Design of Availability-Dependent Distributed Services in Large-Scale Uncooperative Settings

Final PhD Defense
Ramsés Morales
June 25, 2009

Dept. of Computer Science
University of Illinois at Urbana-Champaign

Distributed Protocols Research Group
http://dprg.cs.uiuc.edu

Outline

- **Introduction**
  - Motivation
  - Thesis
  - Contribution
- Availability Monitoring
- Availability-Aware Membership
- Availability-Aware Aggregation
- Conclusion

Large-Scale Distributed Systems

- Many new large distributed systems
  - P2P file storage
  - Multimedia streaming
  - File distribution
  - Grids and research overlays
  - CDNs & Publish-Subscribe
  - Cloud computing
  - etc.

Large-Scale Distributed Systems

- In spite of variety
  - Need to scale with thousands of nodes
  - Perform efficiently
  - Monitoring, aggregation, membership, querying, etc.
- Must deal with churn and availability variations across nodes
  - Availability: fraction of time a node is online
- Churn and low-availability nodes can make system unreliable, unstable, bandwidth expensive [Bhagwan 2003; Godfrey 2006; Li 2004; Rhea 2004; Stutzbach 2006]

Large-Scale Distributed Systems

- Management
  - Crucial need [Kaashoek et al 2005]
  - Difficult problem [Patterson and Gray 2003]
- Selfish and colluding nodes
  - Gnutella
  - Grid (@home systems)
  - Multimedia streaming

Linking Service to Availability

- We believe it is important to link a node's availability to the service it receives
  - Approach 1: Do nothing [Dabek 2001; Rowstron 2001]
  - Approach 2: Implicit link [Cohen 2003]
  - Approach 3: Explicit link of availability and service can be done by global predicate
    - Example: ***Neighbors(x, y) ≡ f(av(x), av(y))***

Thesis Statement

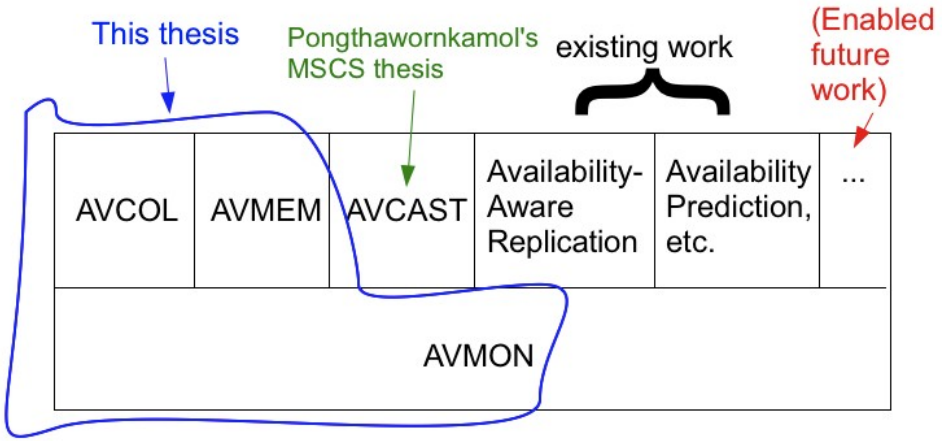
"Availability-dependent global predicates can be efficiently and scalably realized for a class of distributed services, in spite of specific selfish and colluding behaviors, using local and decentralized protocols"

Efficiently and scalably: low memory, message and processing overhead to grow to a large number of nodes.

Local and decentralized protocols: the global predicate is an emergent behavior resulting from actions executed locally at the nodes.

Selfish and colluding: nodes will try to get better service without improving their availability.

Thesis Contributions

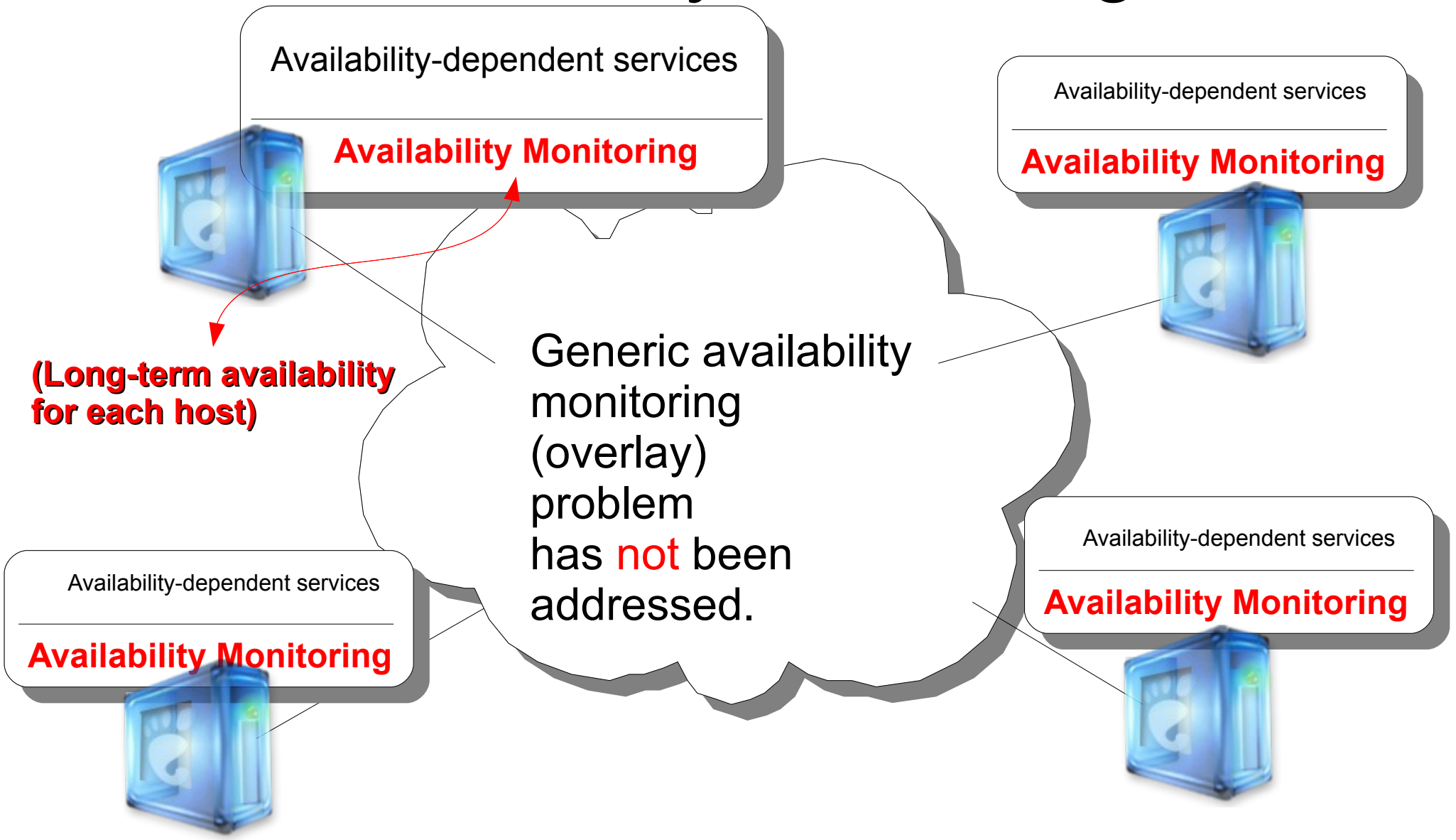


- **AVMEM** [Middleware 07]
  - First availability-aware o
- **AVCOL** [Computer Networ
  - First availability-aware aggregation system
- **AVMEM** & **AVCOL**
  - Implement global predicates relating node availability to service received by the nodes
  - Selfish and colluding nodes are addressed
- **AVMON** [ICDCS 07, IEEE TPDS 09, sourceforge.net]
  - First availability-monitoring overlay
- Mathematical analysis, trace-driven simulation

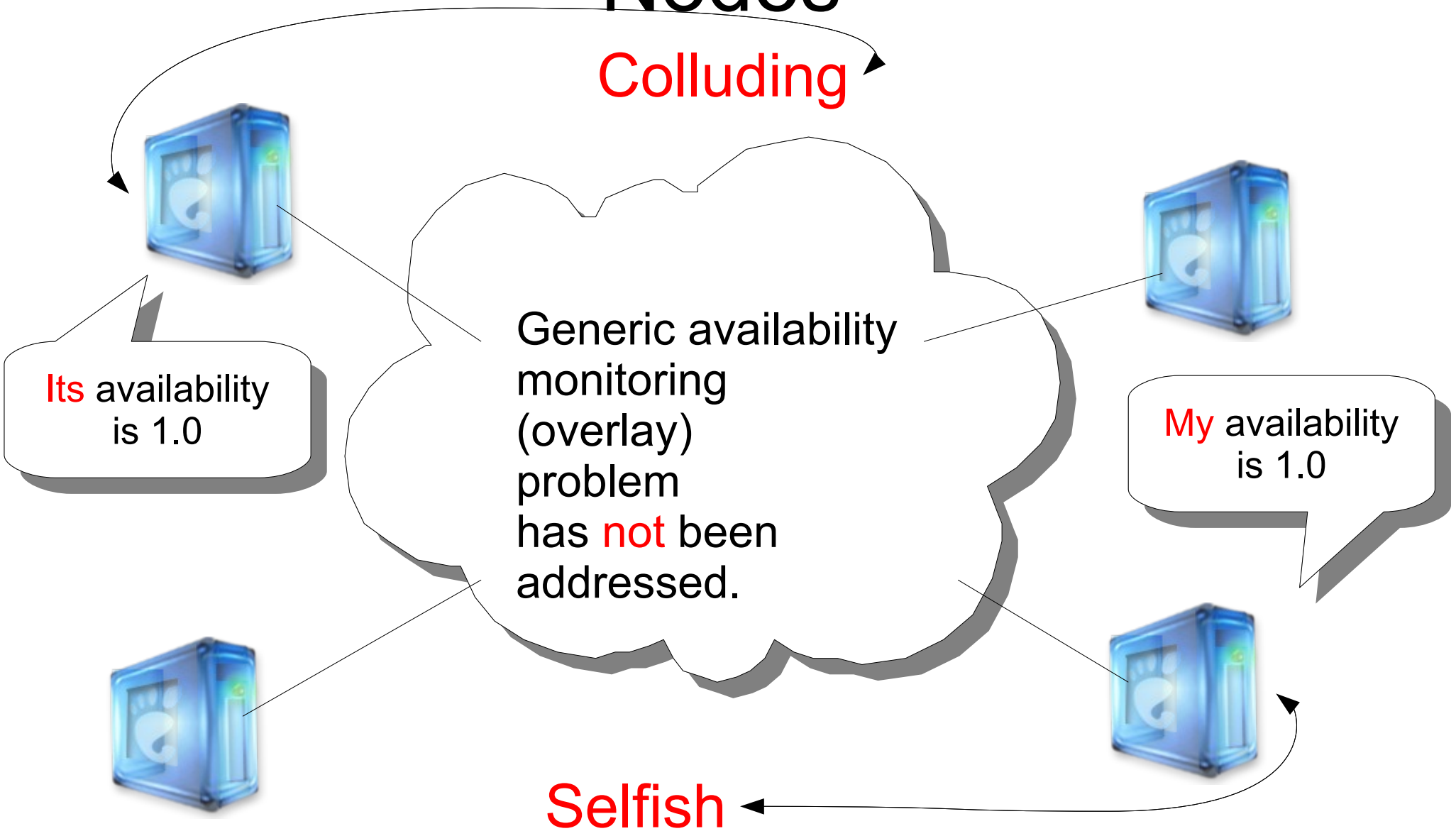
Outline

- Introduction
- **Availability Monitoring**
  - **AVMON** (prelim)
- Availability-Aware Membership
- Availability-Aware Aggregation
- Conclusion

Availability Monitoring

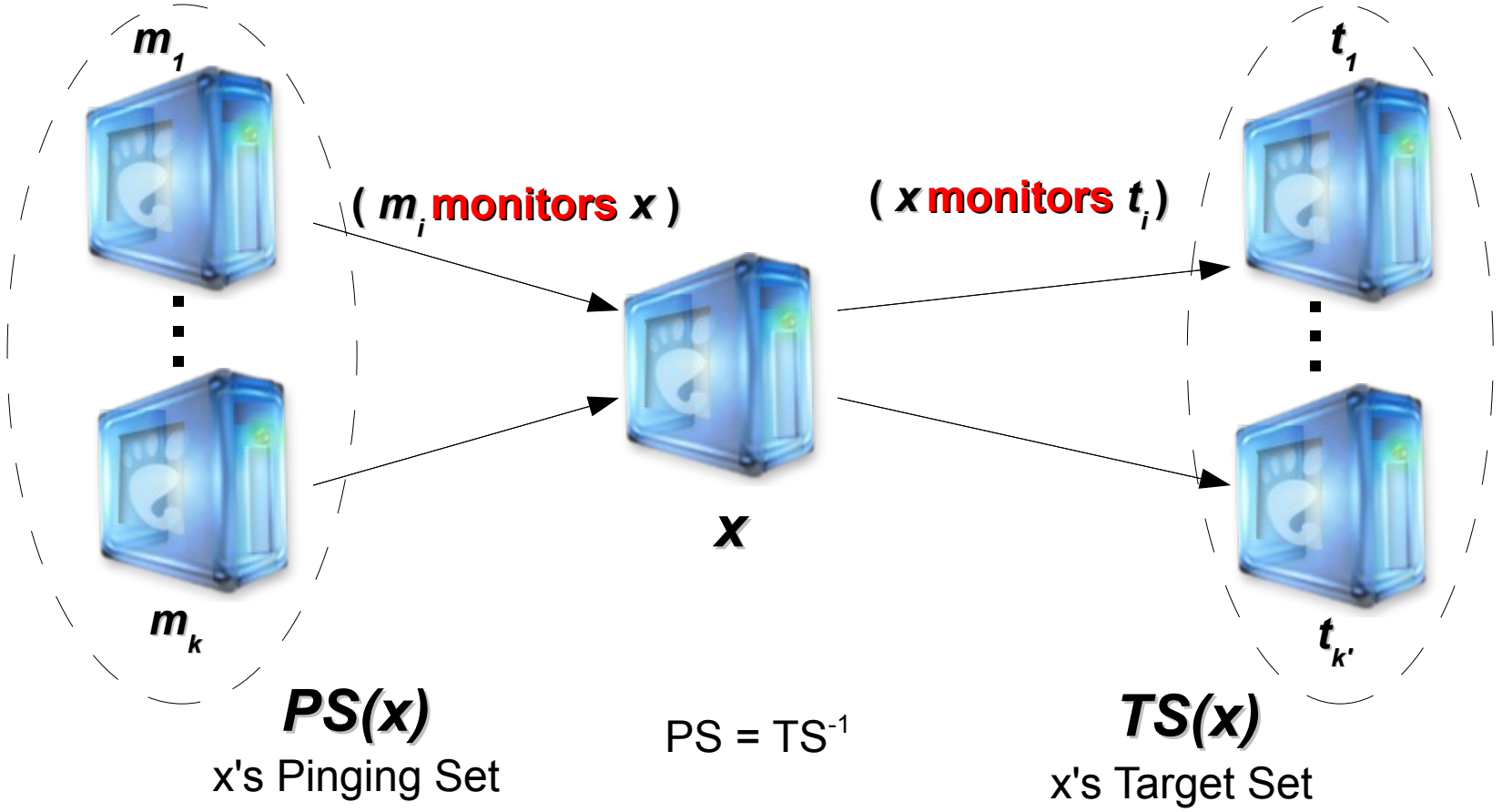


Availability Monitoring and Selfish Nodes



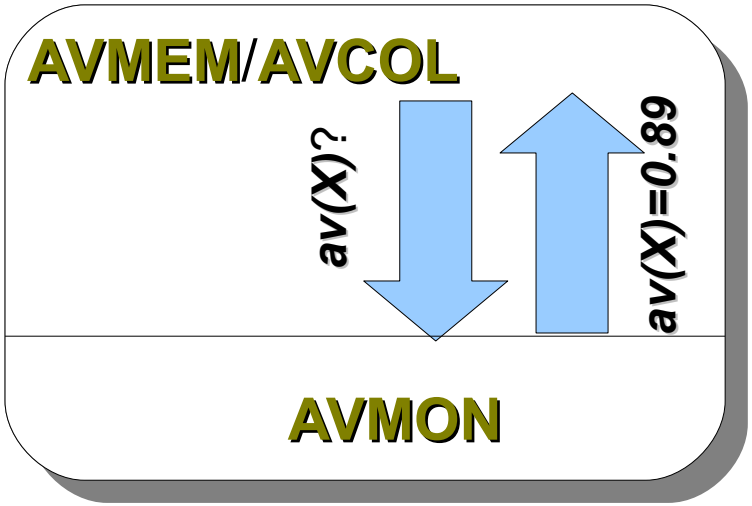
Availability Monitoring Overlay Problem

- *For each node* **x**, *select and discover a small subset of nodes to monitor* **x**.

