

# SAAR: A Shared Control Plane for Overlay Multicast

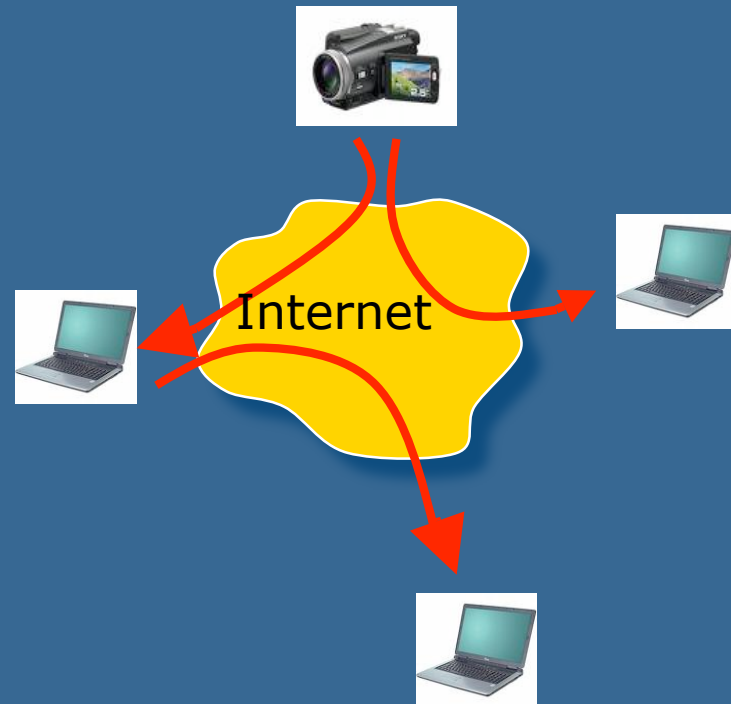
**Animesh Nandi**

Aditya Gangam, Peter Druschel, T.S. Eugene Ng,  
Ion Stoica, Hui Zhang, Bobby Bhattacharjee

(MPI for Software Systems, Rice University, CMU,  
UC Berkeley, University of Maryland)

# Overlay Multicast

**Endsystems cooperatively disseminate content**



# Applications

- **Webcast special events**

e.g. ESM, TMesh ...

