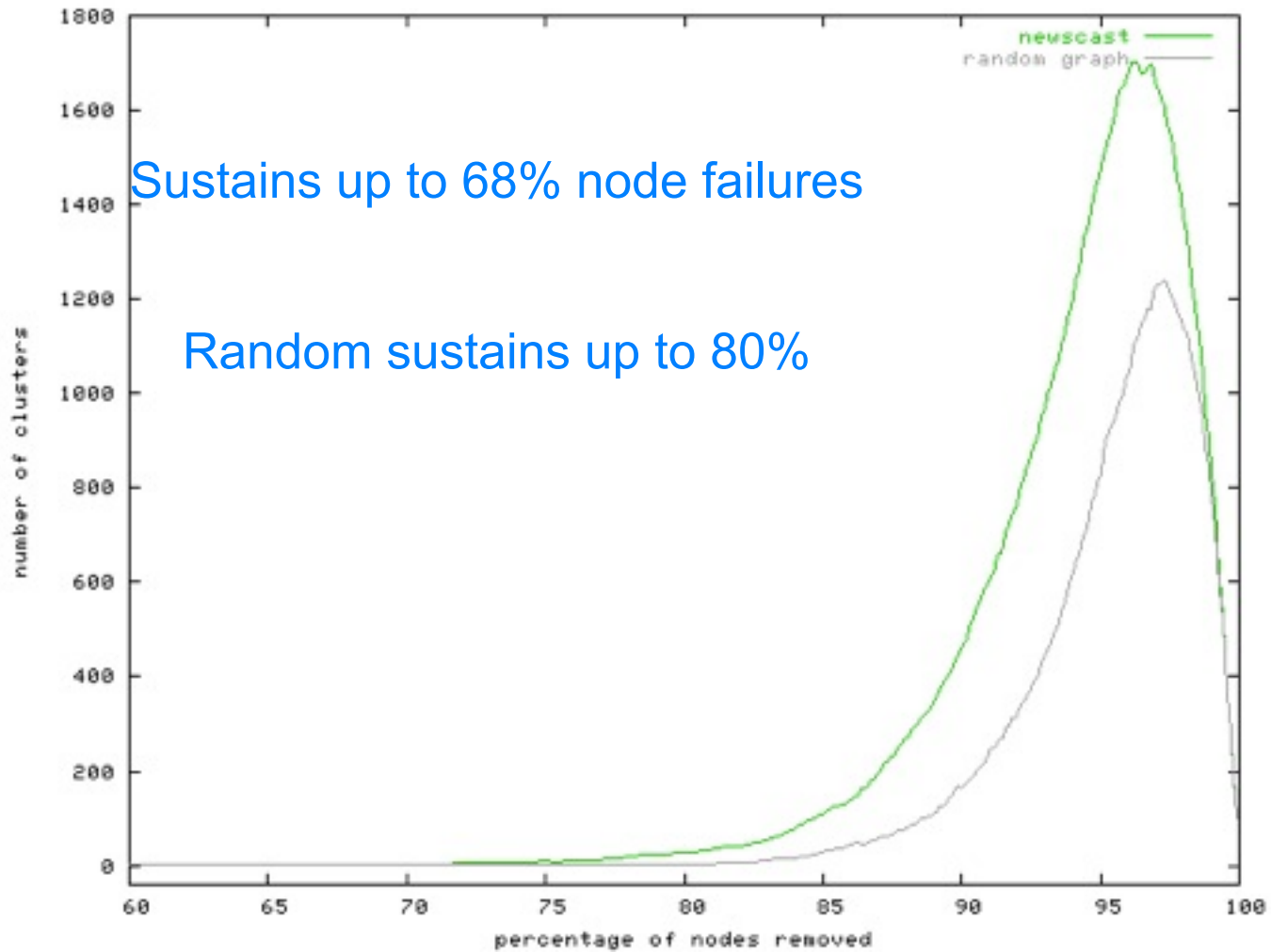
In-Degree Distribution

Affects:

- Robustness
(shows weakly connected nodes)
- Load balancing
- Way epidemics spread



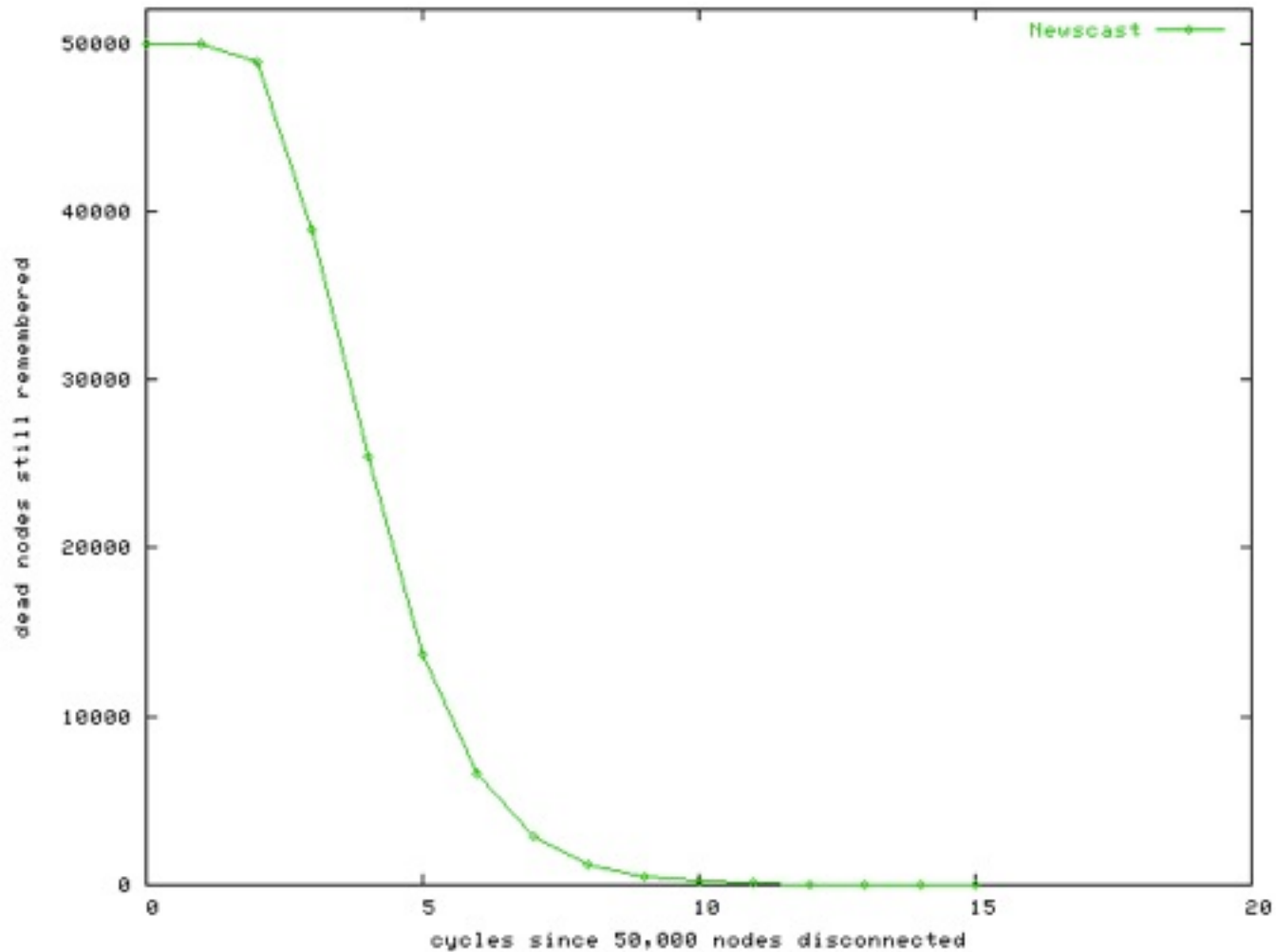
Robustness



Sustains up to 68% node failures

Random sustains up to 80%




Self-healing behaviour

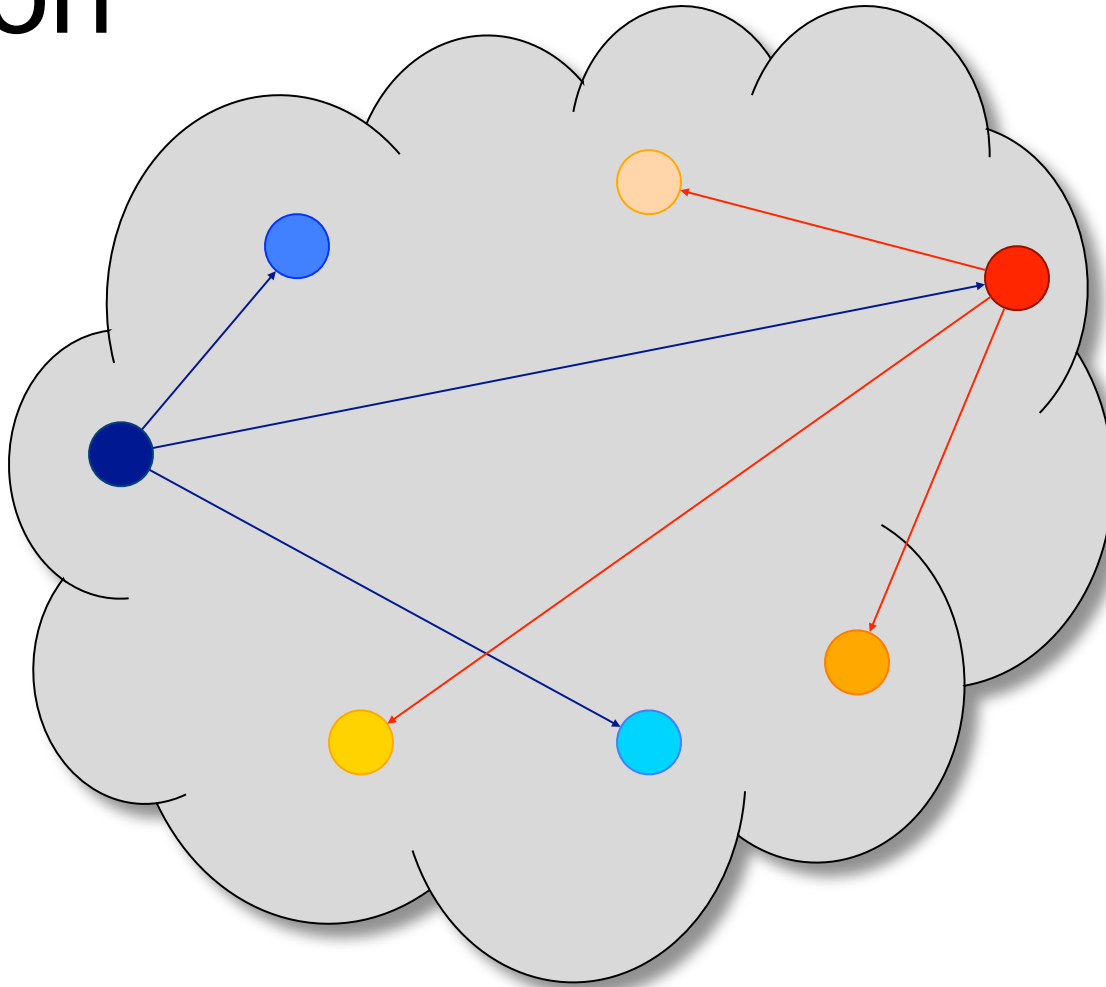





Cyclon

- Descriptor: address + timestamp
- `selectNeighbor()`
 - select the oldest descriptor in the view
- `extract(view, q)`
 - returns a subset of $t-1$ random nodes, plus a fresh local identifier
- `merge(view, msgq)`
 - remove q descriptor
 - discard entries in msg_q : p , nodes already know
 - add msg_q , possibly removing entries sent to q