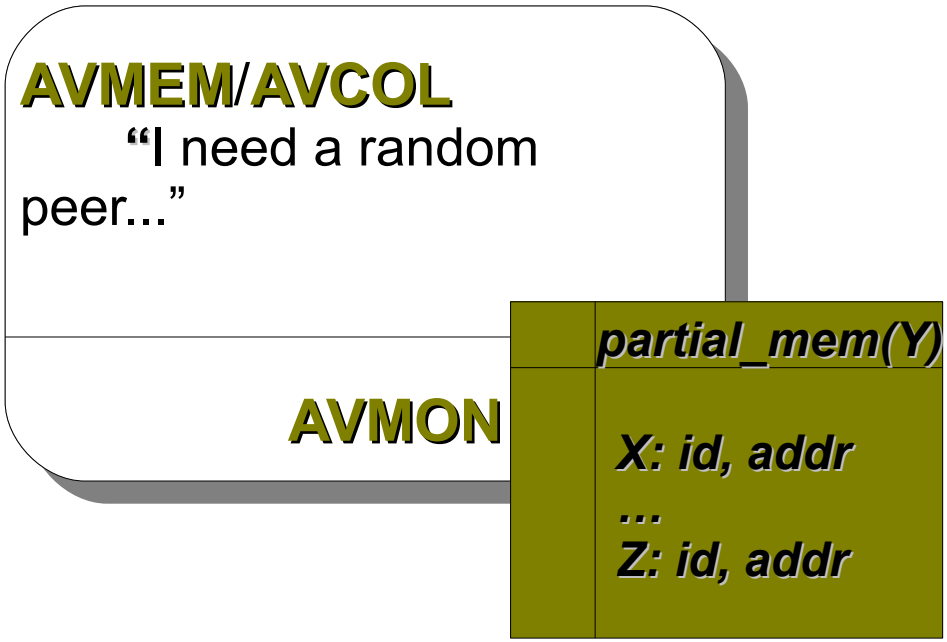


AVMON's use by AVMEM & AVCOL

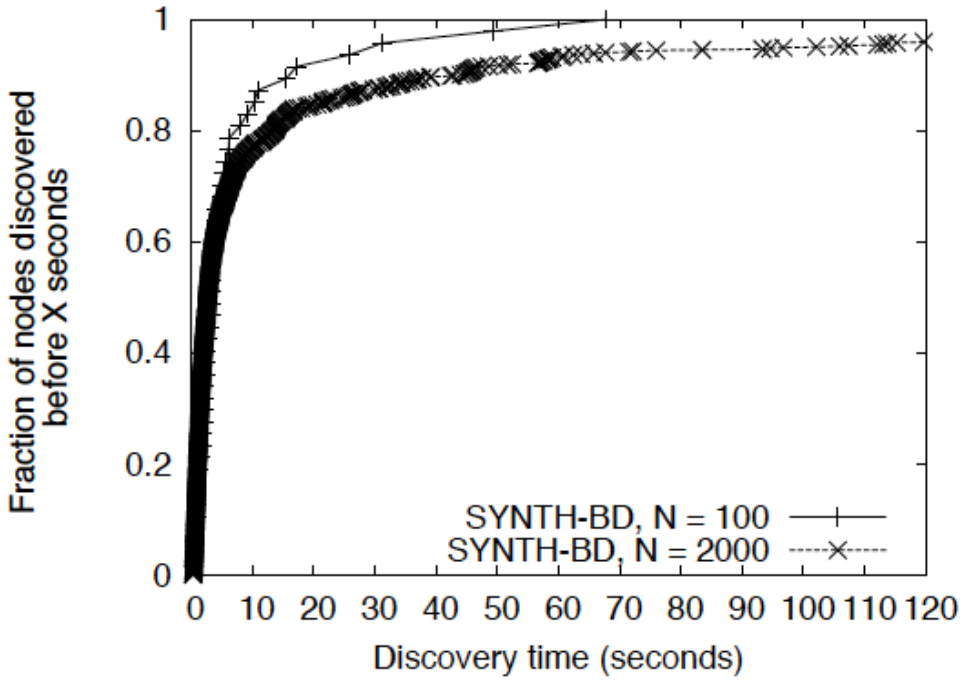
I. Availability information for each node:

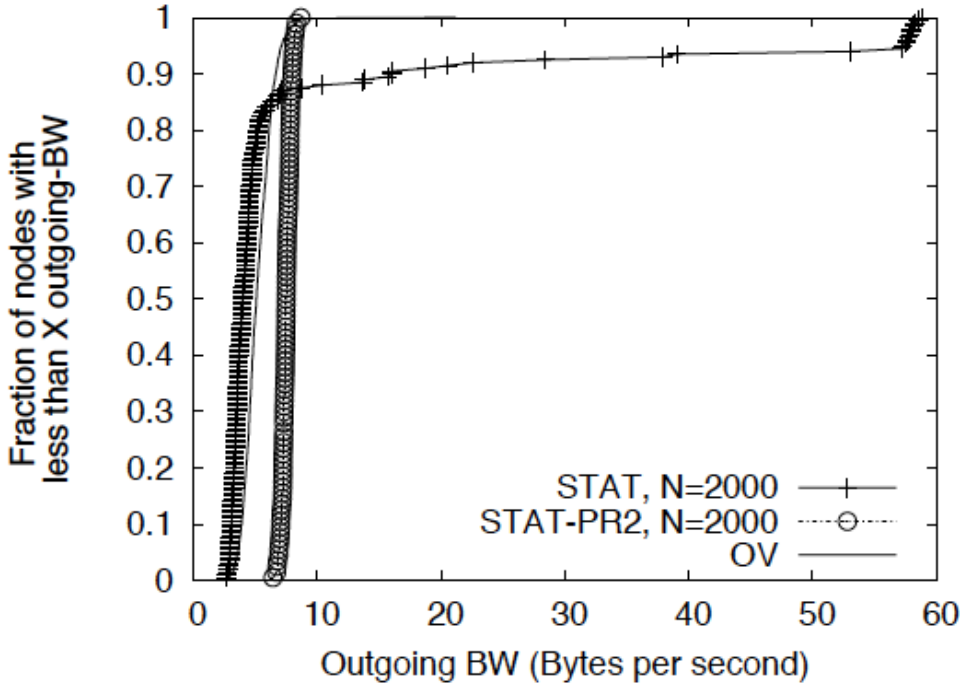


II. Maintains weakly-consistent uniformly-random view:



AVMON's performance





AVMON provides these services effectively and efficiently

Outline

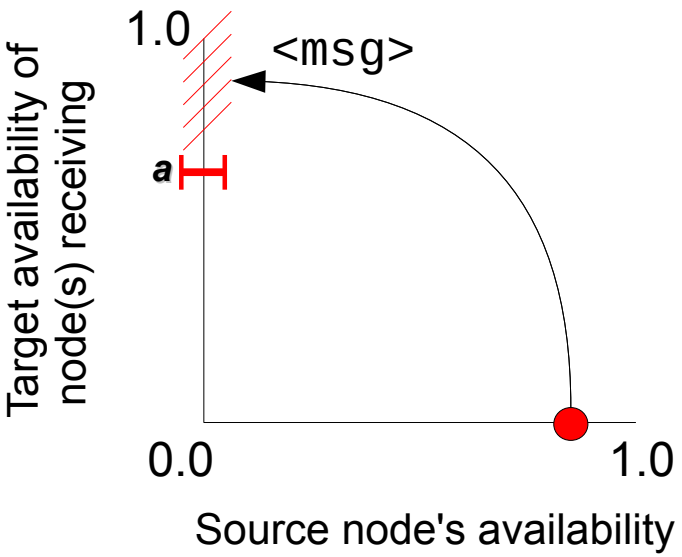
- Introduction
- Availability Monitoring
- **Availability-Aware Membership**
  - Motivation; Problems; Design goals
  - **AVMEM**: Membership graph predicates; Family of availability-aware predicates; Membership maintenance; Membership maintenance; Management operations
  - Experiments
- Availability-Aware Aggregation
- Conclusion

Motivation

- New operations for distributed management applications
  - P2P
  - Grid
  - PlanetLab
- Availability-based operations
  - Multicast
  - Anycast

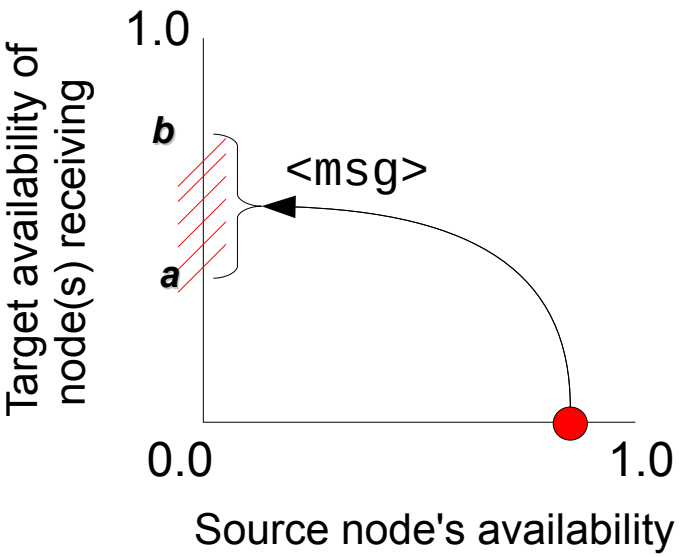
Motivation: Availability-based Multicast & Anycast

- Threshold-multicast, threshold-anycast
  - to nodes with availability **> *a***, ***a* ∈ *[0, 1)***
- Useful for
  - Control operations
    - "Distinguished" peer selection
      - [Liang 2006; Lo 2005; Min 2006]
    - Replica placement
      - [Bhagwan 2004; Chun 2006]



Motivation: Availability-based Multicast & Anycast

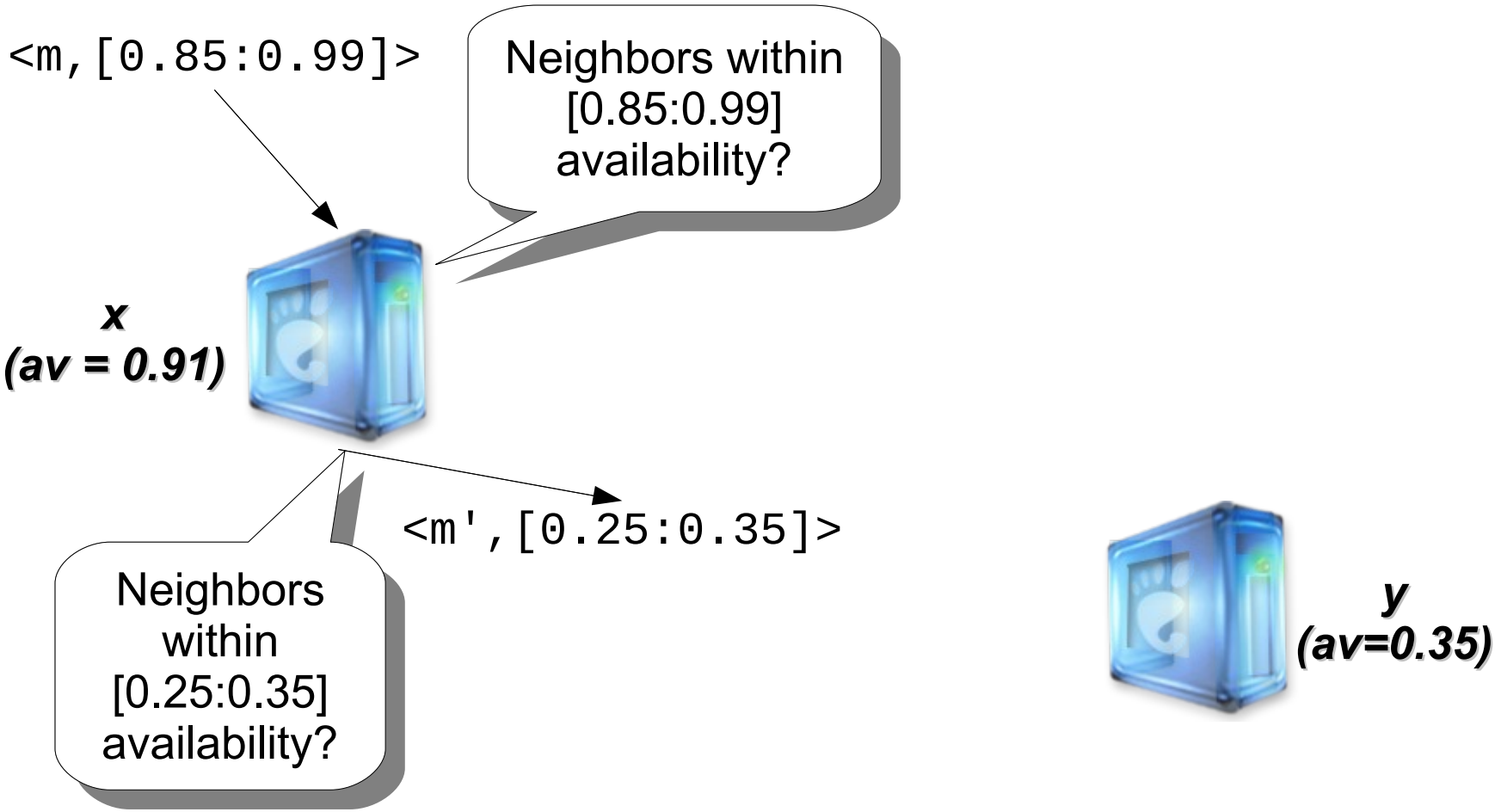
- Range-multicast, range-anycast
  - to nodes with availability in range ***[a, b]⊆ [0, 1]***
- Useful for
  - Node characteristic fingerprint
  - Replica placement
    - [Bhagwan 2004; Chun 2006]
  - Grid instance placement
    - [Cappello 2005]