- **P2P-Internet Television**

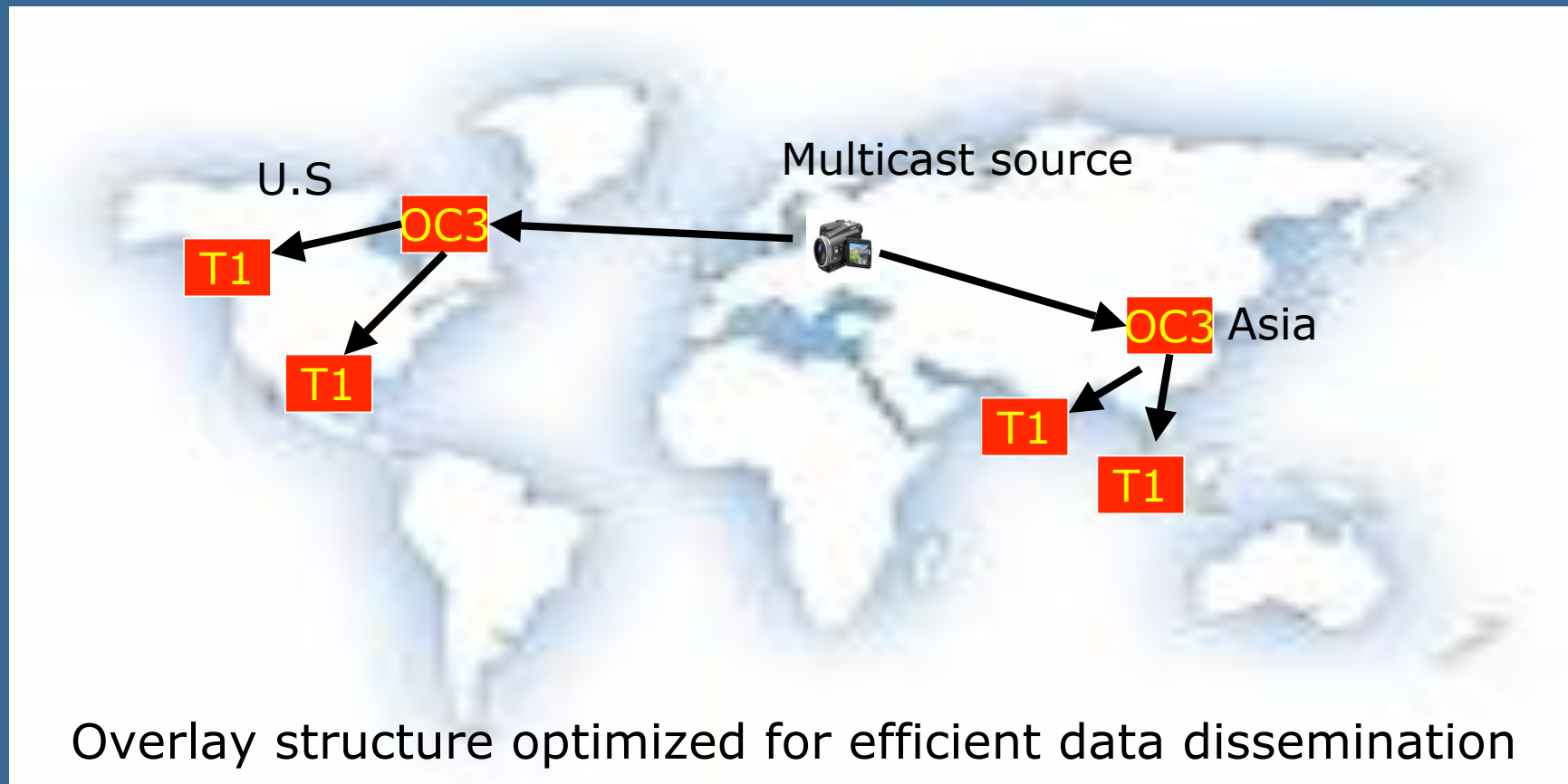
e.g. Coolstreaming, PPLive, Sopcast ...

# Why another Overlay Multicast paper?

## Architectural insight:

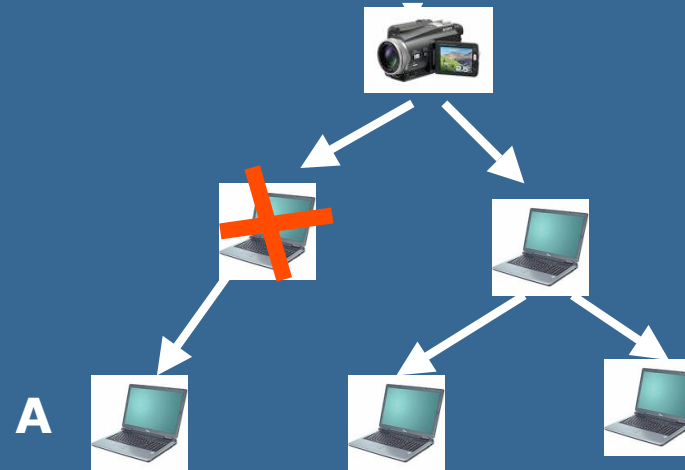
- separate overlay, optimized for control
- shared among many data overlays
- benefits single-tree, multi-tree, mesh