Cyclon

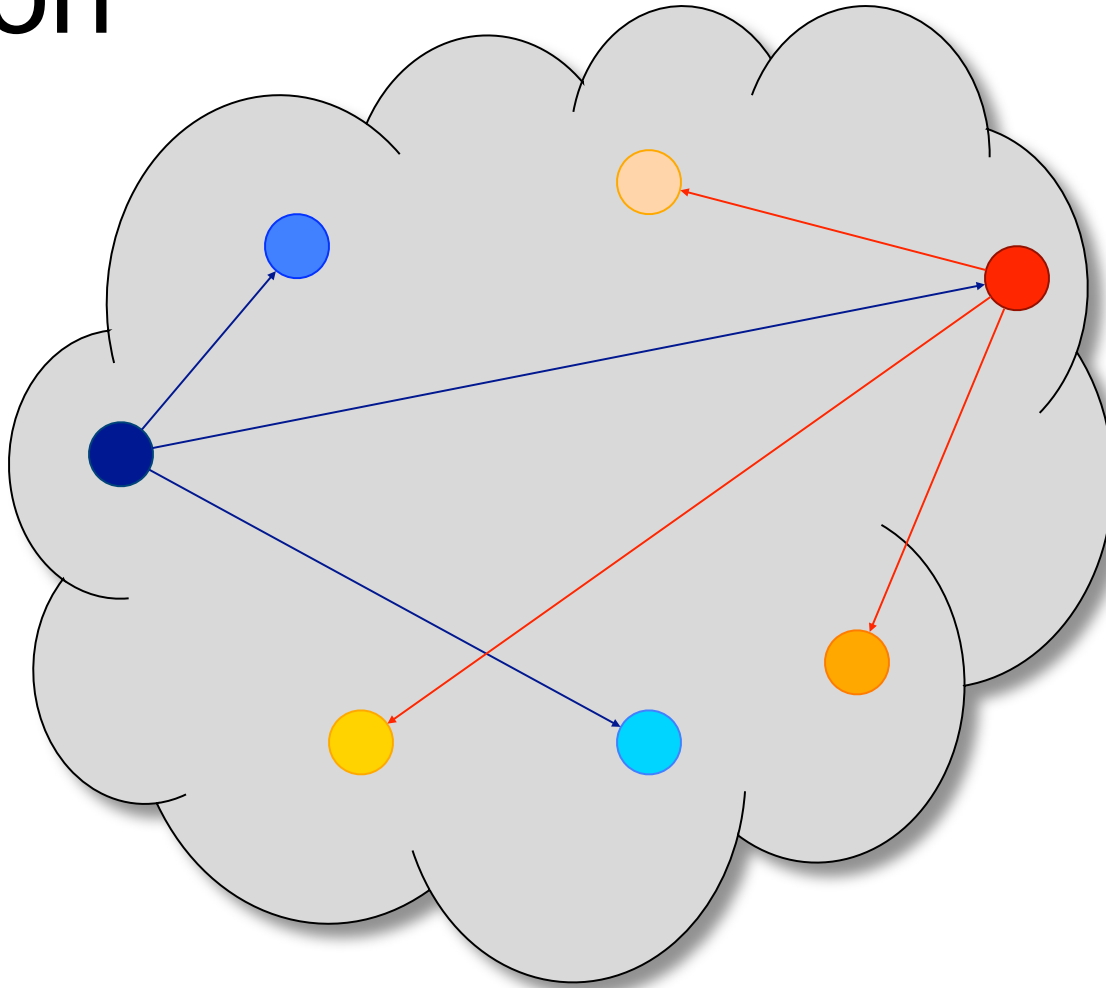
ID & Address	Time stamp
	9
	12
	4






ID & Address	Time stamp
	7
	10
	14

Cyclon

ID & Address	Time stamp
	9
	12
	4

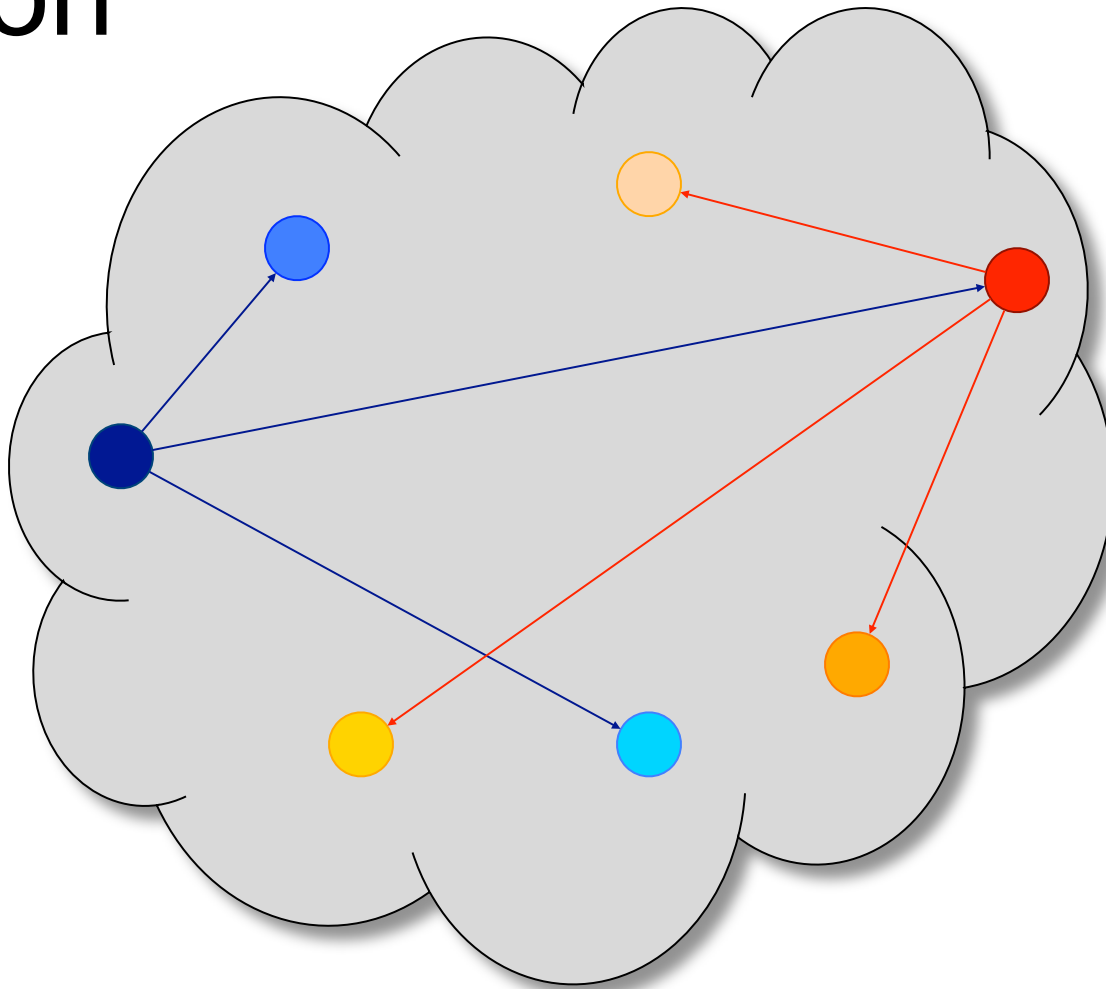


ID & Address	Time stamp
	7
	10
	14

1. Pick oldest peer from my view

Cyclon

ID & Address	Time stamp
Blue	9
Cyan	12
Red	4

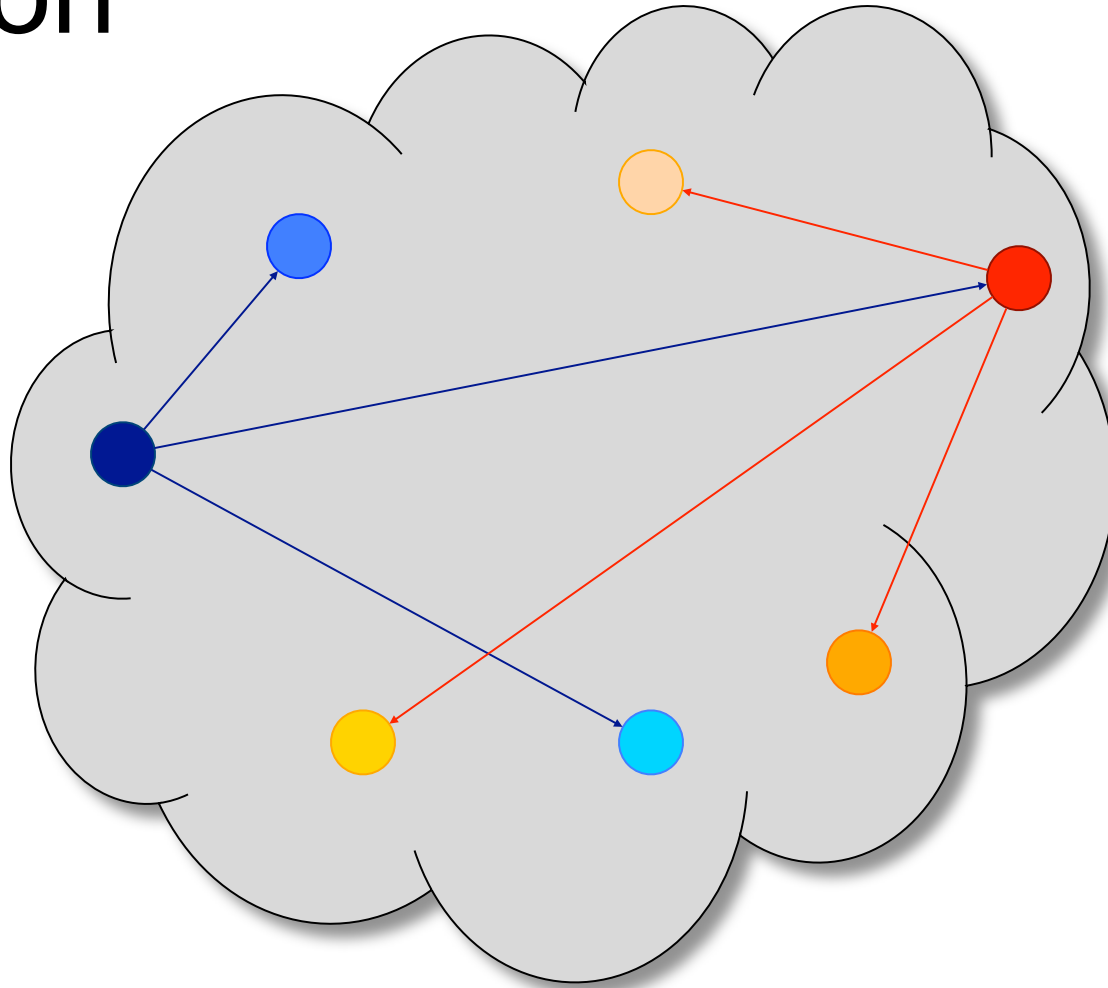


ID & Address	Time stamp
Light Orange	7
Yellow	10
Orange	14

1. Pick oldest peer from my view

Cyclon


ID & Address	Time stamp
Blue	9
Cyan	12
Red	4




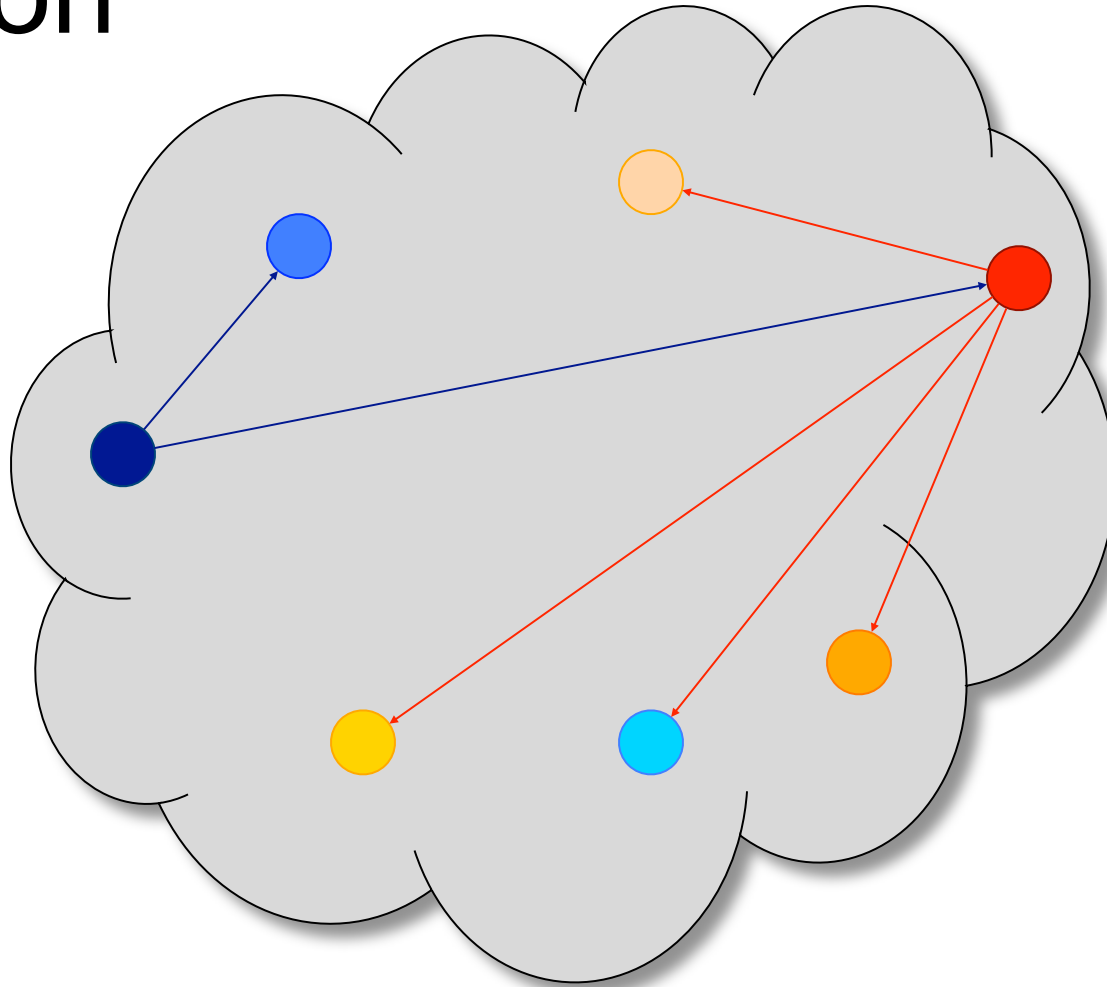
ID & Address	Time stamp
Light Orange	7
Yellow	10
Orange	14