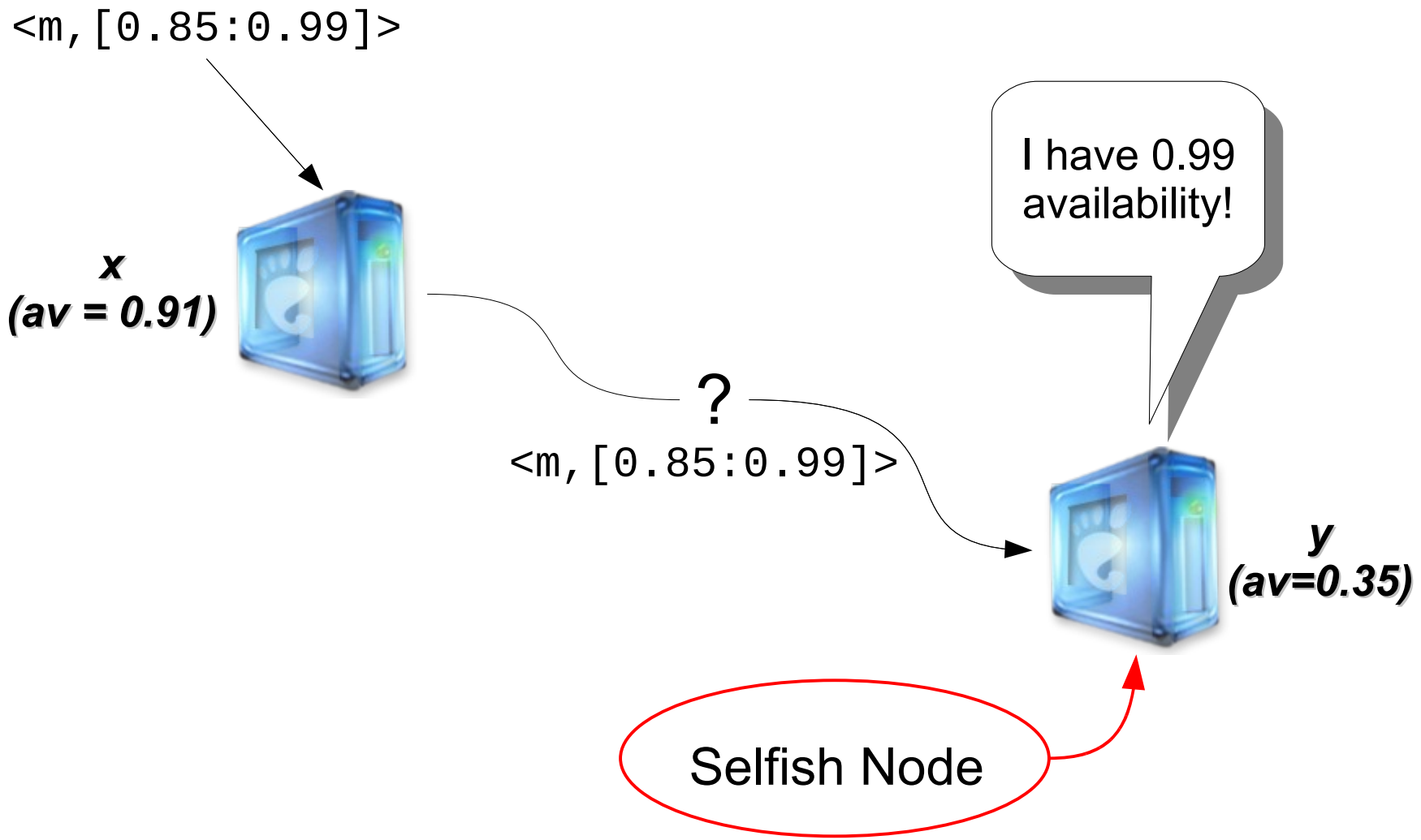
Problems

- Difficult to predict availability variations across nodes and time
- Selfish and colluding nodes

Problems


<m,[0.85:0.99]>
Neighbors within [0.85:0.99] availability?
x
(av = 0.91)
<m',[0.25:0.35]>
Neighbors within [0.25:0.35] availability?
y
(av=0.35)

Problems


<m,[0.85:0.99]>
x
(av = 0.91)
?
<m,[0.85:0.99]>
I have 0.99 availability!
y
(av=0.35)
Selfish Node

Discarded Solutions

- Centralized
- DHT-based [Rowstron 2001; Stoica 2001]
  - ***nodeID = H(av(x))***
- Overlays for range-queries
  - Skip-trees, Voronoi, Segment trees, others
    - [Aspnes 2003; Harvey 2003; Shu 2005; Sheng 2006]

Our Approach

- Need:
  - Availability-Aware Membership, **AVMEM**
  - Availability Monitoring, **AVMON**

Design Goals

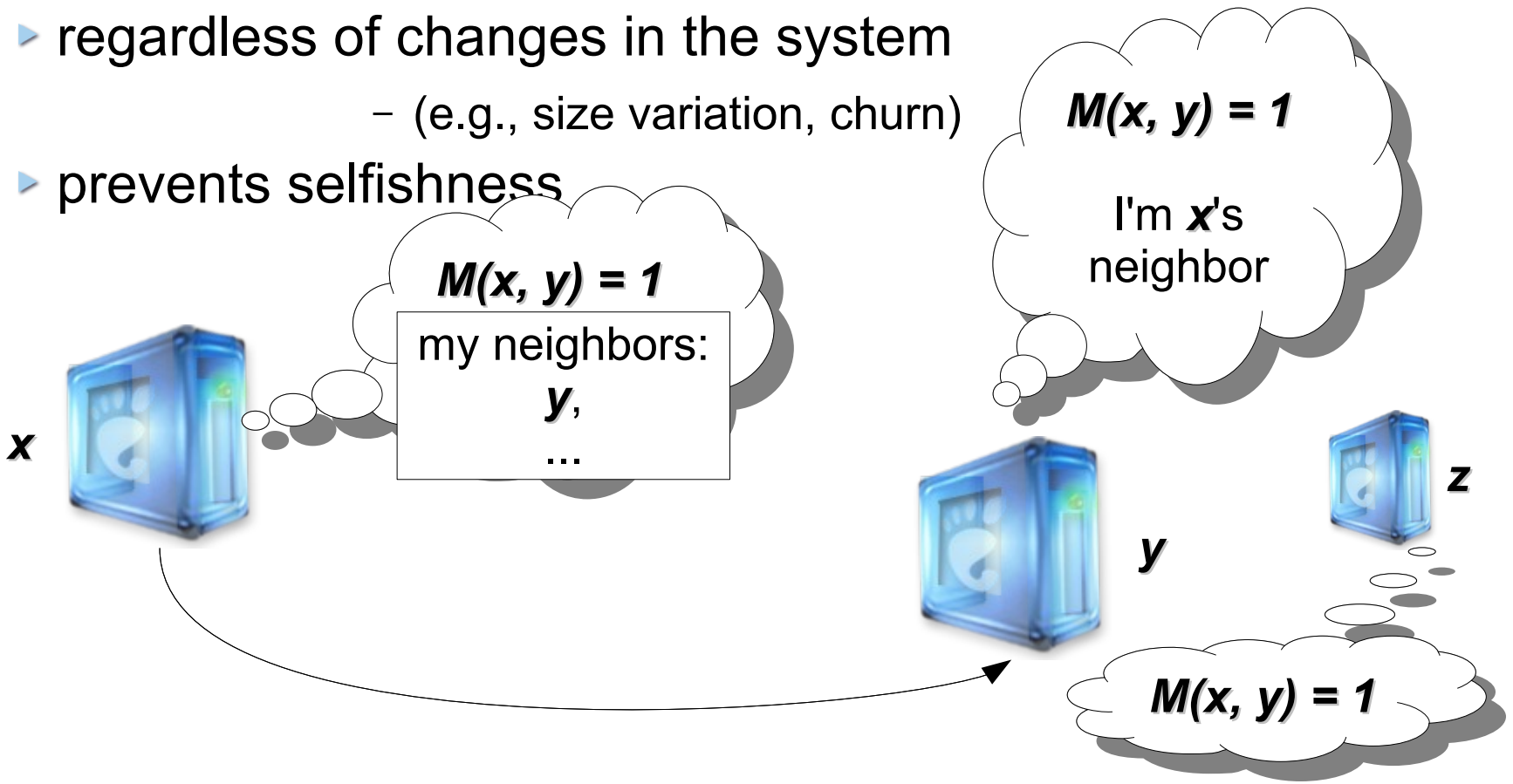
- Overlay where neighbor selection is based on node availability
- Decentralized realization of a global-predicate for membership relations
- Availability-based operations for management operations

Global Predicate

- ***M(x, y) ∈ {0, 1}***
  - Depends on:
    - ***av(x)***, ***av(y)***
    - ***y***'s public id, e.g., `<ip, port>`
    - ***x***'s public id, e.g., `<ip, port>`

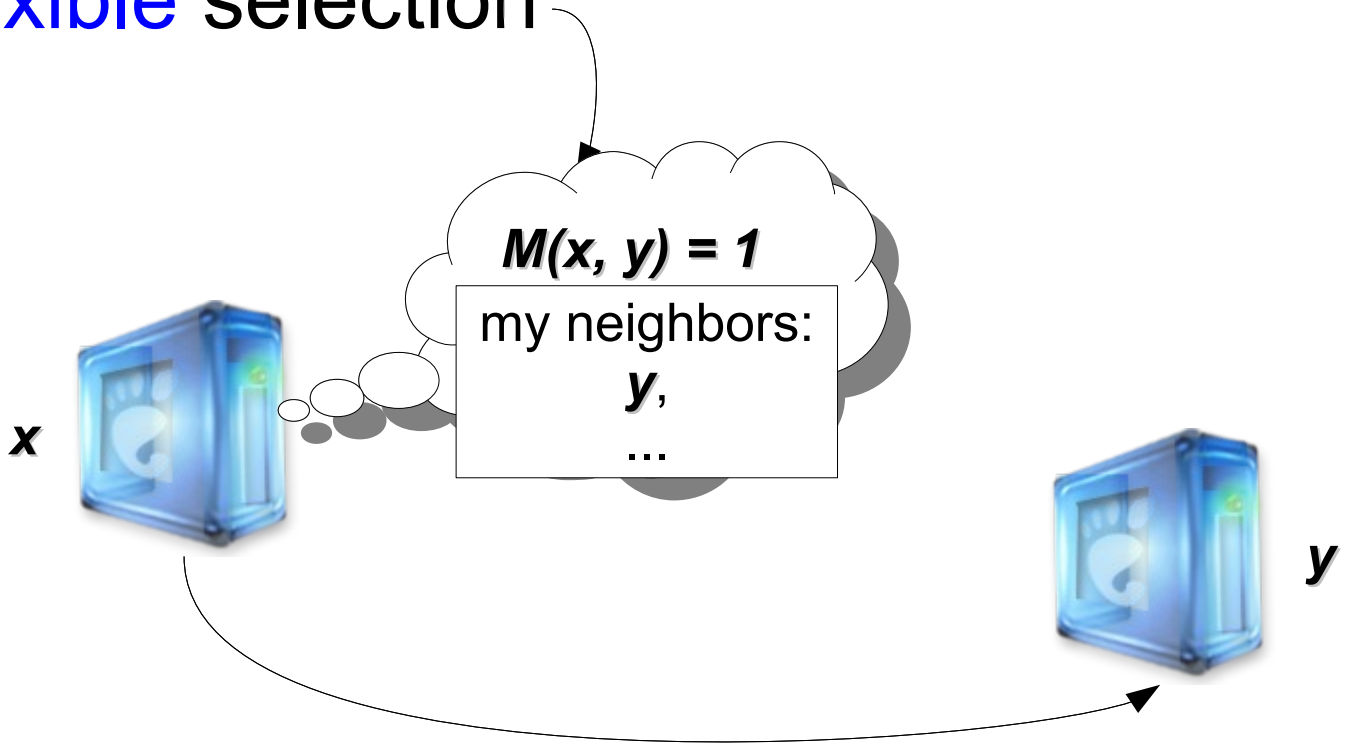
Global Predicate

- ***M(x, y) ∈ {0, 1}***
  - Consistent
    - regardless of changes in the system
      - (e.g., size variation, churn)
    - prevents selfishness