# Data dissemination



# Control Mechanism

**Employs control mechanisms to build/repair overlay structure**

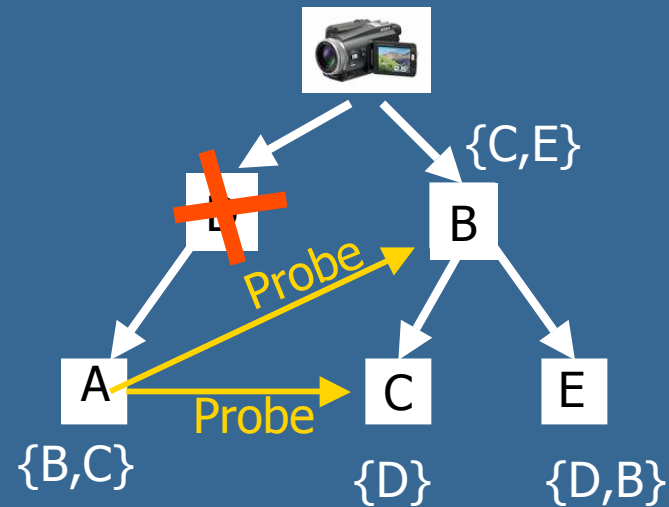


# ESM's Data/Control

**Data overlay optimized for latency/bandwidth**

**Gossip to distribute membership info**

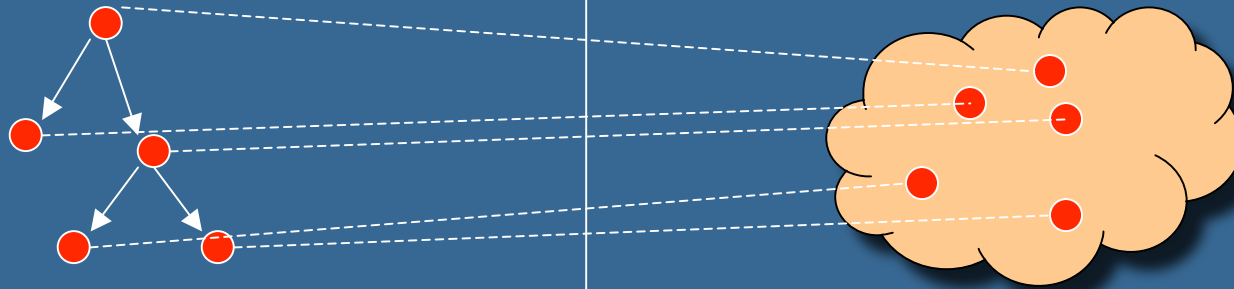
**Probing to select parents**



**Does not scale to large groups and high membership churn**

**Summary: efficient data plane structure but unscalable control**

# Idea: Decouple control/data into separate overlays

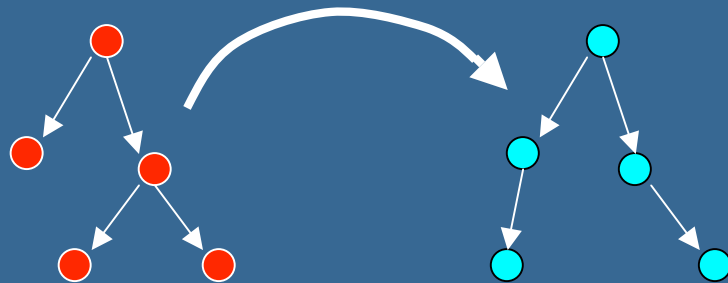


**Dataplane optimized  
for data dissemination**

**Separate overlay  
provides efficient control**

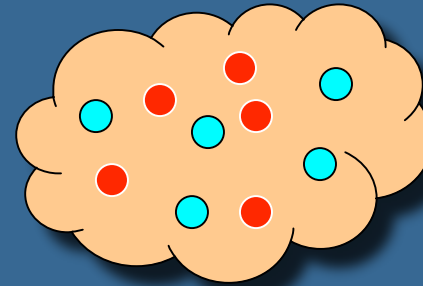
**Summary: Using separate overlays avoids efficiency tradeoffs**

# Sharing control overlay



**BBC**

**CNN**



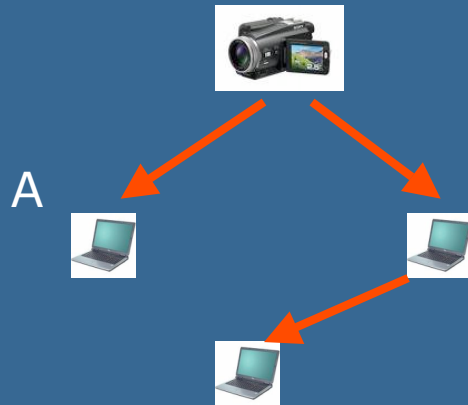
**Shared overlay  
provides  
efficient control**