1. Pick oldest peer from my view
2. Exchange some neighbors (the pointers)

Cyclon

ID & Address	Time stamp
	9


	4
--	---

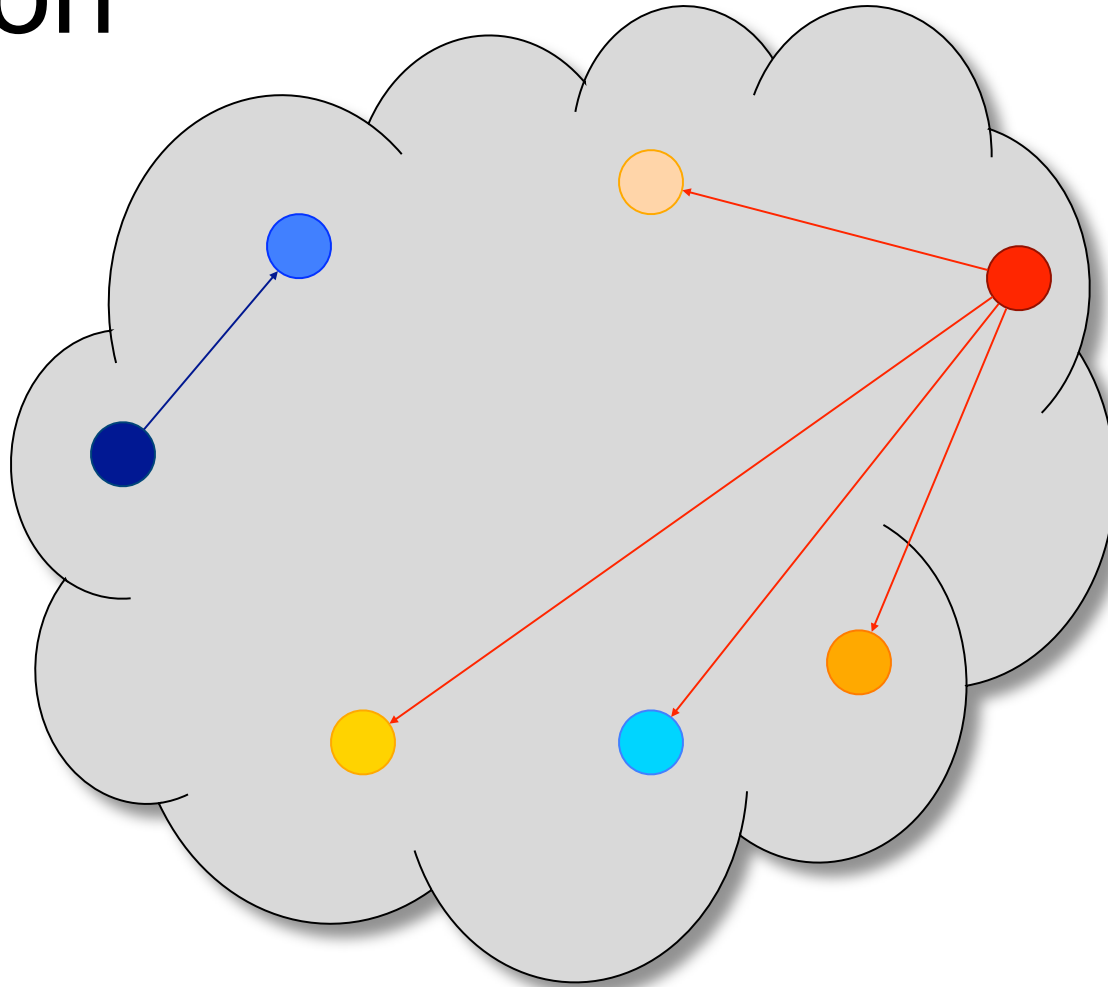







ID & Address	Time stamp
	7
	10
	14
	12

1. Pick oldest peer from my view
2. Exchange some neighbors (the pointers)

Cyclon


ID & Address	Time stamp
	9

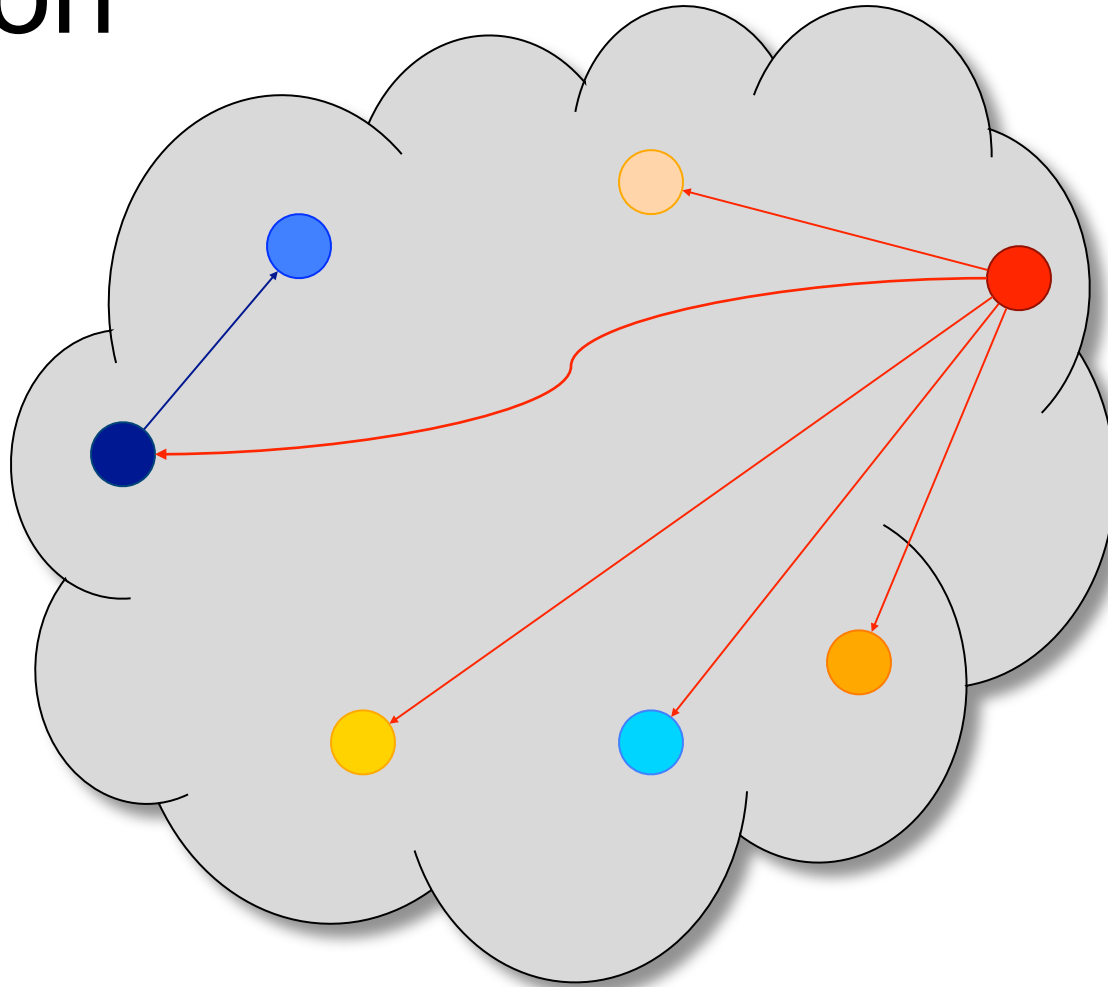







ID & Address	Time stamp
	7
	10
	14
	12
	4

1. Pick oldest peer from my view
2. Exchange some neighbors (the pointers)

Cyclon



ID & Address	Time stamp
	9

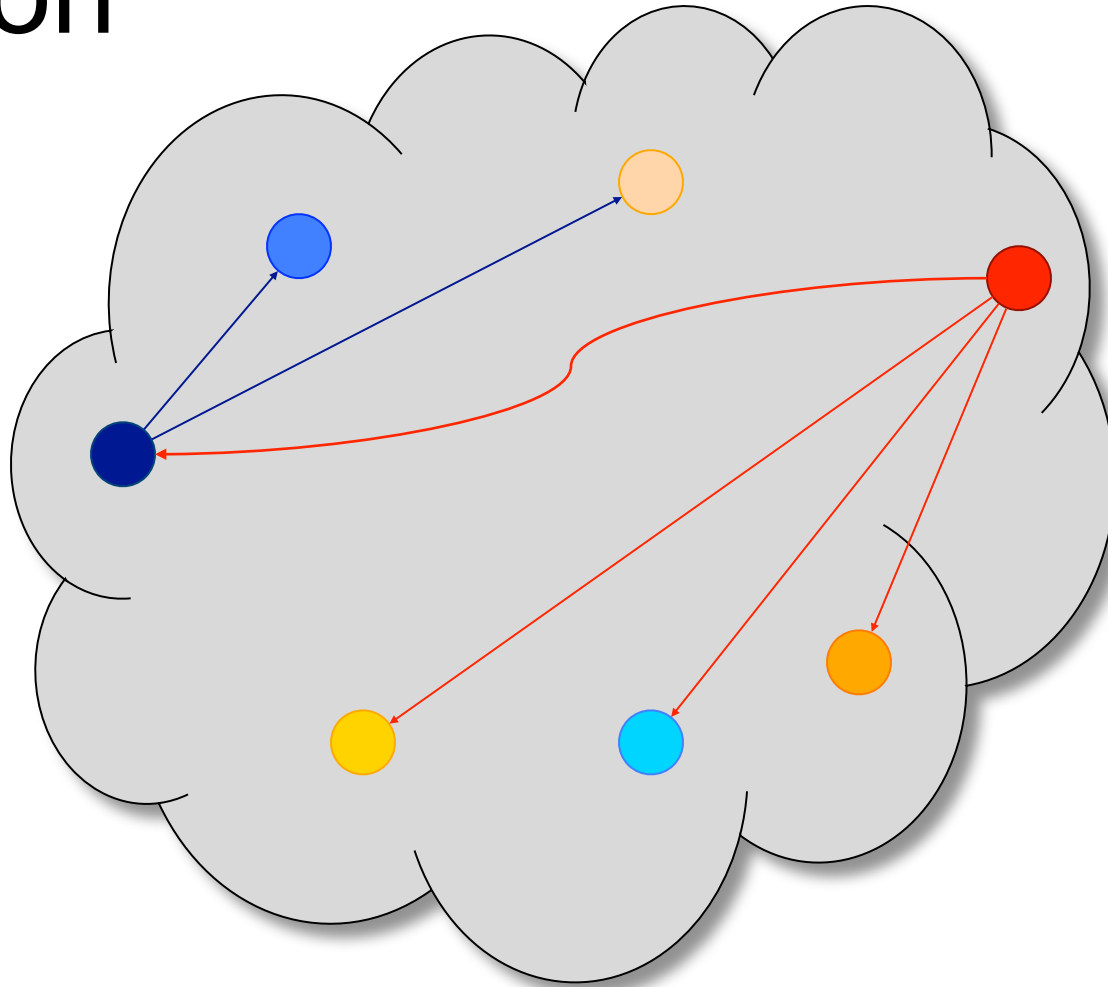


ID & Address	Time stamp
	7
	10
	14
	12
	20





1. Pick oldest peer from my view
2. Exchange some neighbors (the pointers)

Cyclon

ID & Address	Time stamp
	9
	7






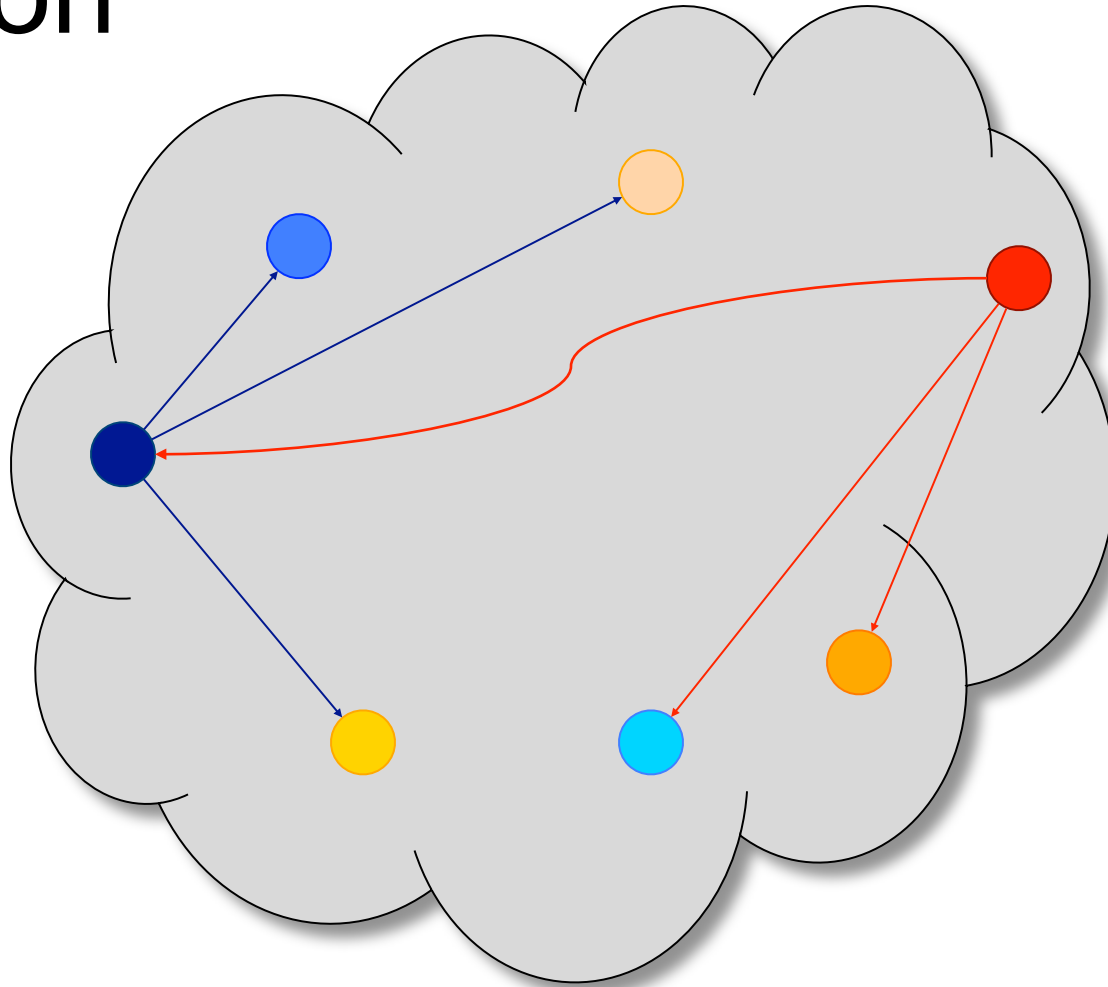
ID & Address	Time stamp
--------------	------------