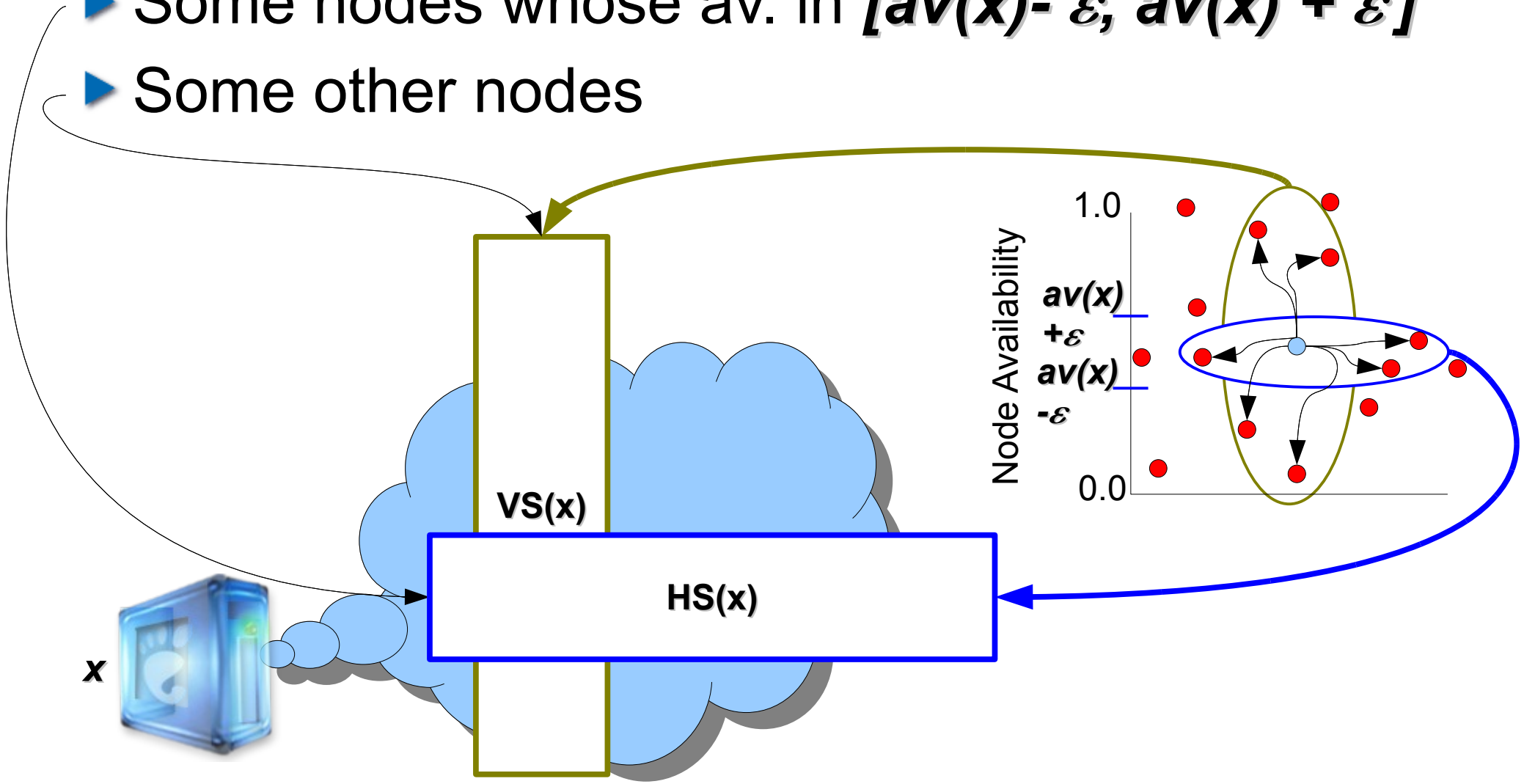
Global Predicate

- Small number of neighbors
- Randomized neighbors
- Flexible selection



AVMEM: Membership Graph Predicates

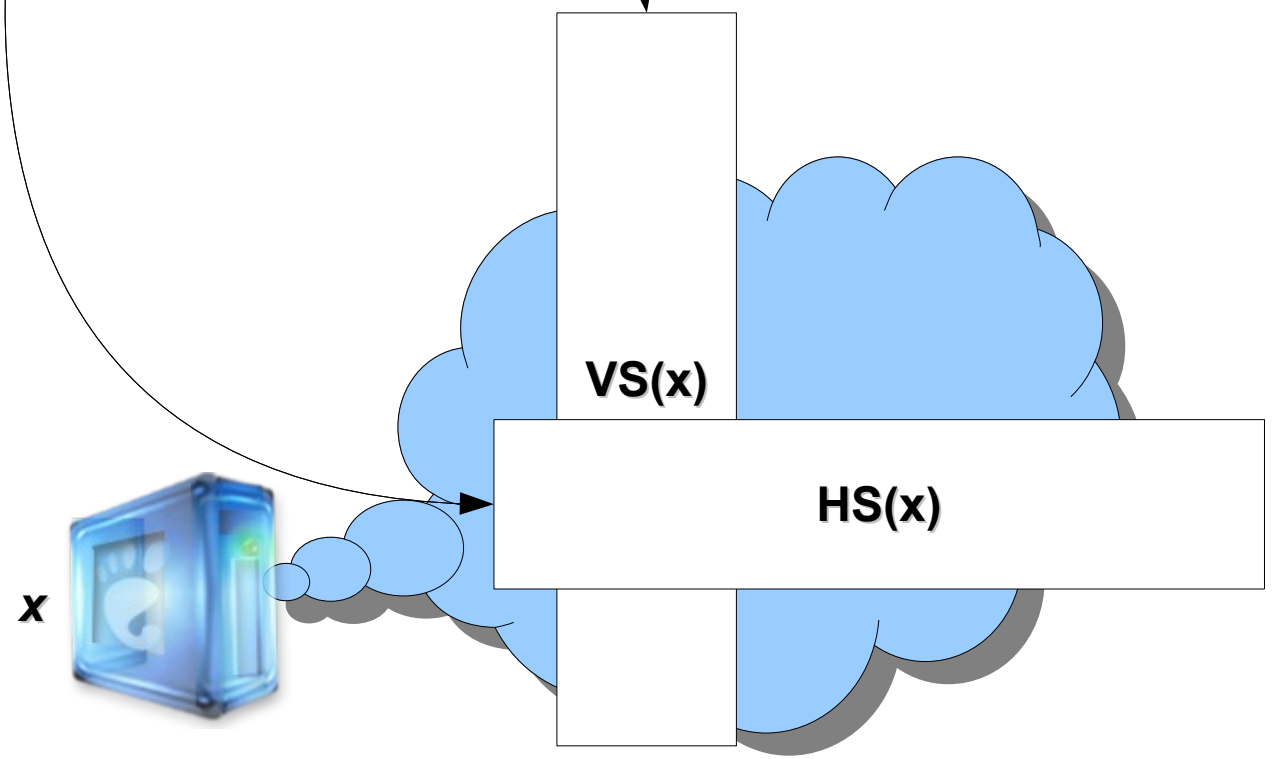
- Some nodes whose av. in ***[av(x)- ε, av(x) + ε]***
- Some other nodes



AVMEM: Membership Graph Predicates

$$M(x, y) \equiv \{ H(id(x) \mid id(y)) \leq f(av(x), av(y)) \}$$

$$M(x, y) = 1 \text{ iff } x \longrightarrow y$$

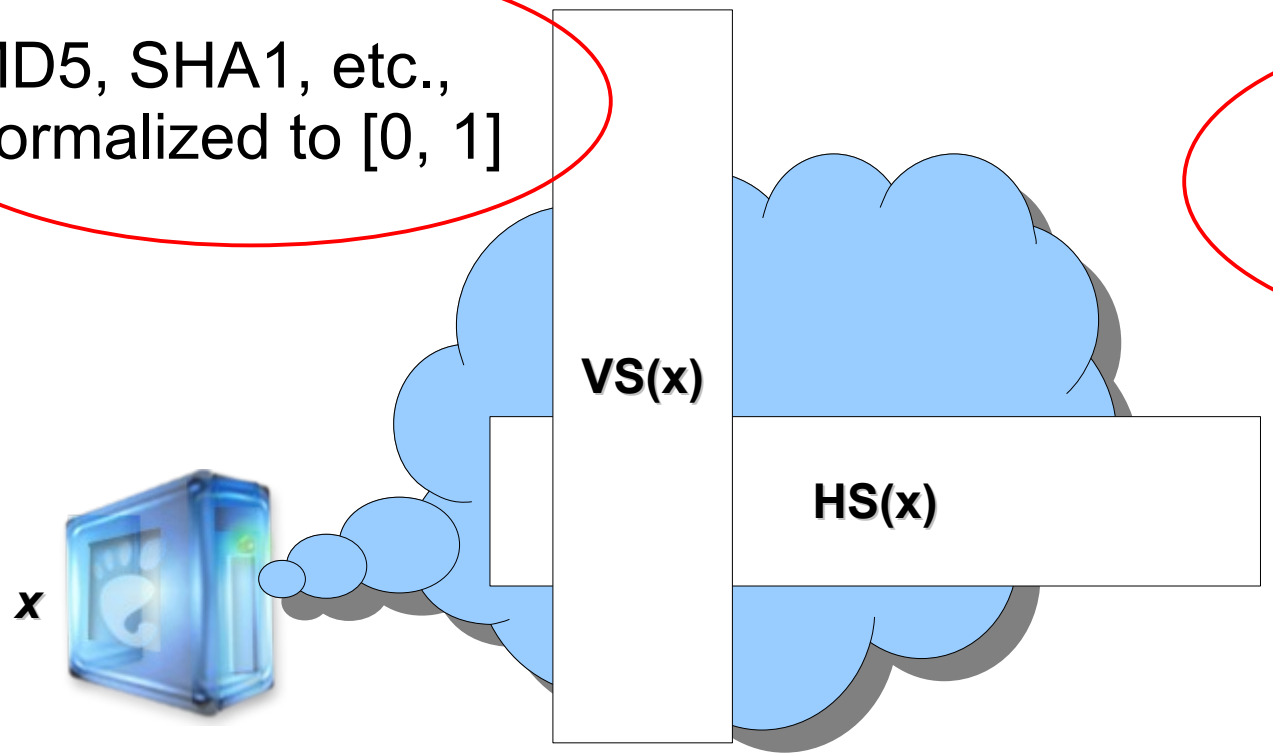


AVMEM: Membership Graph Predicates

$$M(x, y) \equiv \{ H(id(x) \mid id(y)) \leq f(av(x), av(y)) \}$$

MD5, SHA1, etc., normalized to [0, 1]

core function that determines emergent behavior



AVMEM: Family of Availability-Aware Predicates

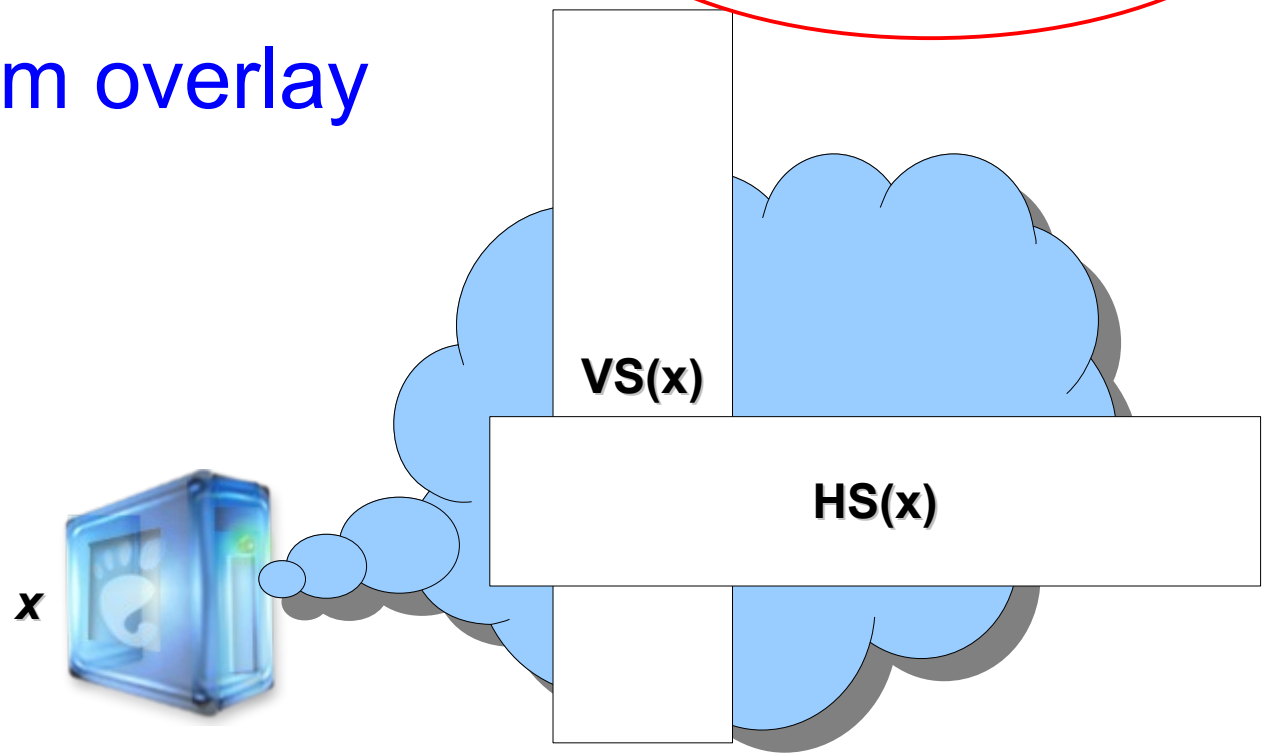
- Vertical sub-predicate:
  - Constant vertical sliver
  - Logarithmic vertical sliver
  - Logarithmic-decreasing vertical sliver
- Horizontal sub-predicate:
  - Constant horizontal sliver
  - Logarithmic-constant horizontal sliver

AVMEM: Constant Vertical/Horizontal Sliver

f(av(x), av(y))= $O\left(\frac{log(N)}{N}\right)$

Stable system size:
Number of online nodes varies within a constant factor of ***N***
(P2P systems, PlanetLab, Grid)
[Bhagwan 2003; Stutzbach 2006]

- Uniformly random overlay

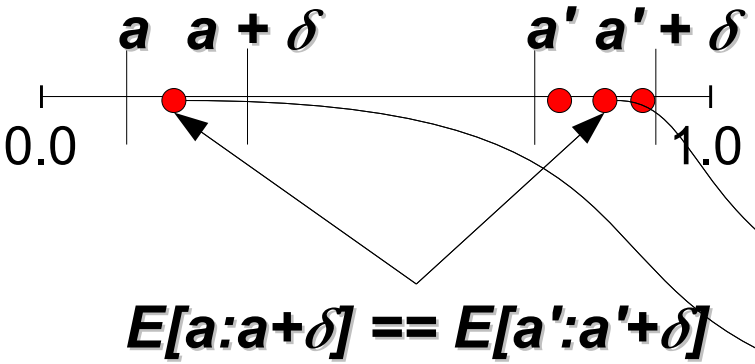


AVMEM: Logarithmic Vertical Sliver

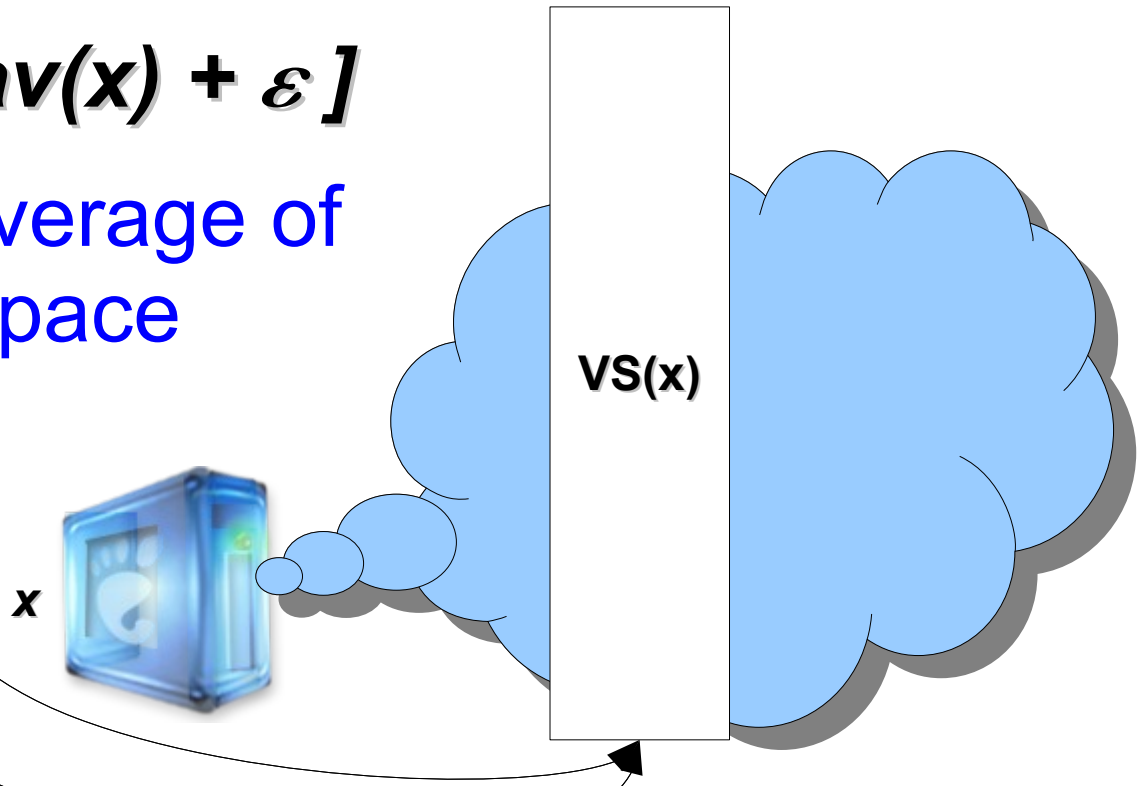
$$f(av(x), av(y)) = \frac{c_1 \cdot log(N)}{N \cdot p(av(y))}$$

fraction of nodes in *[av(y)-δ,av(y)]*

- $y \notin [av(x) - \varepsilon, av(x) + \varepsilon]$
- Uniformity of coverage of the availability space