**Channel switching does not  
affect control overlay  
membership**

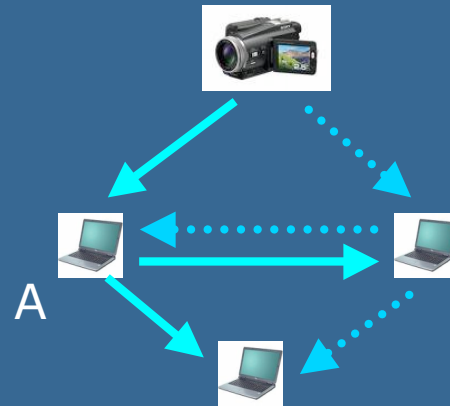
**Summary: Sharing reduces control overlay churn.**

# Neighbor Acquisition

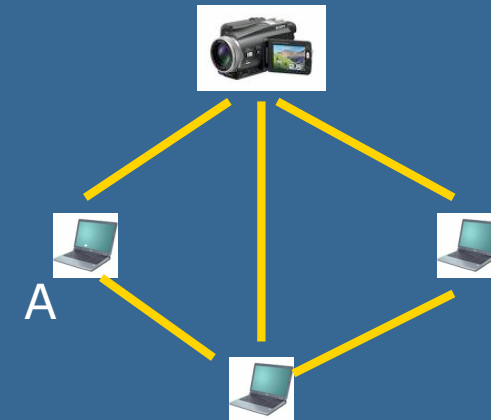
single-tree



multi-tree



mesh

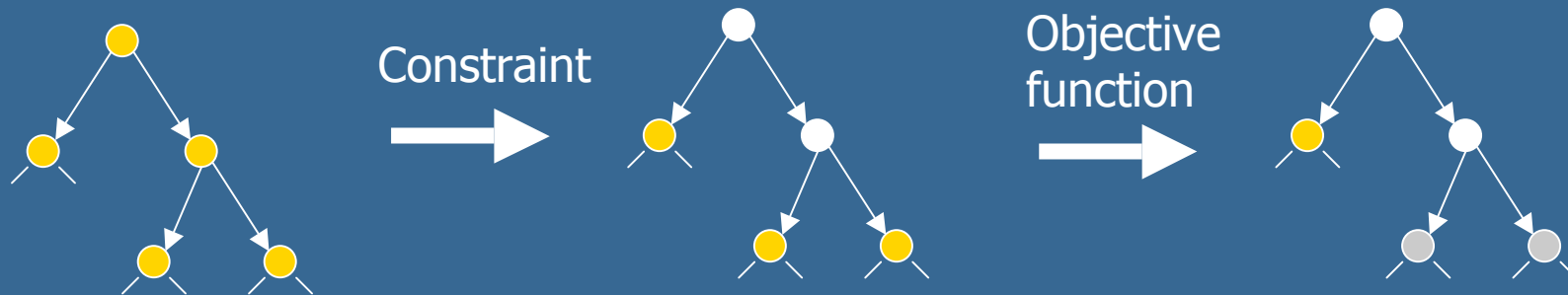


**Summary: Neighbor acquisition is fundamental control task**

# Anycast Primitive for Neighbor Acquisition

anycast (groupId  $g$ , constraint  $p$ , objective function  $o$ )

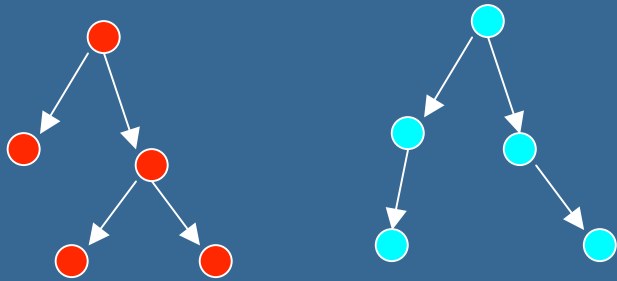
(Amongst the members of  $g$  satisfying  $p$ , chooses the one that maximizes  $o$ )