	10
	14
	12
	20

1. Pick oldest peer from my view
2. Exchange some neighbors (the pointers)


Cyclon

ID & Address	Time stamp
	9
	7
	10



ID & Address	Time stamp
	14




	14
---	----

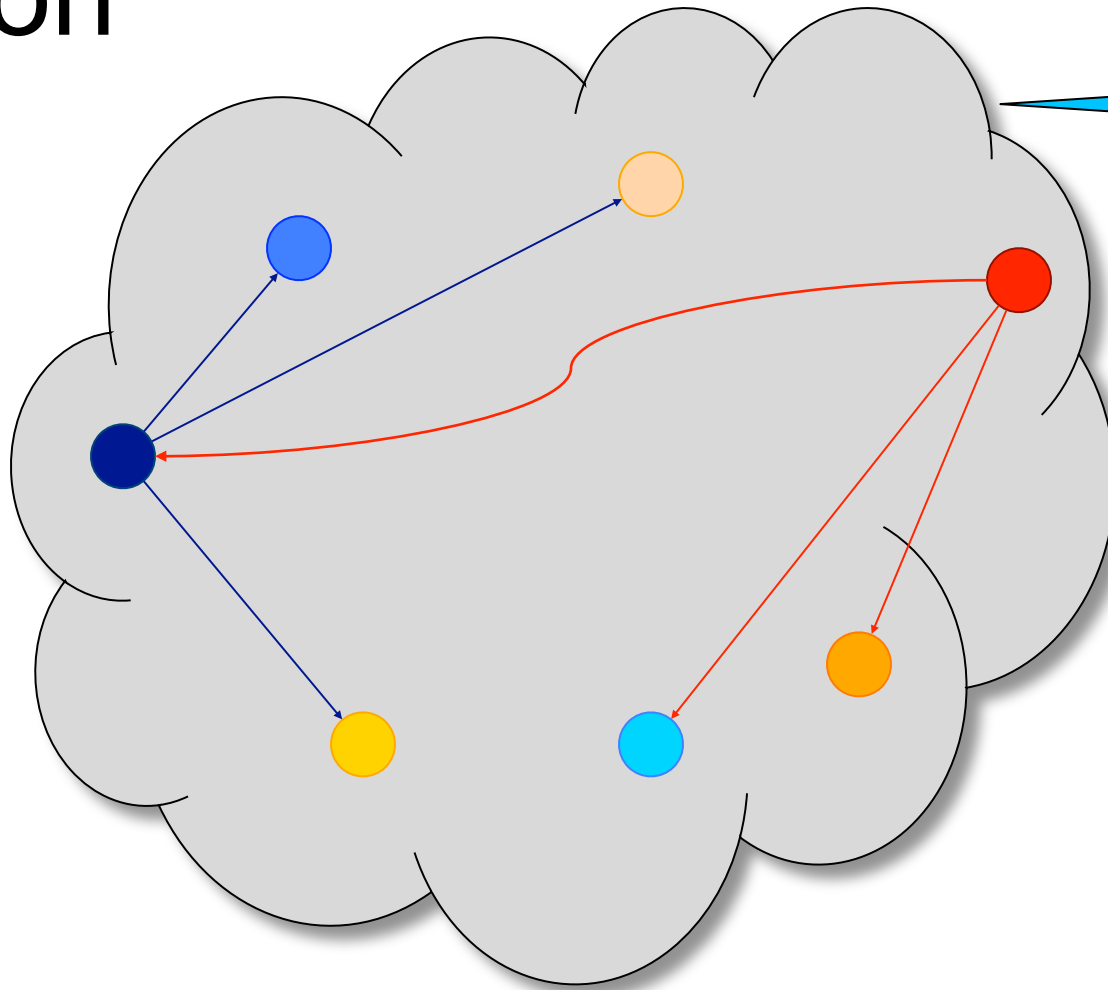
	12
---	----

	20
---	----

1. Pick oldest peer from my view
2. Exchange some neighbors (the pointers)

Cyclon


ID & Address	Time stamp
	9
	7
	10




Guaranteed connectivity

ID & Address	Time stamp
	

	14
---	----

	12
---	----

	20
---	----

1. Pick oldest peer from my view
2. Exchange some neighbors (the pointers)

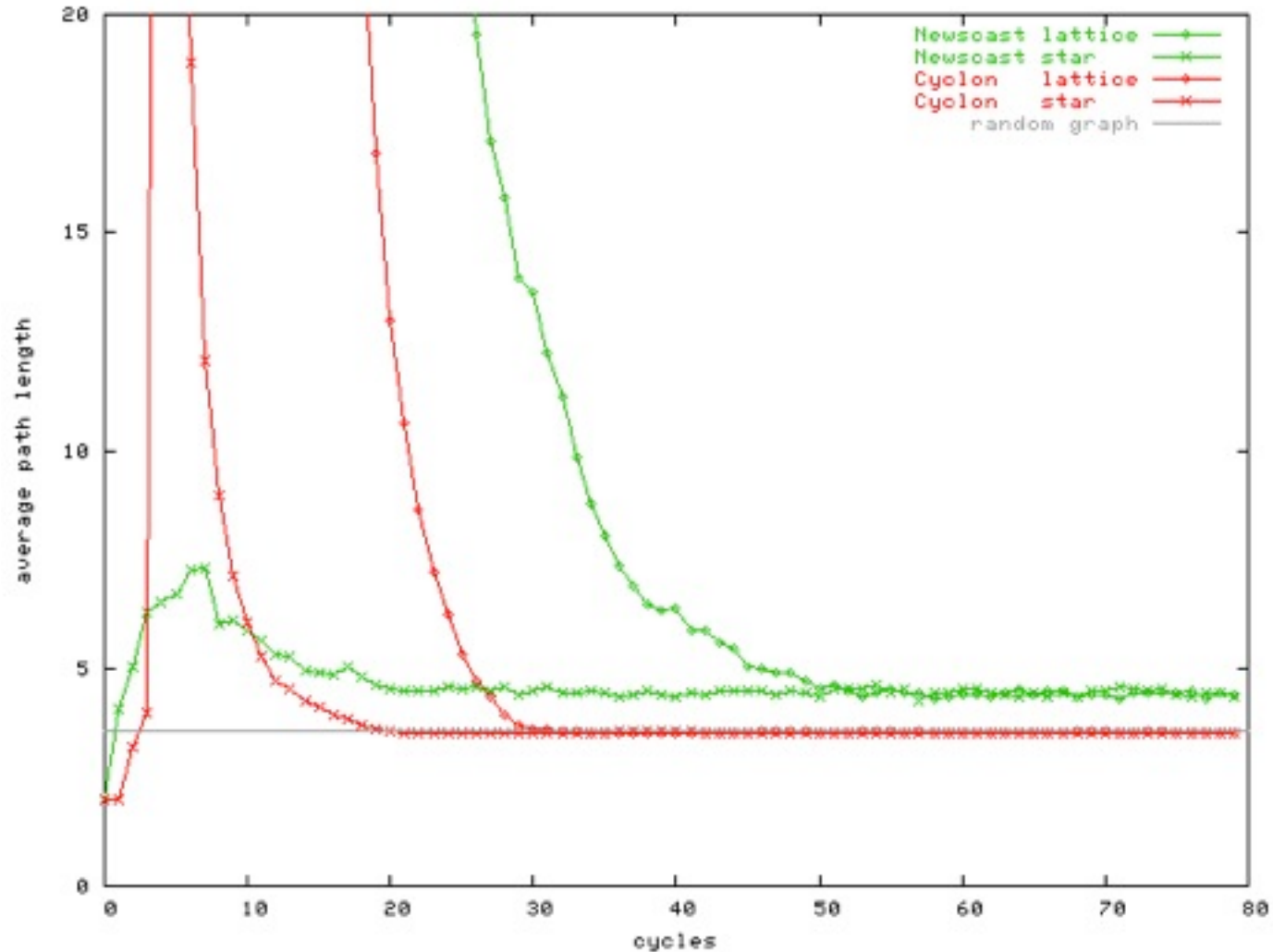


Obvious advantages of Cyclon

- Connectivity is guaranteed
- Uses less bandwidth
 - Only small part of the view is sent