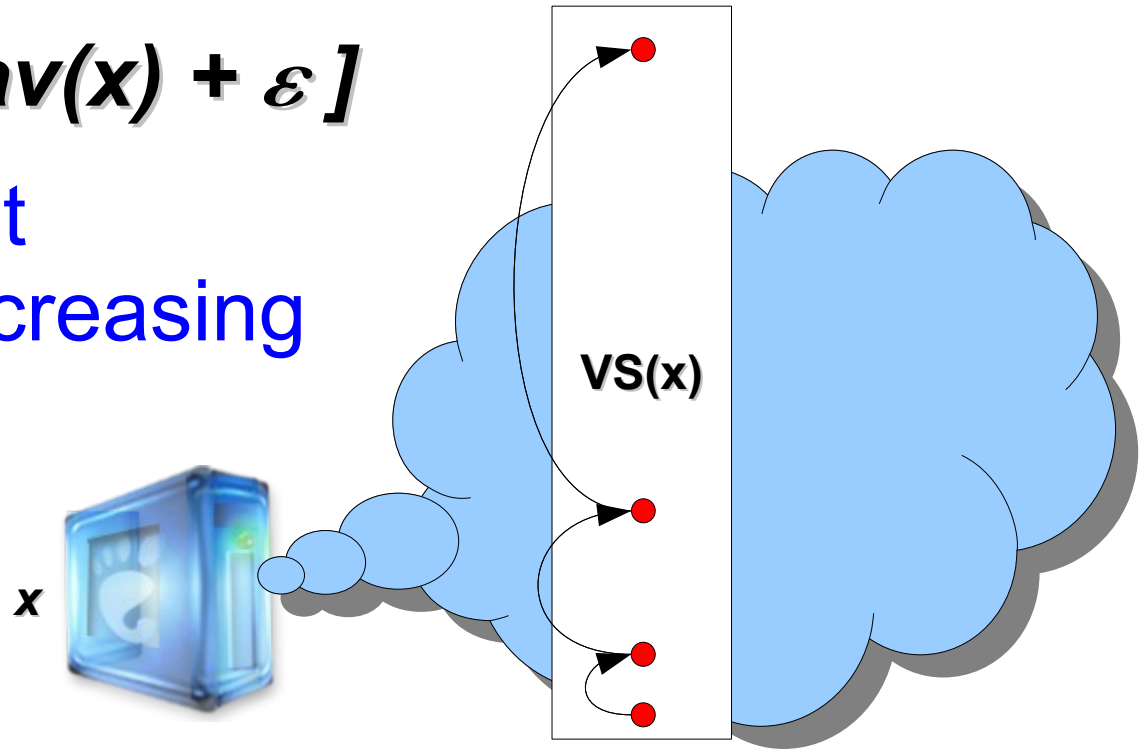
E[a:a+δ] == E[a':a'+δ]



AVMEM: Logarithmic-decreasing Vertical Sliver

f(av(x), av(y))= $\dfrac{c_1 \cdot log(N)}{N \cdot p(av(y)) \cdot |av(y) - av(x)|}$

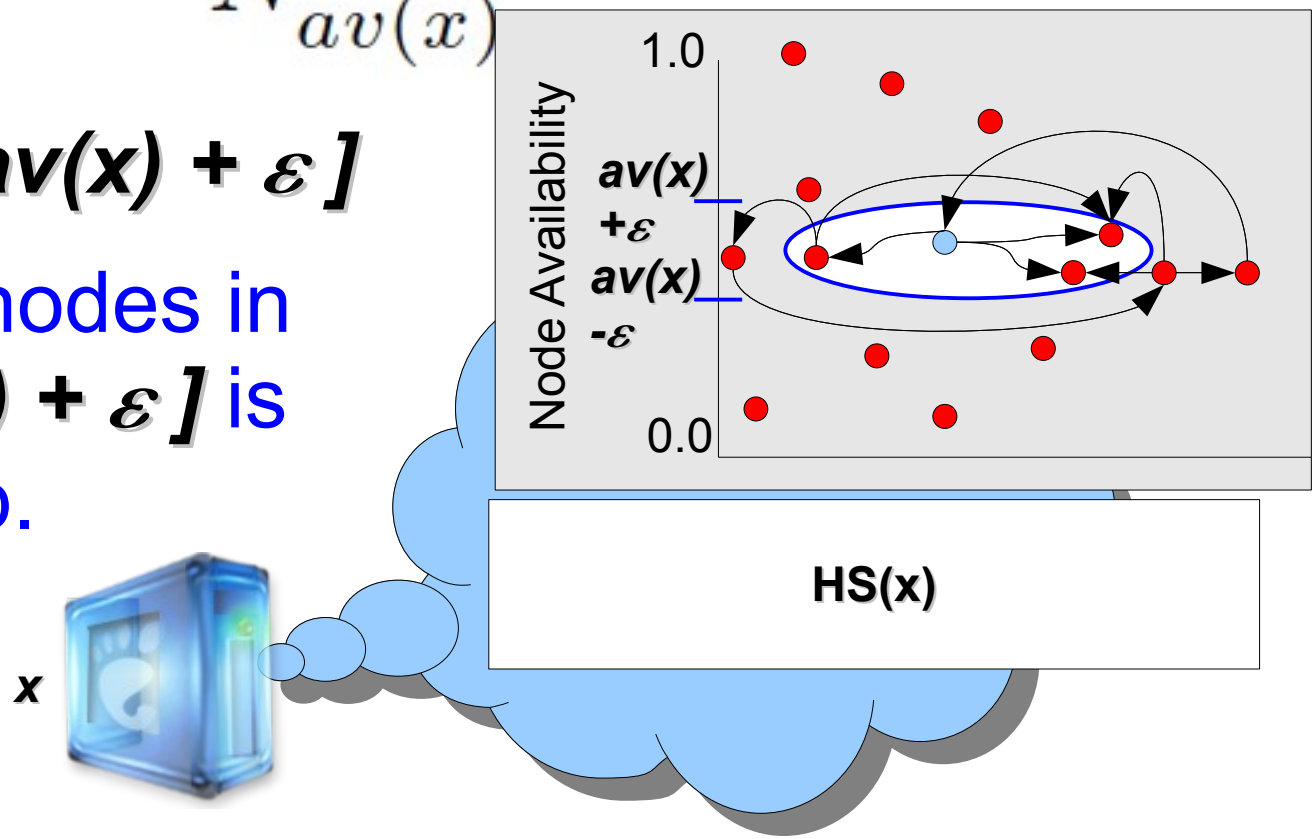
- ***y ∉ [av(x) – ε, av(x) + ε]***
- Neighbors are at exponentially increasing distance from ***x***



AVMEM: Logarithmic Horizontal Sliver

$$f(av(x), av(y)) = \frac{c_2 \cdot log(N_{av(x)})}{N^{min}_{av(x)}}$$

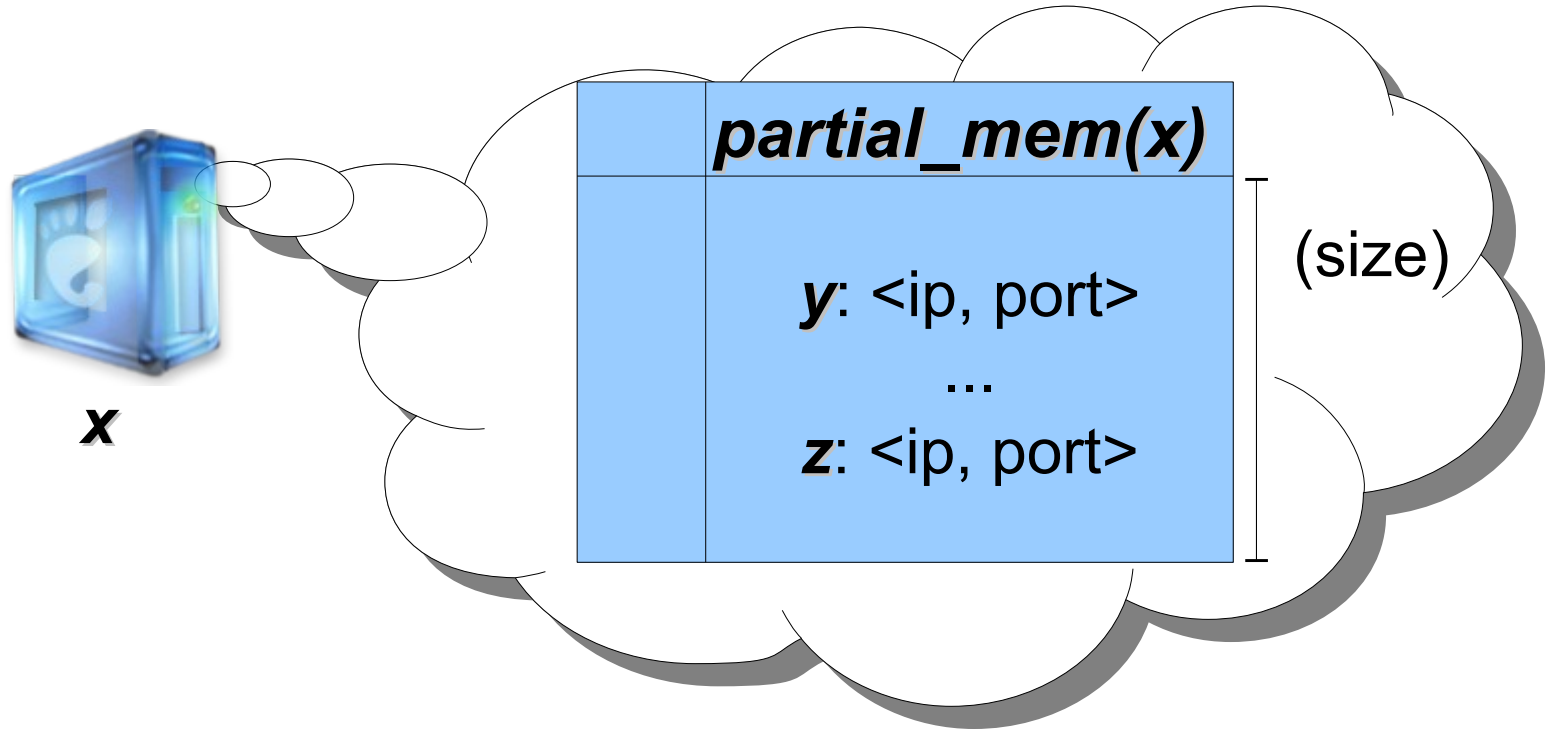
- $y \in [av(x) - \varepsilon, av(x) + \varepsilon]$
- Sub-overlay of nodes in $[av(x) - \varepsilon, av(x) + \varepsilon]$ is connected w.h.p.



AVMEM: Membership Maintenance

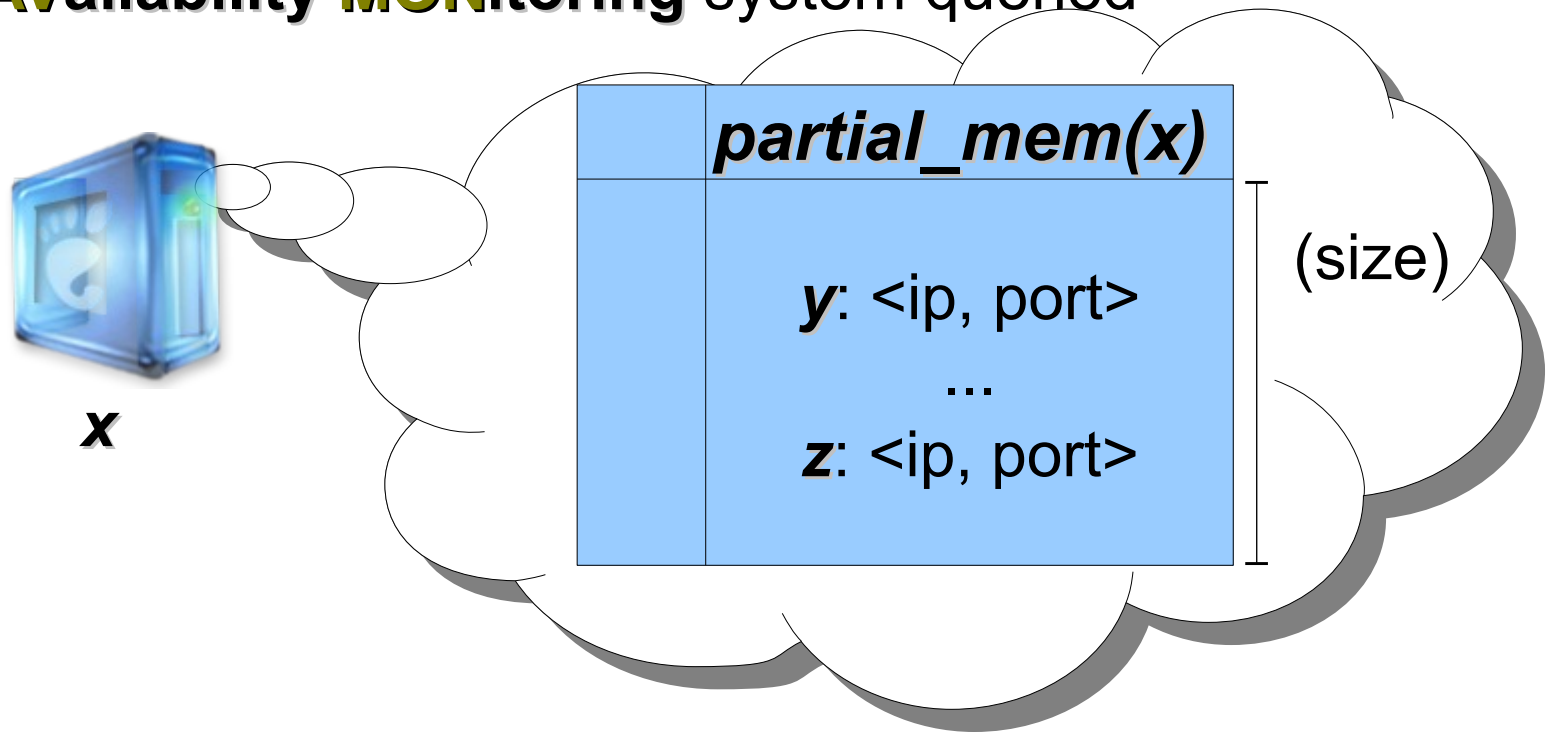
AVMEM: Membership Maintenance

- Discovery Sub-Protocol:
  - Needs random changing partial membership list
    - [Ganesh 2003 (SCAMP)], [Jelasity 2005 (T-Man)],[Voulgaris 2005 (CYCLON)]
  - Needs an availability monitoring system (**AVMON**)