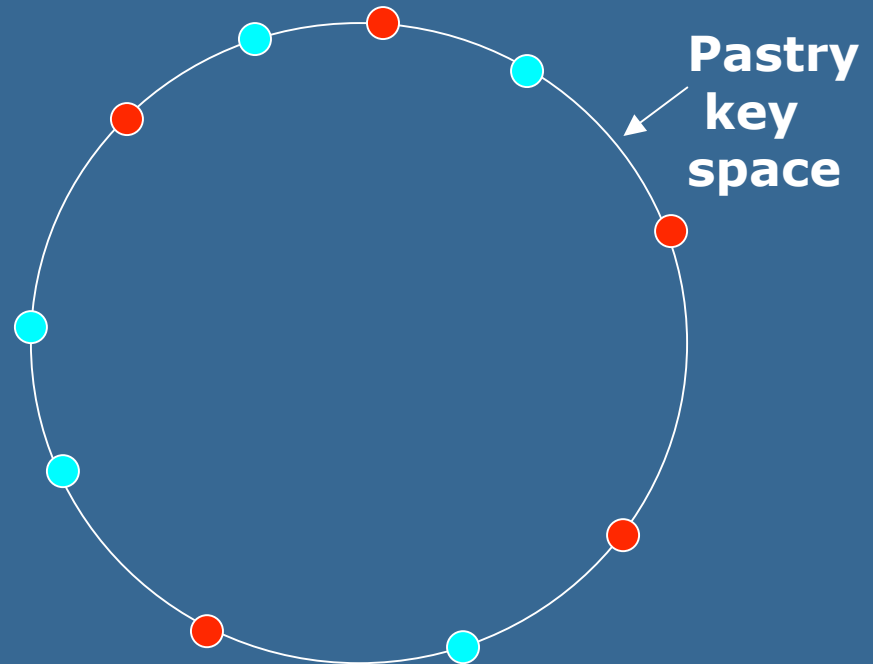
**Summary: Anycast primitive used to build different overlays**

**Question: Can the control overlay  
efficiently implement this anycast  
primitive ?**

# Idea: Use structured overlay

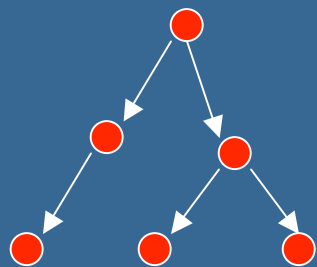


**Data planes optimized  
for efficient data  
dissemination**



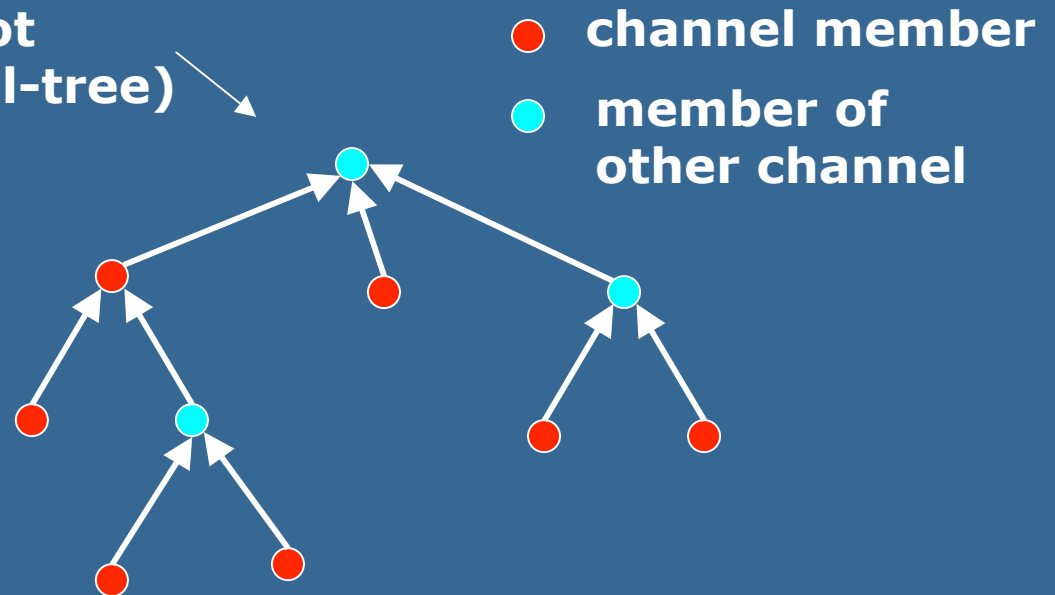
**Shared structured overlay  
provides efficient  
anycast service**