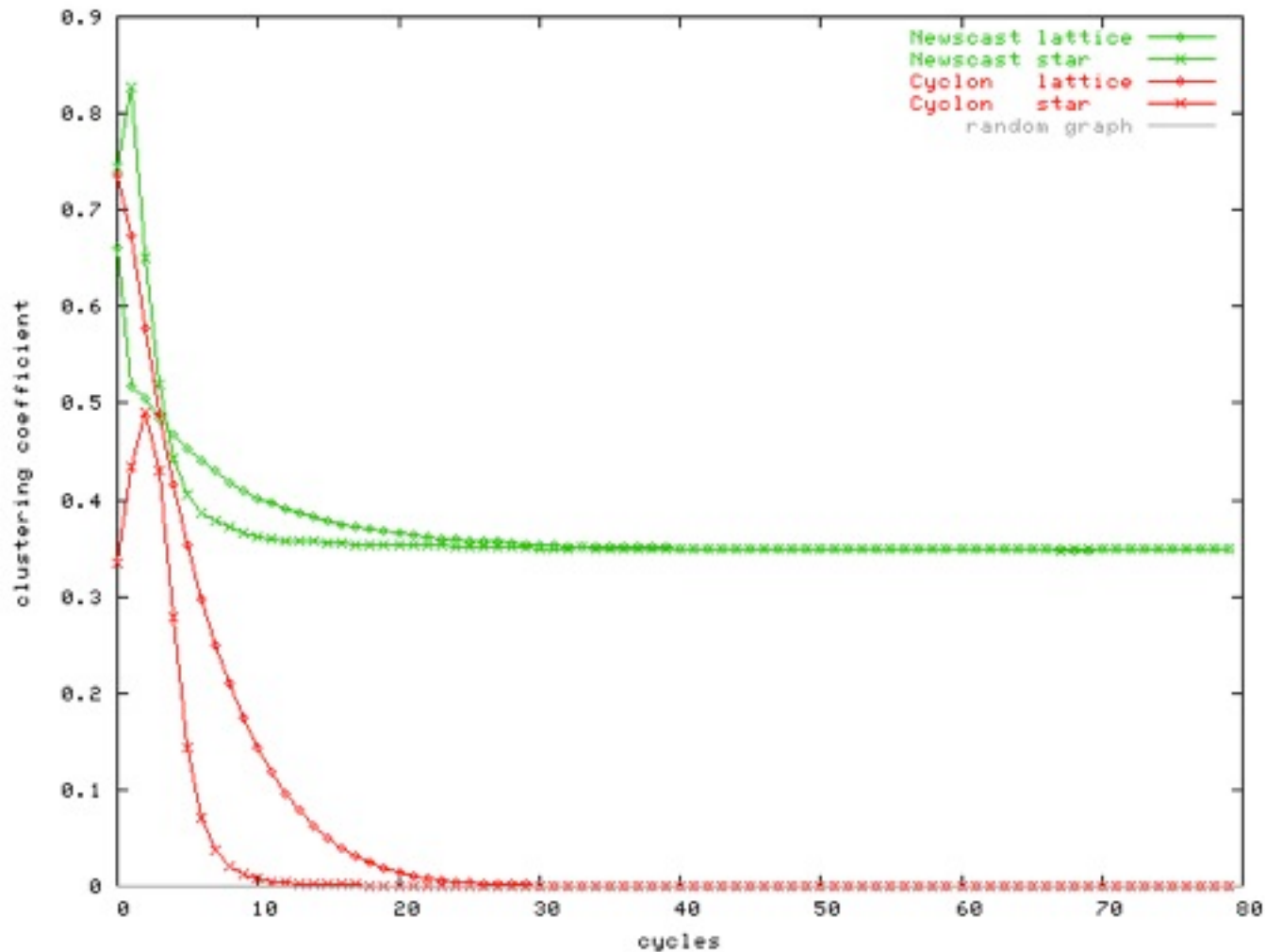
Average path length

- Indication of the *time* and *cost* to flood the network



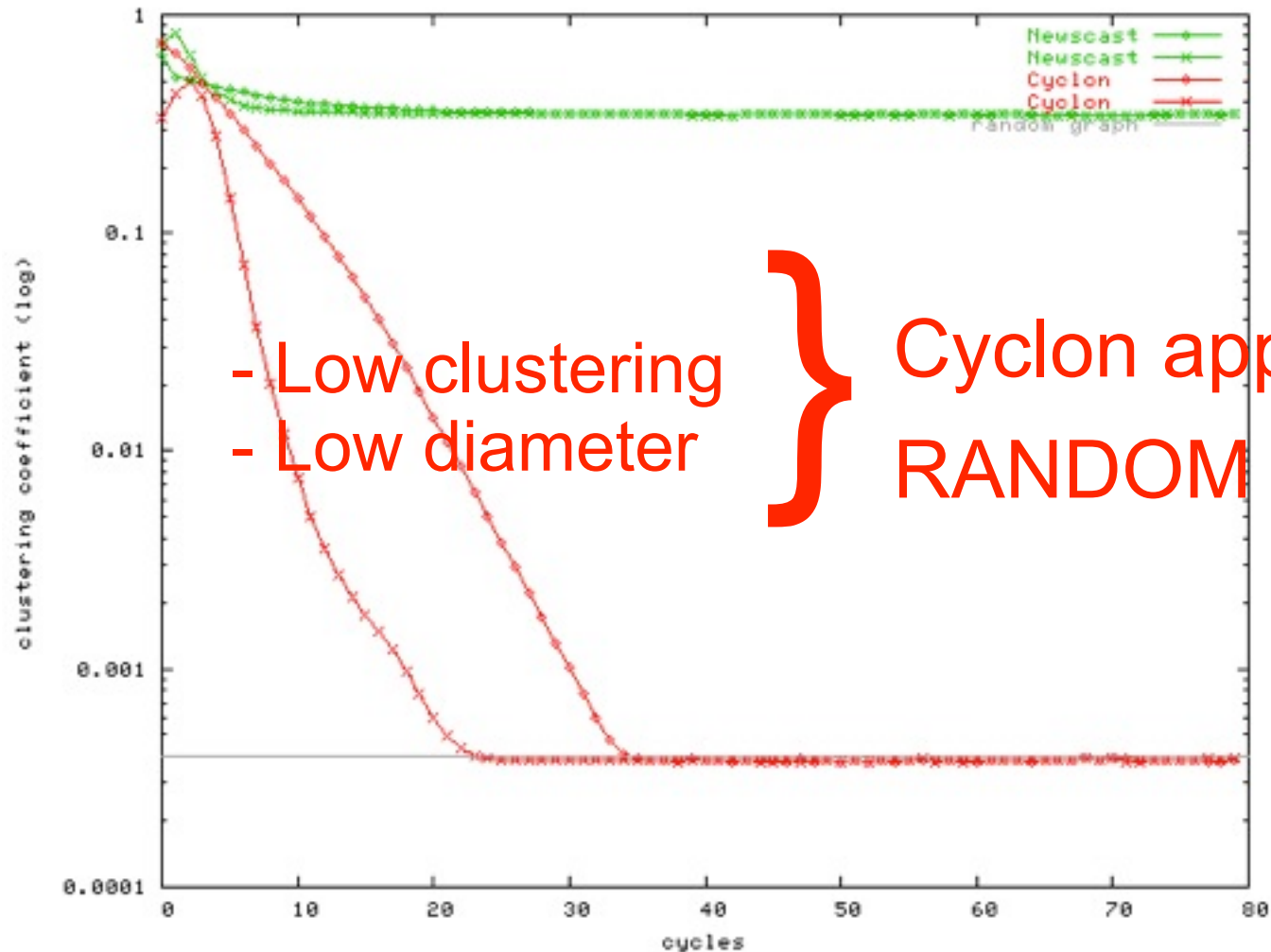
Clustering

- High clustering is bad for:
 - **Flooding**: It results in many redundant messages
 - **Self-healing**: Strongly connected cluster → weakly connected to the rest of the network