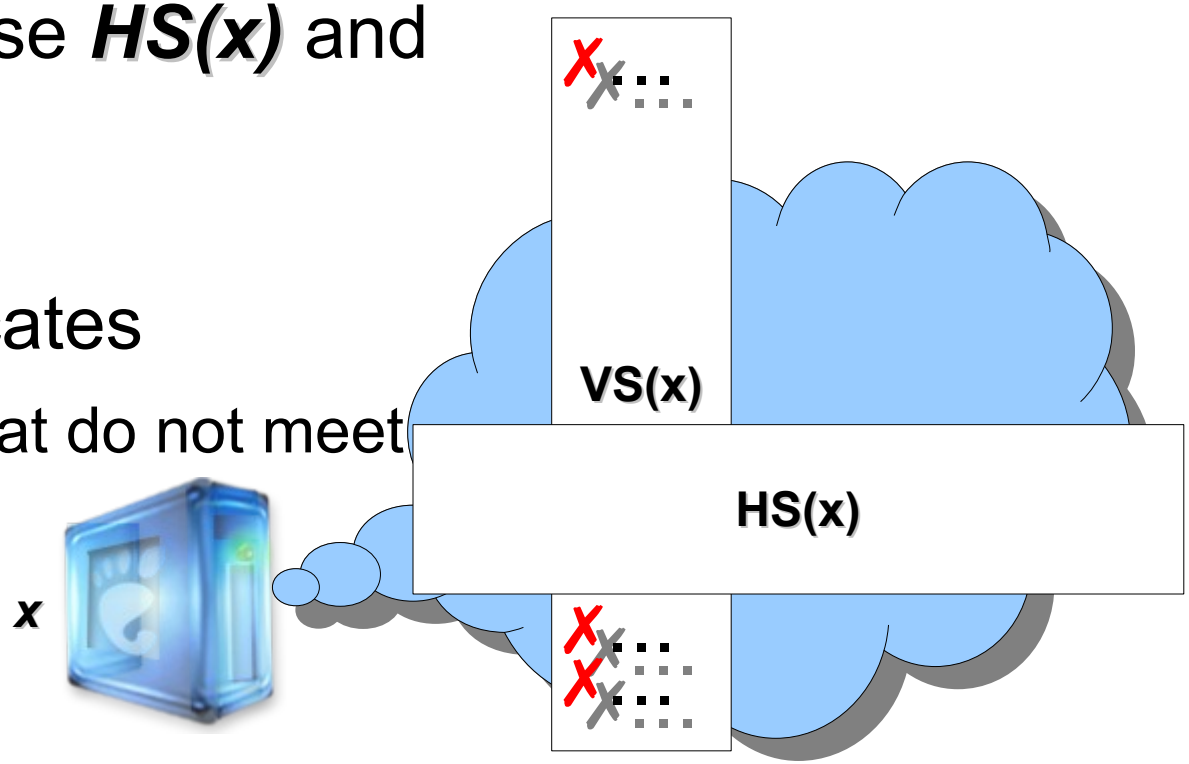
AVMEM: Membership Maintenance

- Discovery Sub-Protocol:
  - Periodically traverse ***partial_mem(x)***
    - Exclude nodes already in ***HS(x) + VS(x)***
    - Predicates are applied to nodes in partial membership
    - **AVailability MONitoring** system queried

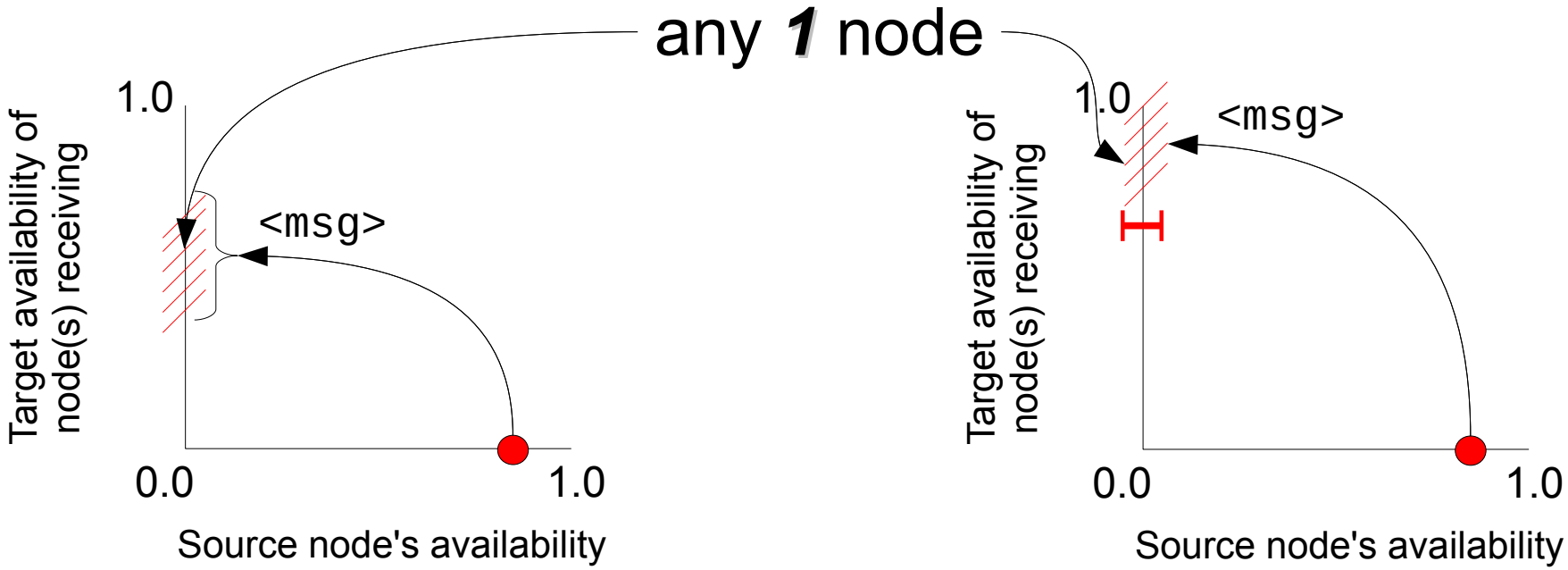


AVMEM: Membership Maintenance

- Refresh sub-protocol:
  - Entries can become stale
    - Nodes can change availability
  - Periodically traverse ***HS(x)*** and ***VS(x)***
    - 20 min
  - Recompute predicates
    - Drop neighbors that do not meet predicates



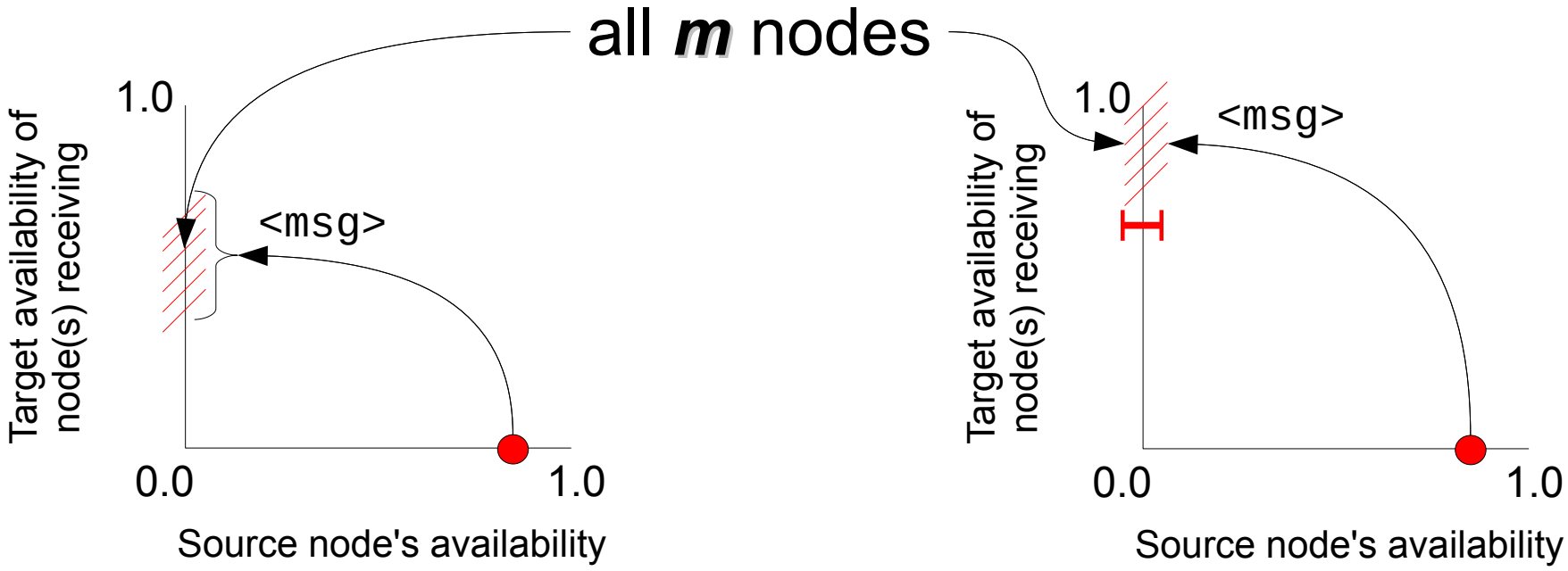
AVMEM: Threshold/Range-Anycast



AVMEM: Threshold/Range-Anycast

- Greedy Forwarding:
  - Select as next hop **AVMEM**-neighbor with availability closest to target
  - TTL
- Retried Greedy Forwarding:
  - Try next-best **AVMEM**-neighbor if best is unresponsive
  - up to ***k*** retries
- Simulated Annealing:
  - Choose random **AVMEM**-neighbor with ***p = exp(-delta / ttl),*** otherwise greedy

AVMEM: Threshold/Range-Multicast



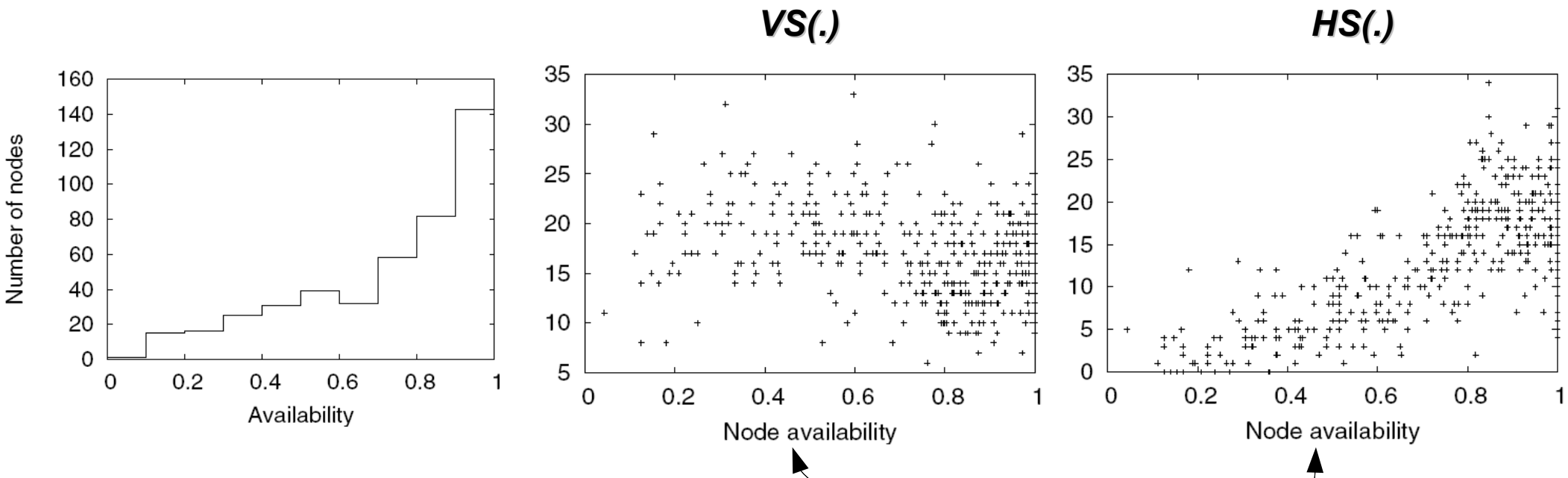
AVMEM: Threshold/Range-Multicast

- Flooding:
  - Forwards to neighbors in range
  - Ignore duplicate messages
- Gossiping:
  - Once per period, forward to ***fanout*** neighbors in range (excluding previously chosen neighbors)
  - Repeat N_g times
  - $N_g * fanout = log(N)$ [Ganesh 2003]
  - Ignore duplicate messages

Experimental Setup

- Implemented in C
- Use Overnet Availability traces [Bhagwan 2003]
  - Injected as collected
  - 1440 nodes
  - Around 550 nodes online at any time
- Logarithmic Vertical Sliver
- Logarithmic Horizontal Sliver

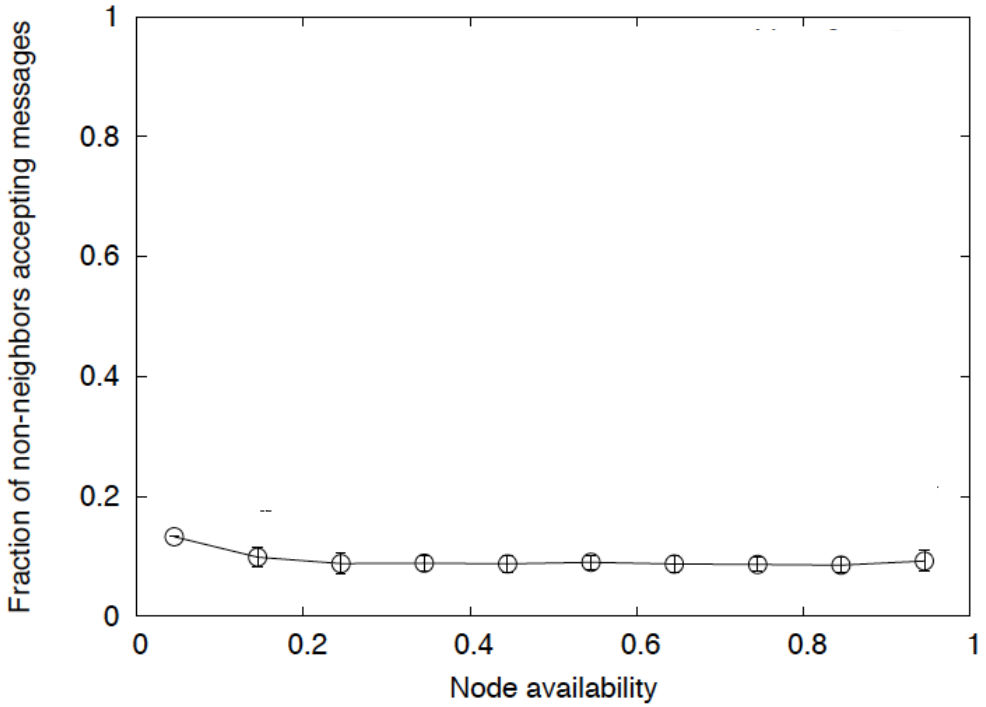
Sliver micro-benchmark



- ***VS(.)*** size is uncorrelated to availability
- Increase in ***HS(.)*** is sublinear