# SAAR: Per-channel control tree



Dataplane View

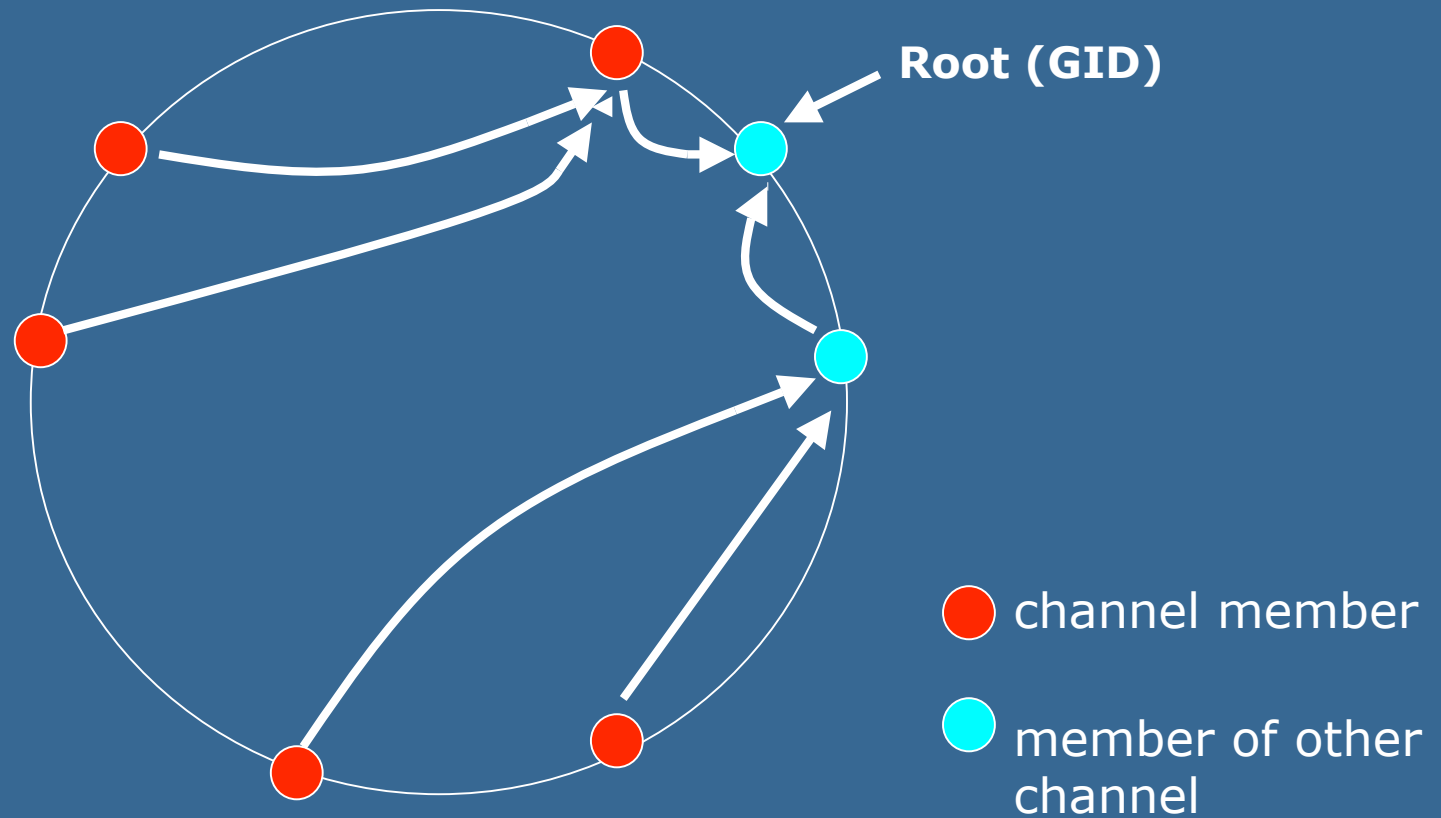
Root  
(Control-tree)