Clustering (log scale)

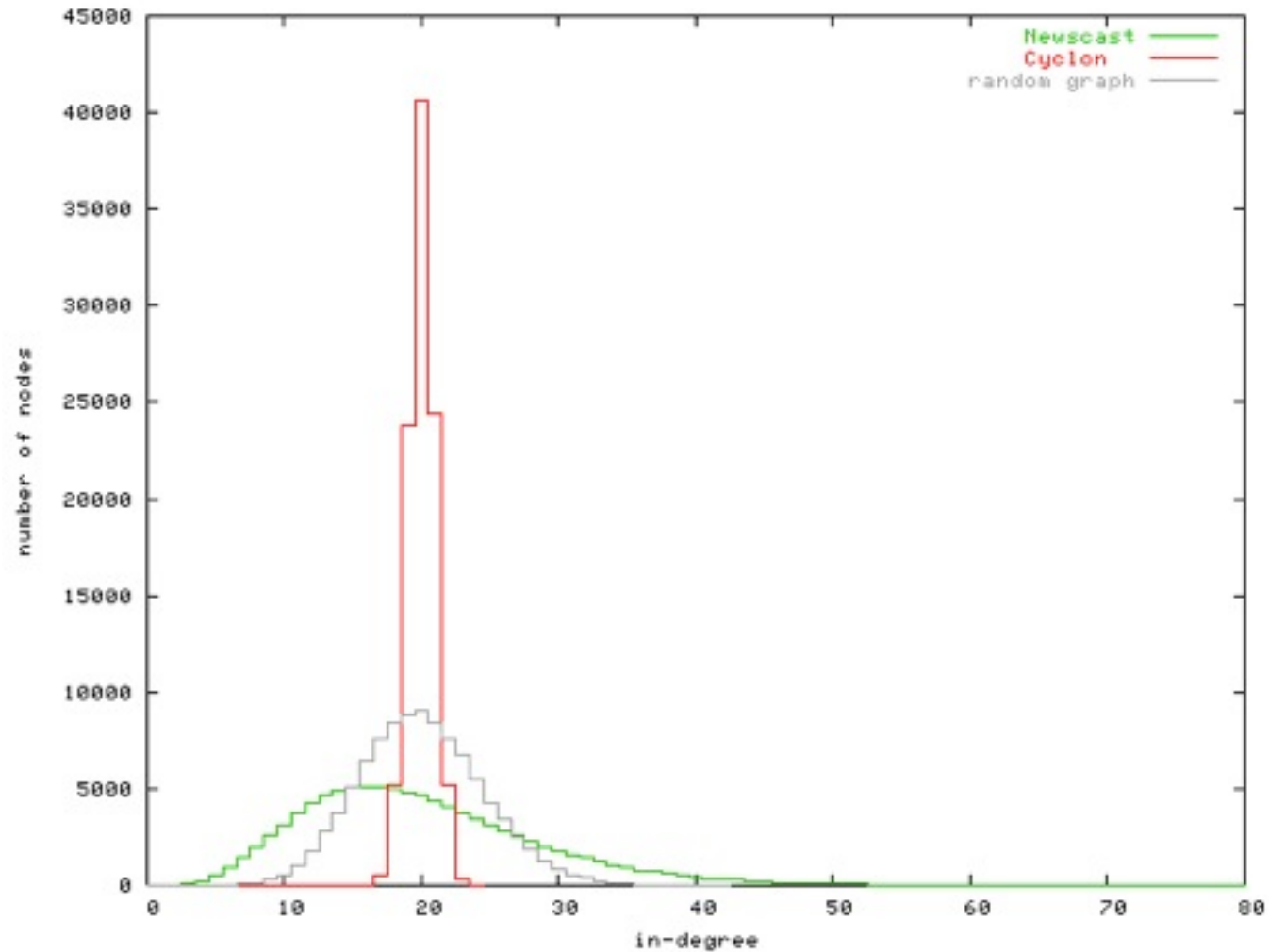
- High clustering is bad for:
 - **Flooding**: It results in many redundant messages
 - **Self-healing**: Strongly connected cluster → weakly connected to the rest of the network



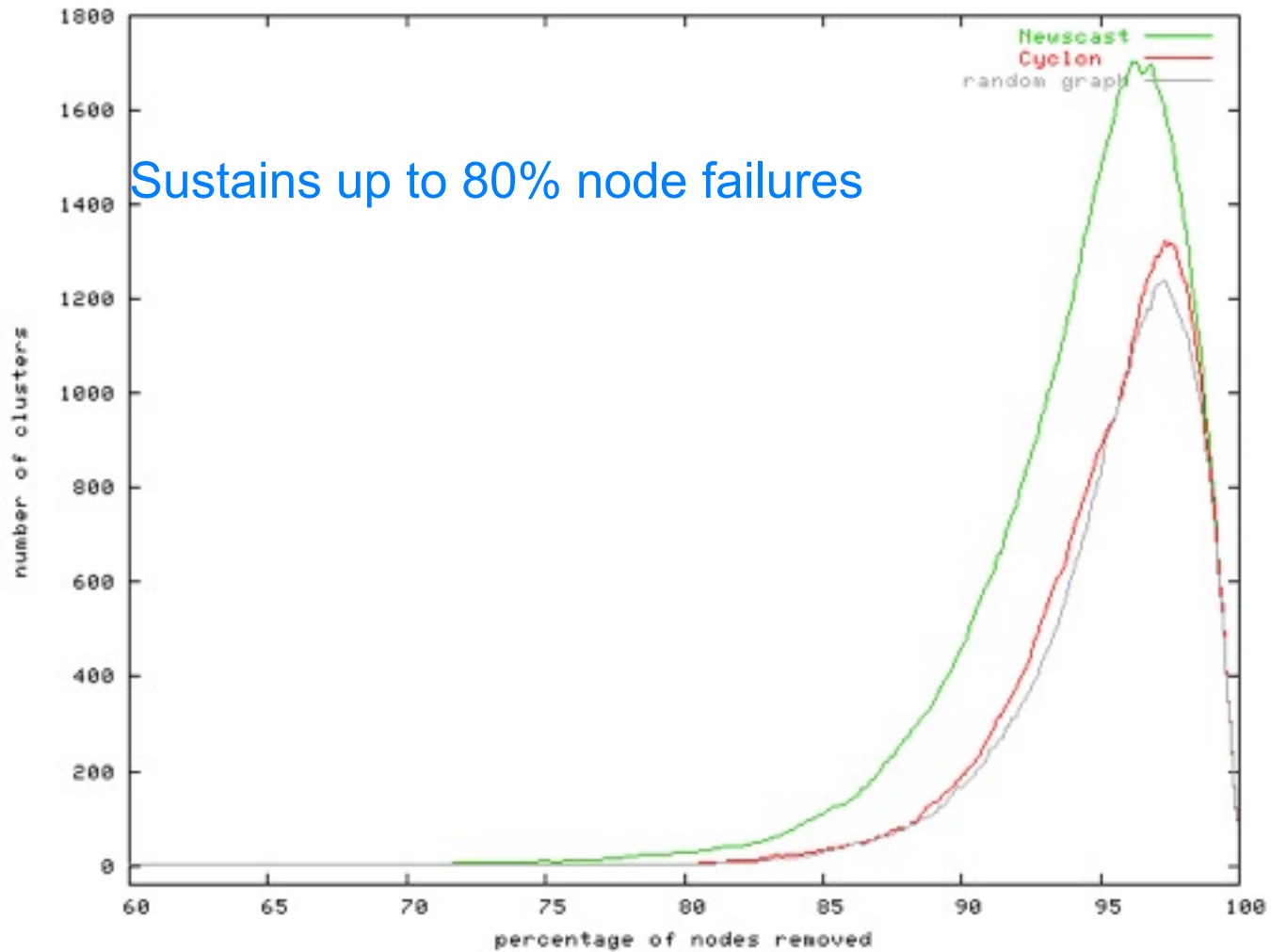
In-Degree Distribution

Affects:

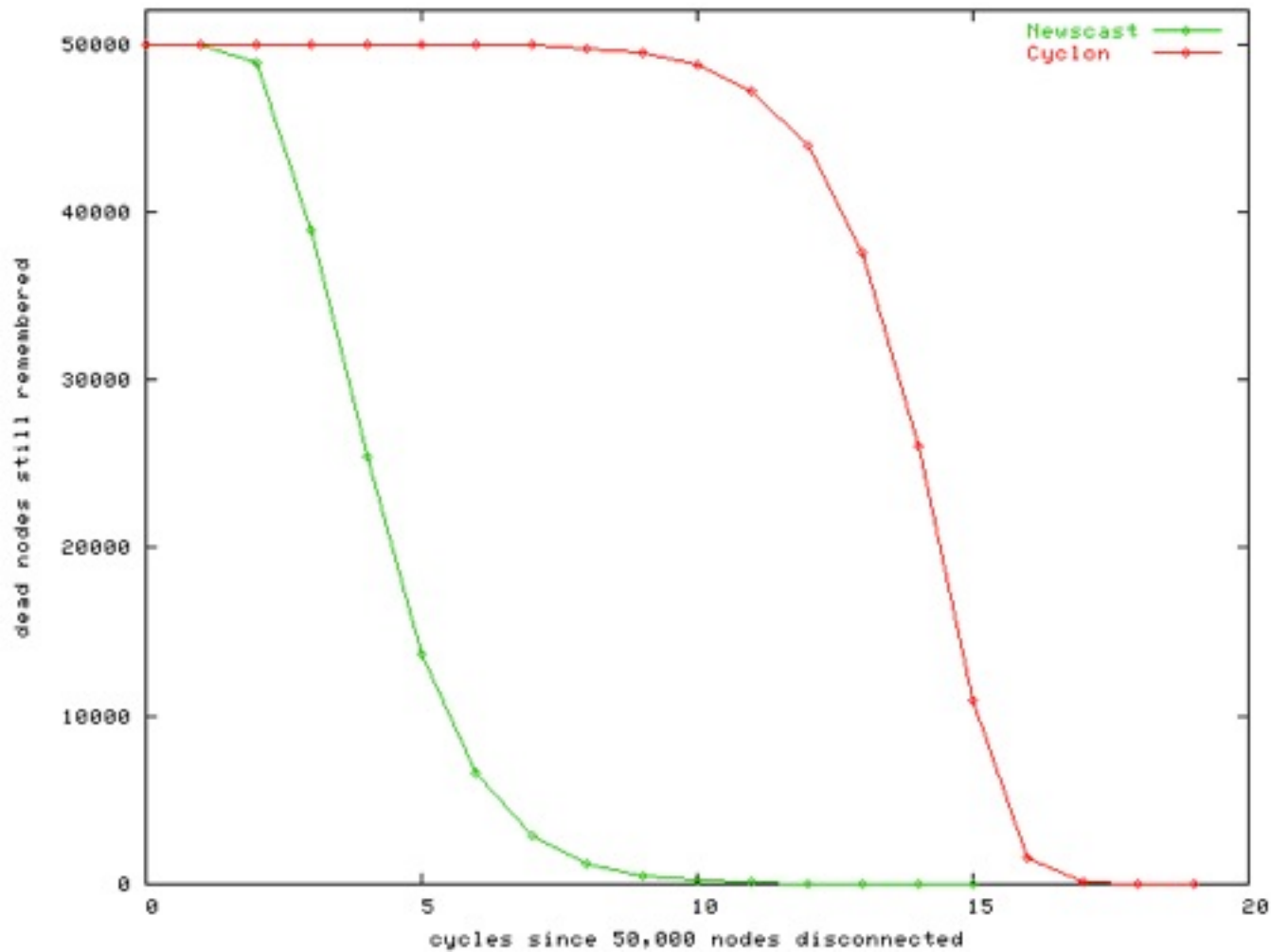
- Robustness (shows weakly connected nodes)
- Load balancing
- Way epidemics spread



Robustness

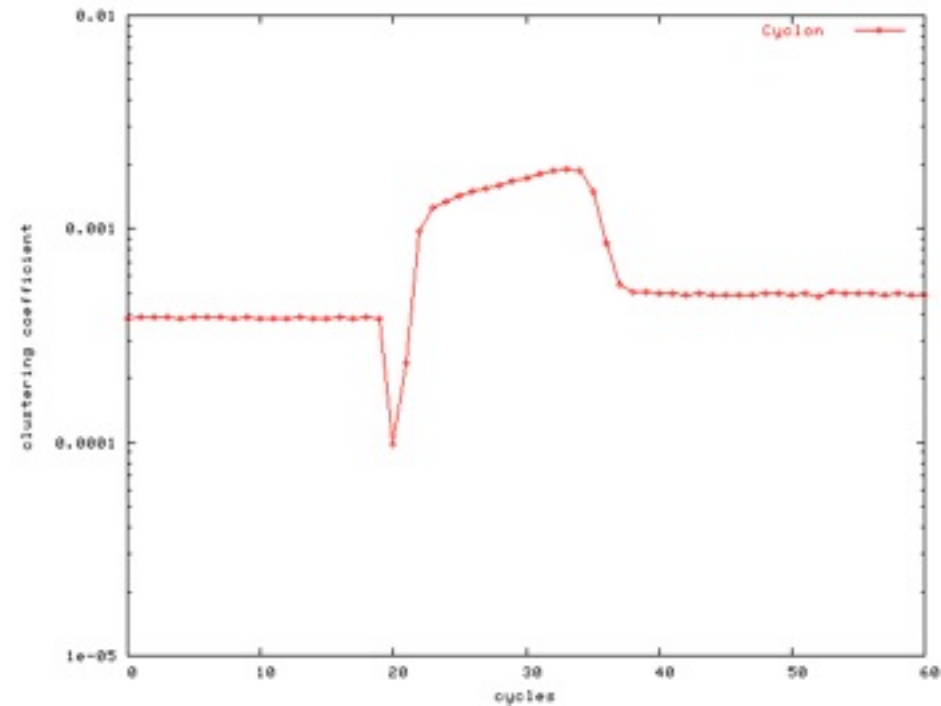
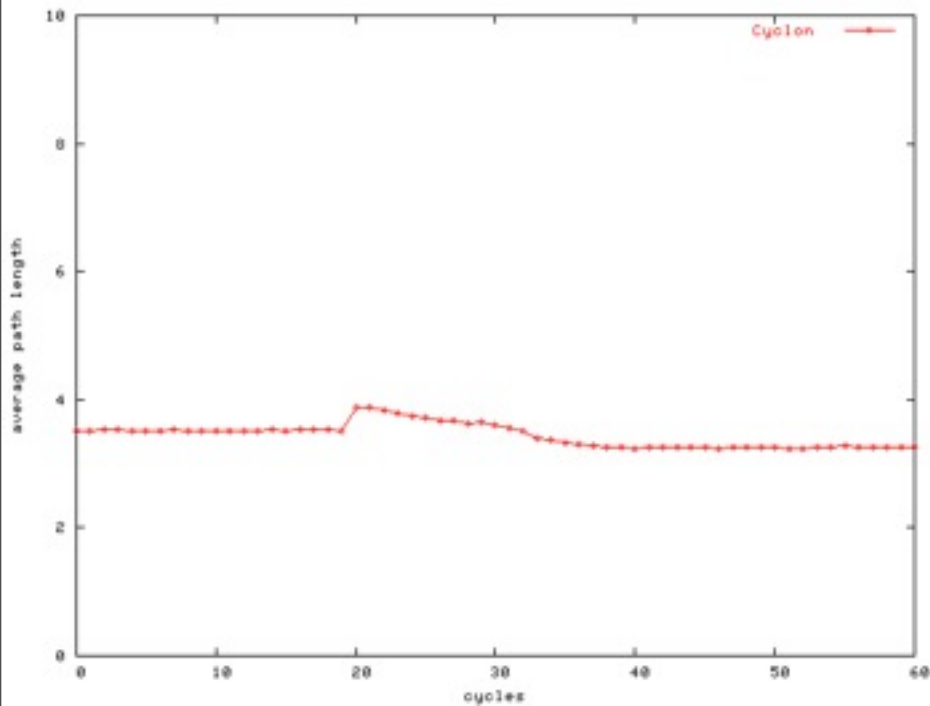