Flooding Attack

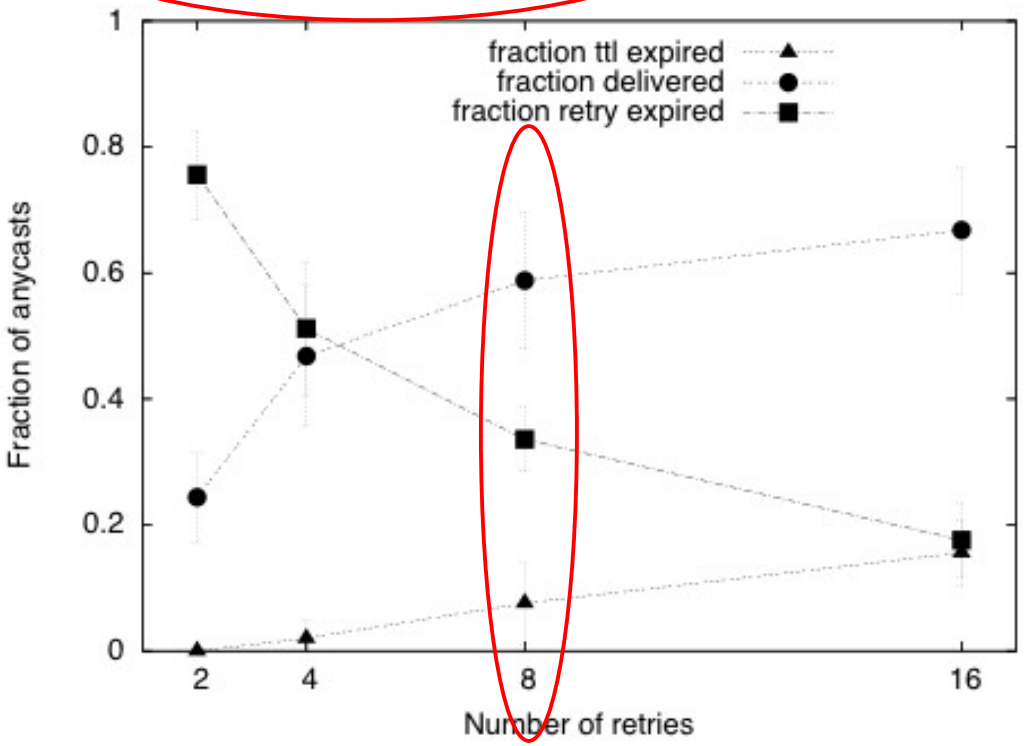
Non-peers accepting incoming:

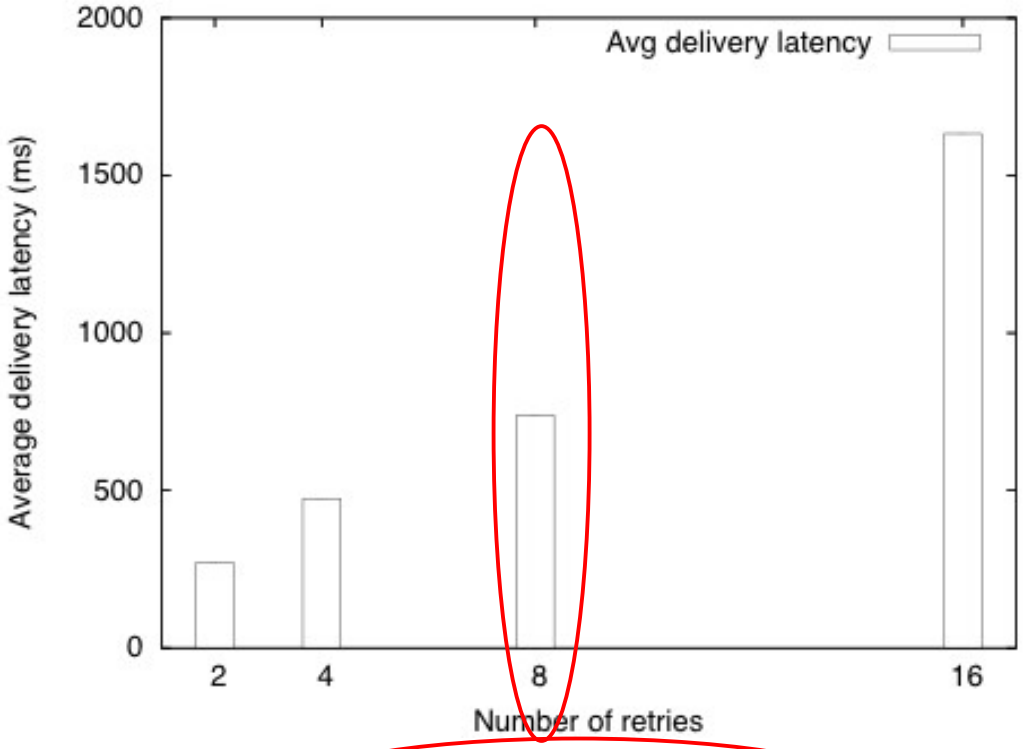


- Fewer than 10% accept incoming selfish msg
- Independent of node's availability

Retried Greedy Anycast

node in HIGH availability
sending to [0.15, 0.25]





latency in [20ms, 80ms]

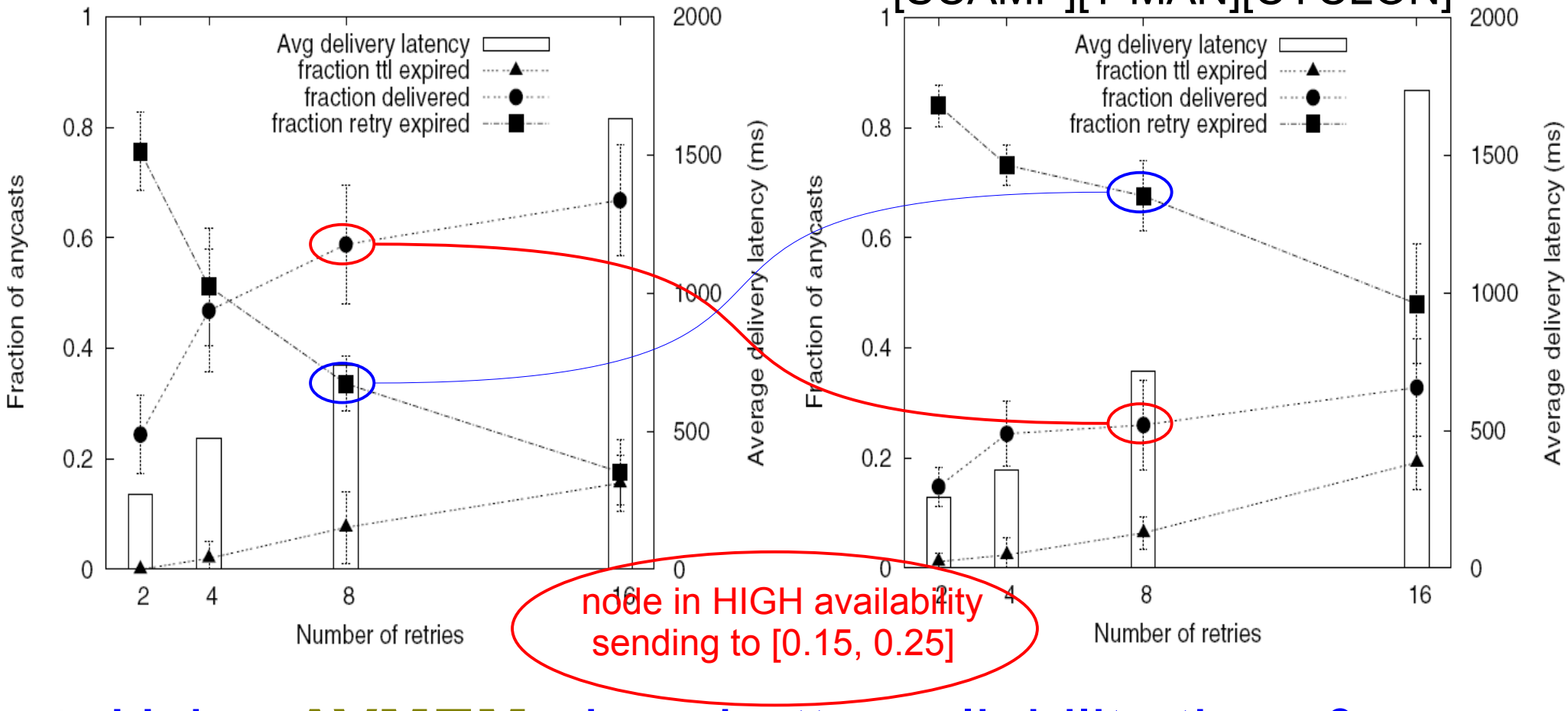
- Reliability improves in harsh scenario
- Reliability improves faster than latency increase

Retried Greedy Anycast

Random Graph

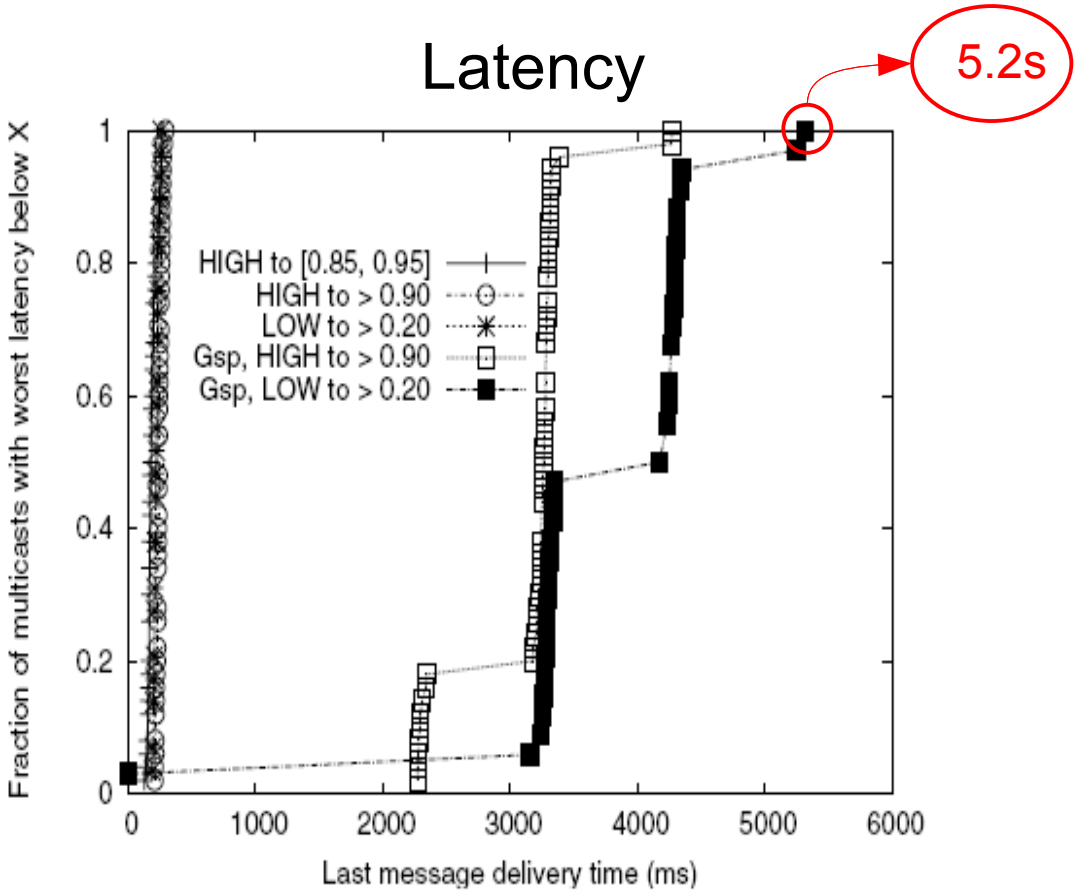
[SCAMP][T-MAN][CYCLON]

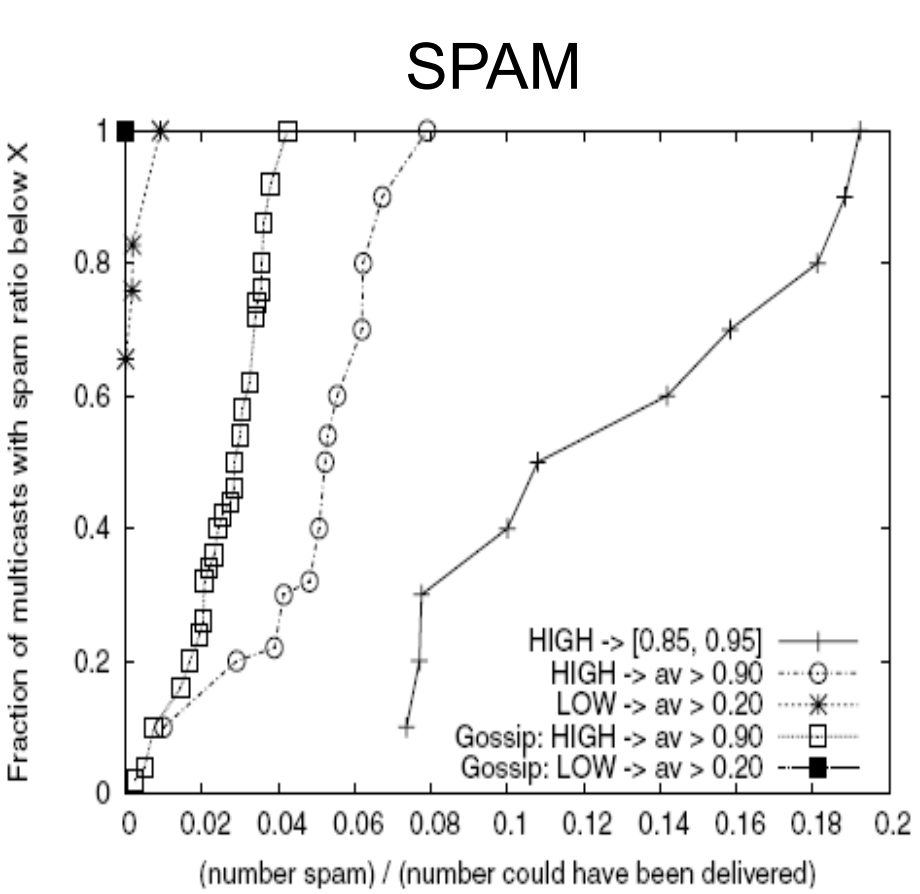
AVMEM



- Using **AVMEM** gives better reliability than & comparable latency to using random overlay