Self-healing behaviour



Self-healing behaviour

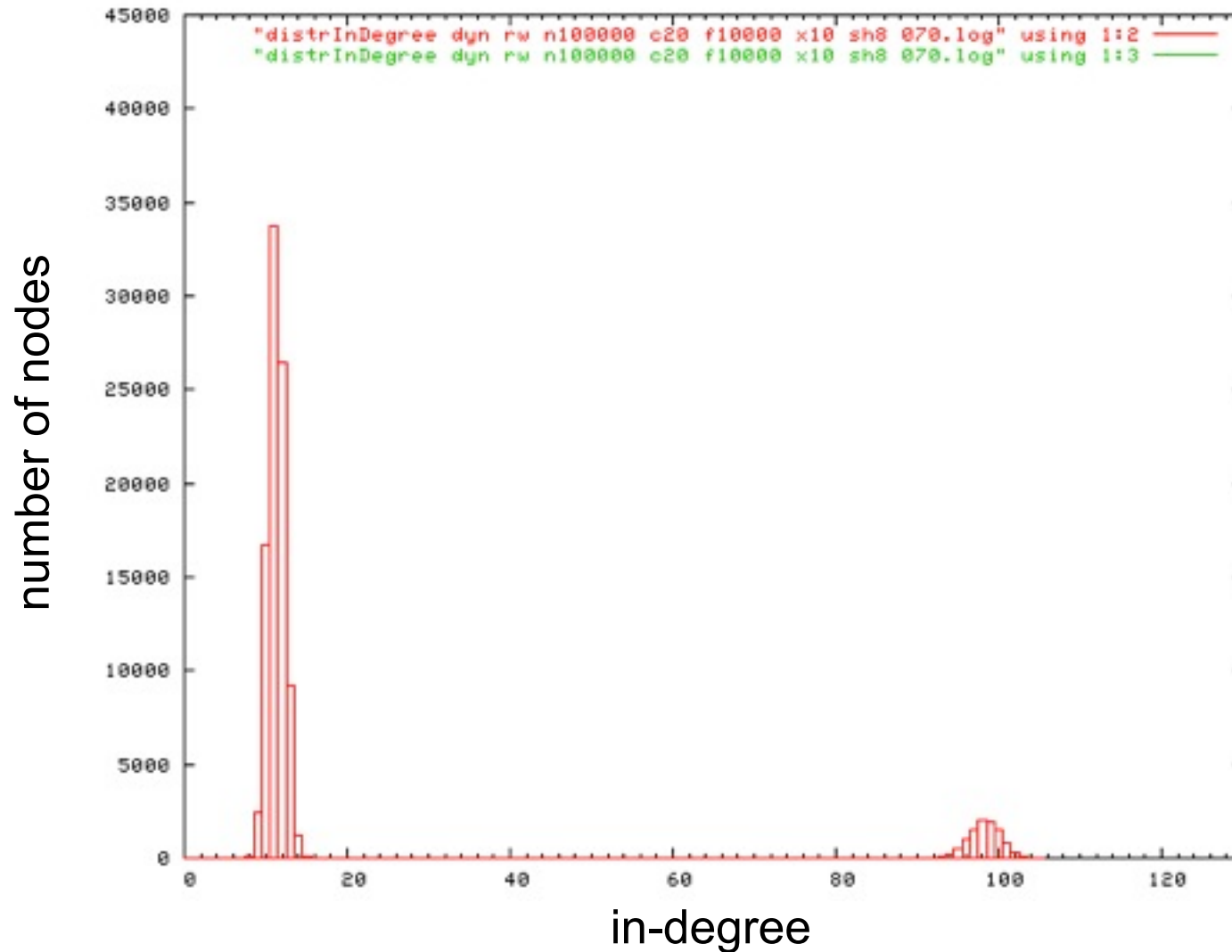
Killed 50,000 nodes at cycle 19



Non-symmetric overlays

- Non-uniform period → Non symmetric topologies
- A node's in-degree is proportional to its *gossiping frequency*
- Can be used to create topologies with “super-nodes”

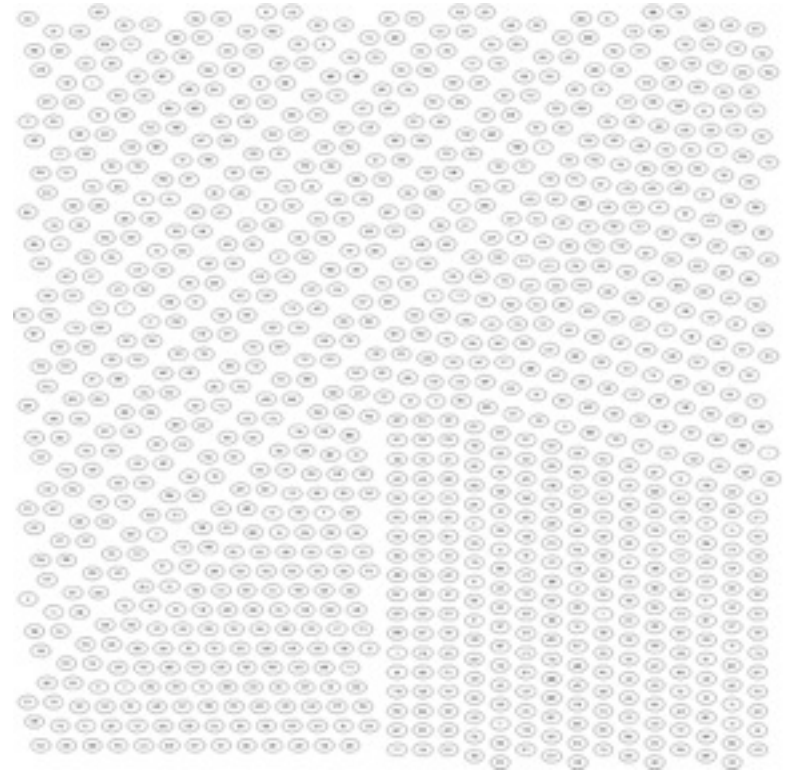
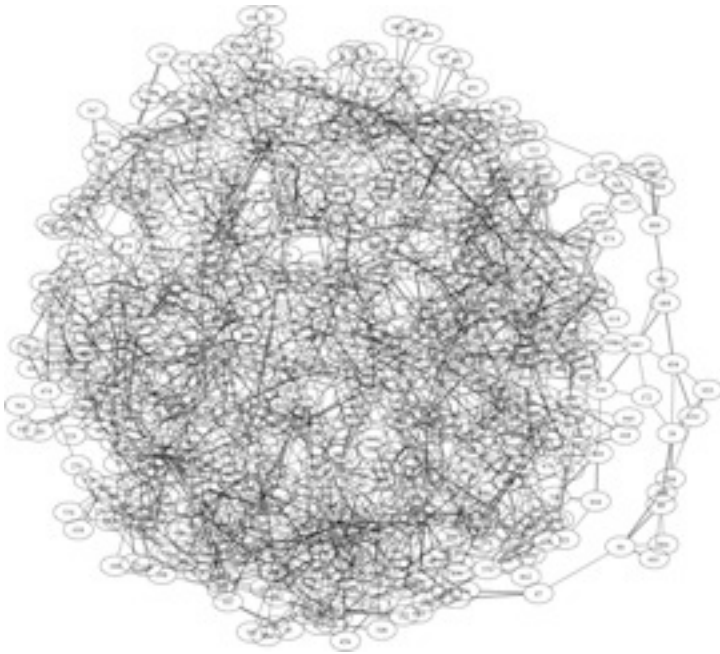
Non-symmetric topologies



Secure peer sampling

- This approach is vulnerable to certain kinds of malicious attacks
- Hub attack
 - Hub attack involves some set of colluding nodes always gossiping their own ID's only
 - This causes a rapid spread of only those nodes to all nodes - we say their views become "polluted"
 - At this point all non-malicious nodes are cut-off from each other
 - The malicious nodes may then leave the network leaving it totally disconnected with no way to recover
 - Hence the hub attack hijacks the speed of the Gossip approach to defeat the network

Secure peer sampling

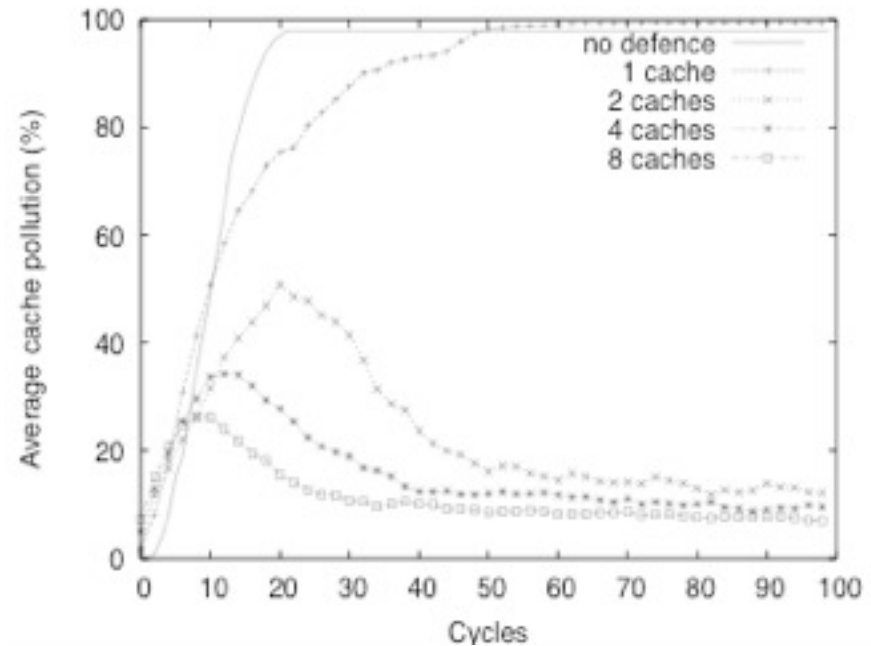
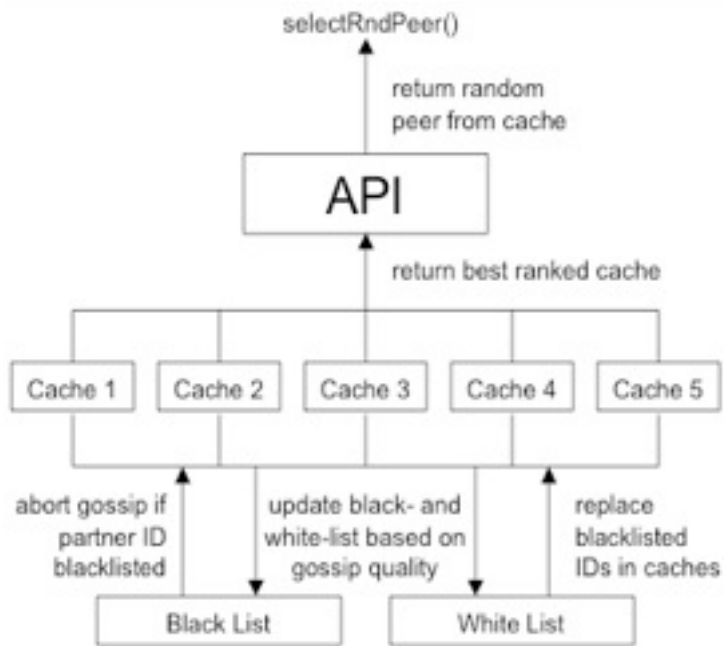


Peer sampling - solution

■ Algorithm

- Maintain multiple independent views in each node
- During a gossip exchange measure similarity of exchanged views
- With probability equal to proportion of identical nodes in two views reject the gossip and blacklist the node
- Otherwise, whitelist the node and accept the exchange
- Apply an aging policy to to both white and black lists
- When supplying a random peer to API select the current “best” view

Secure peer sampling



- 1000 nodes
- 20 malicious nodes