Multicast Efficiency

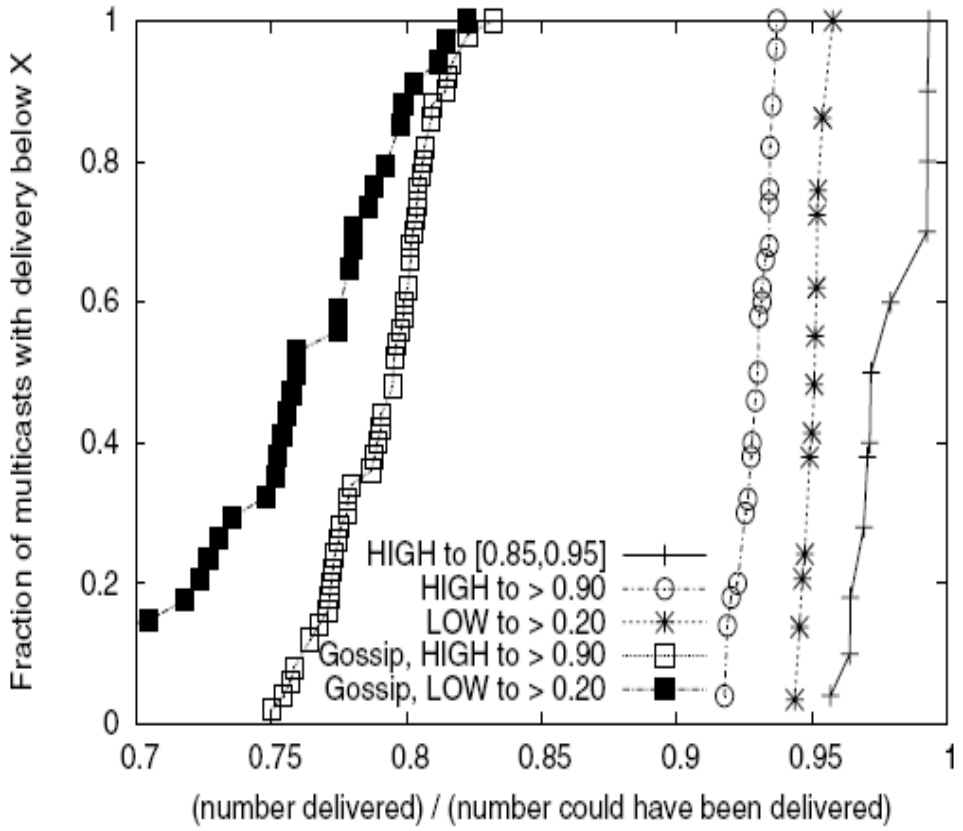




- Flooding is fast <= 300ms
- Gossip saves BW
- SPAM is low < 0.2

simulated latency in [20ms, 80ms]

Multicast Reliability



- Gossip > 70% reliability
- Flooding > 90% reliability

Outline

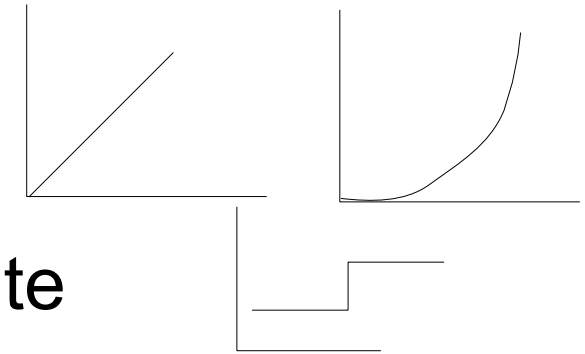
- Introduction
- Availability Monitoring
- Availability-Aware Membership
- **Availability-Aware Aggregation**
  - Motivation
  - Problem Statement
  - Techniques
  - Results
- Conclusion

Motivation

- Aggregation is important in distributed applications
  - resource utilization collection
  - system performance statistics
  - vote collection, for leader election, replication, etc.
- Biased aggregates useful when nodes have varying degree of contribution
  - “Bigger say” if more contribution
- Incentivizes nodes
- Mitigates freeloading

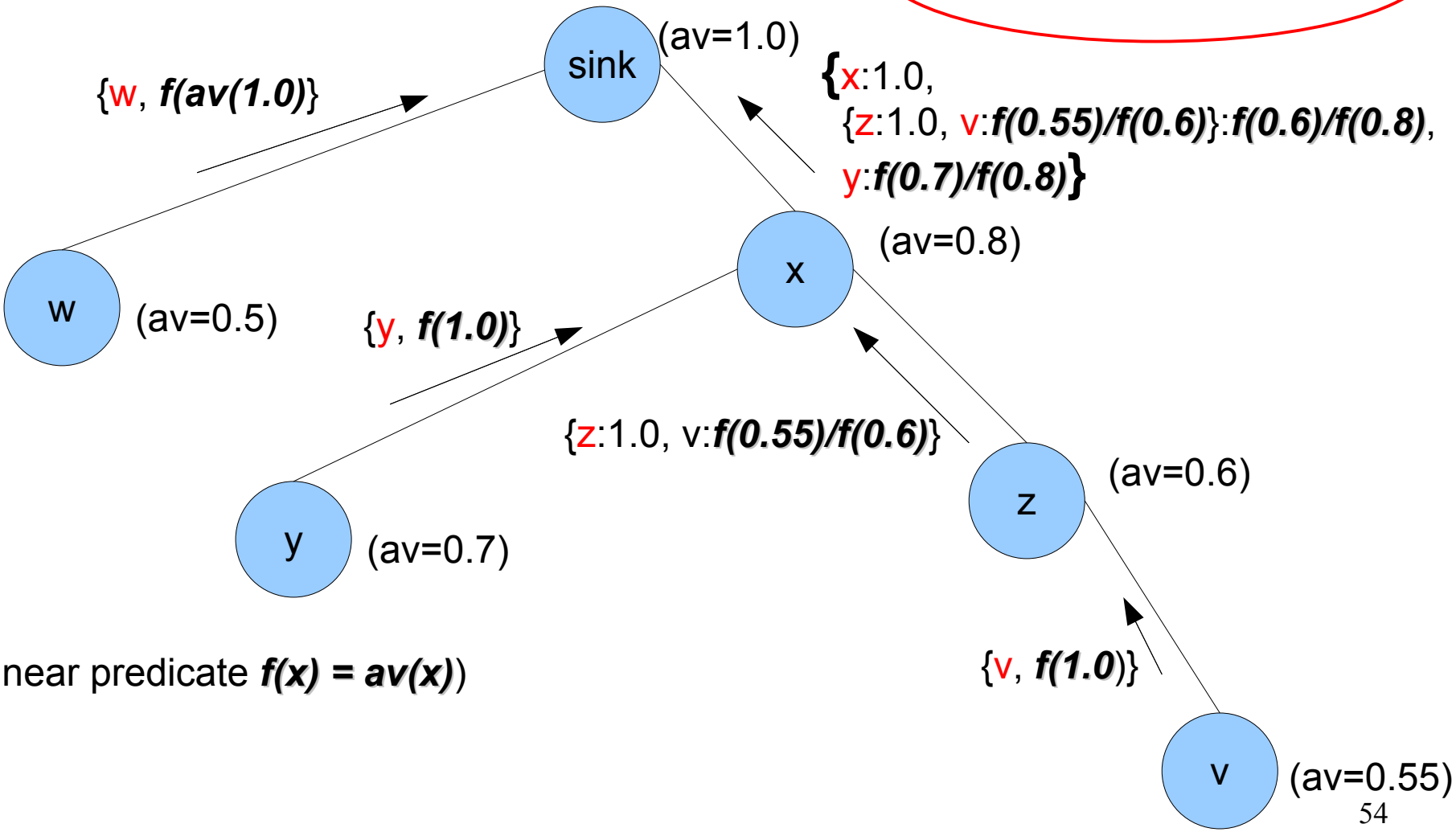
Problem Statement

- For each node, the probability that a global aggregate will include that node's own value is directly proportional to that nodes availability
  - Inclusion probability (***f(x)***)
- Implemented by global predicate
  - ***f(x) = av(x)***, linear predicate
  - ***f(x) = (av(x))^2***, quadratic predicate
  - **if *(av(x) > 0.5) f(x) = 1.0* else *f(x) = 0.0***, bimodal predicate



Availability-Aware Tree Aggregation

{w:0.5/1.0, {x:1.0, y:0.7/0.8, {z:1.0, v:0.55/0.6}:0.6/0.8 }:0.8/1.0 }={w:0.5, x:0.8, y:0.7, z:0.6, v:0.55}

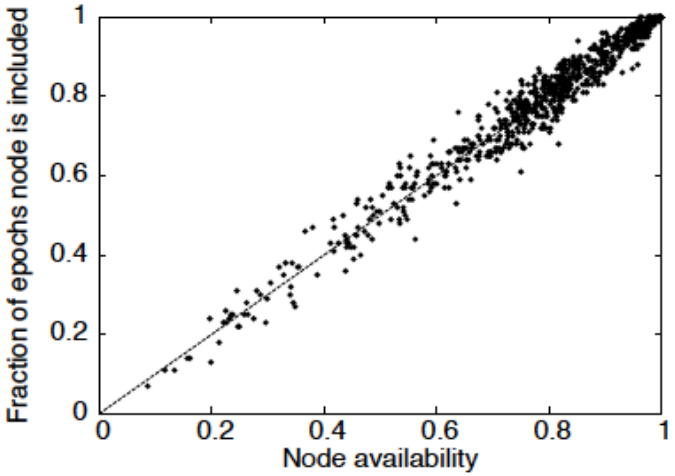


(assume linear predicate ***f(x) = av(x)***)

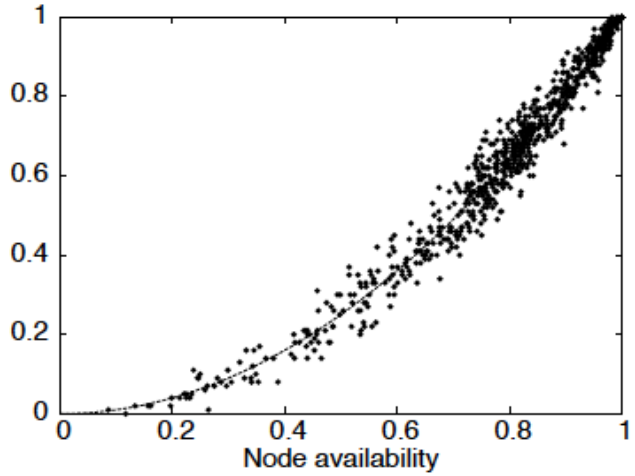
Techniques to Build the Tree

- Child Selection
- Parent Selection
- Auditing
  - To catch greedy selfish nodes

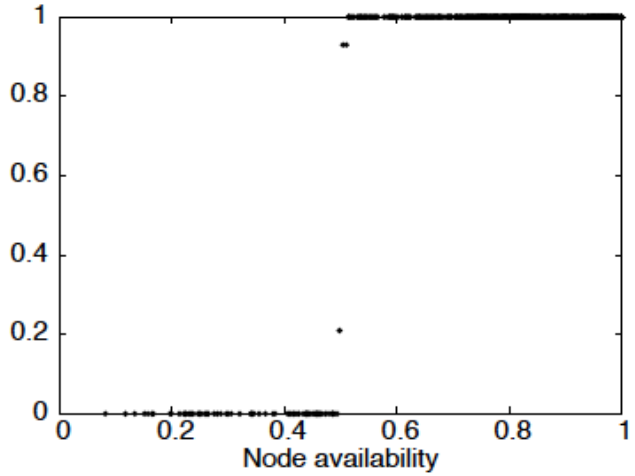
Predicate Satisfaction



(a) $f(x) = av(x)$ (Linear predicate)



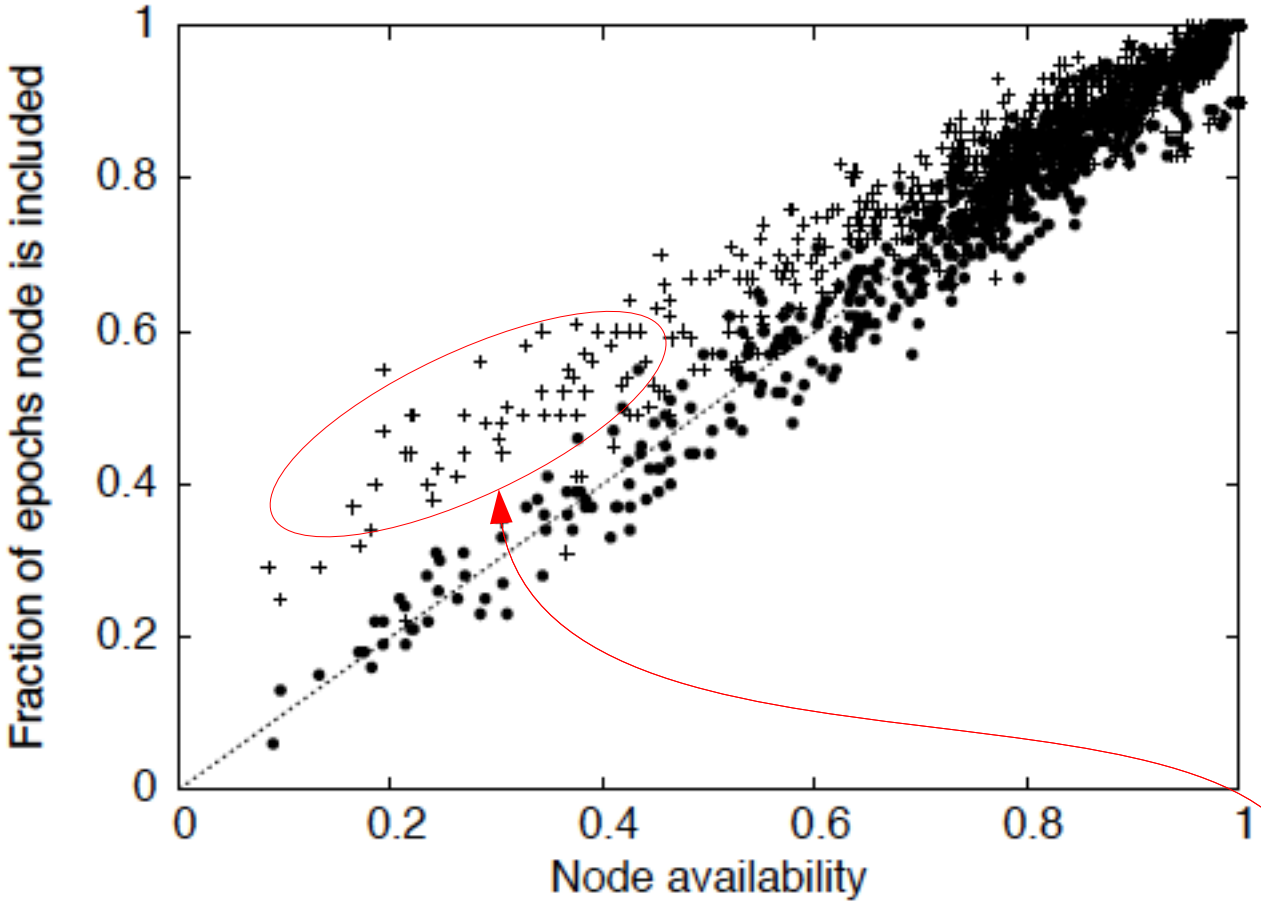
(b) $f(x) = (av(x))^2$ (Quadratic predicate)



(c) `if` $(av(x) > 0.5)$ $f(x) = 1.0$ `else` $f(x) = 0.0$ (Bimodal predicate)

AVCOL properly realizes the global predicates

Effect of Valid Colluders on Predicate



AVCOL's predicate is very resilient to 50% selfish and colluders

Outline

- Introduction
- Availability Monitoring
- Availability-Aware Membership
- Availability-Aware Aggregation
- **Conclusions**

Conclusions

- This thesis has specified availability-dependent global predicates, and shown how they can be efficiently and scalably realized for a class of distributed services, in spite of specific selfish and colluding behaviors, using local and decentralized protocols.

Conclusions

- **AVMON**, the first availability-monitoring overlay
  - efficient
  - scalable
  - churn resilient
  - fast discovery
- **AVMEM**, the first availability-aware overlay
  - efficiently uses availability-dependent predicates to manage peer relationships
  - tolerates selfish nodes
  - range/threshold multicast/anycast can be effective and efficient

Conclusions

- **AVCOL**, the first availability-aware aggregation system
  - efficiently implements availability-dependent predicates, relating a node's availability to its inclusion probability
  - predicates are realized in spite of selfish and colluding nodes

Thank you!