Control plane view

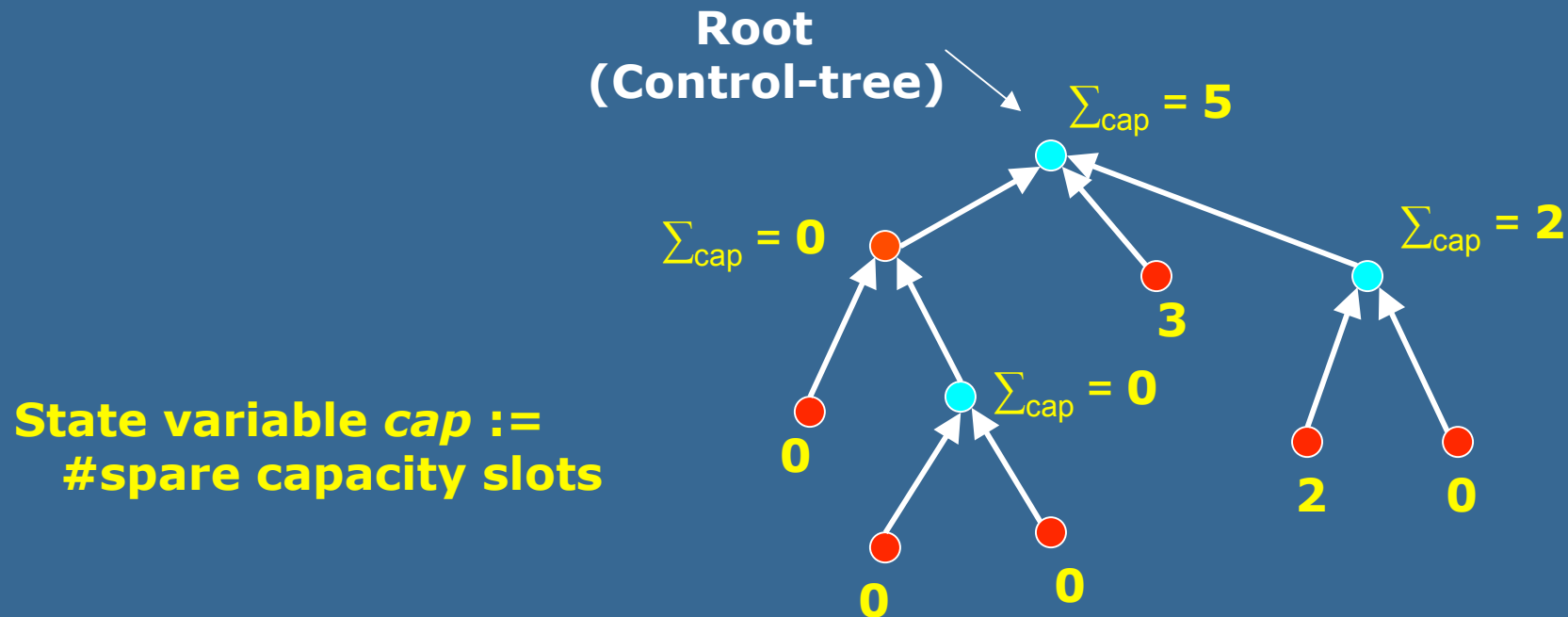
**Control tree's structure is optimized for control and independent of data overlay structure.**

# Control tree formation



**Tree structure formed efficiently**

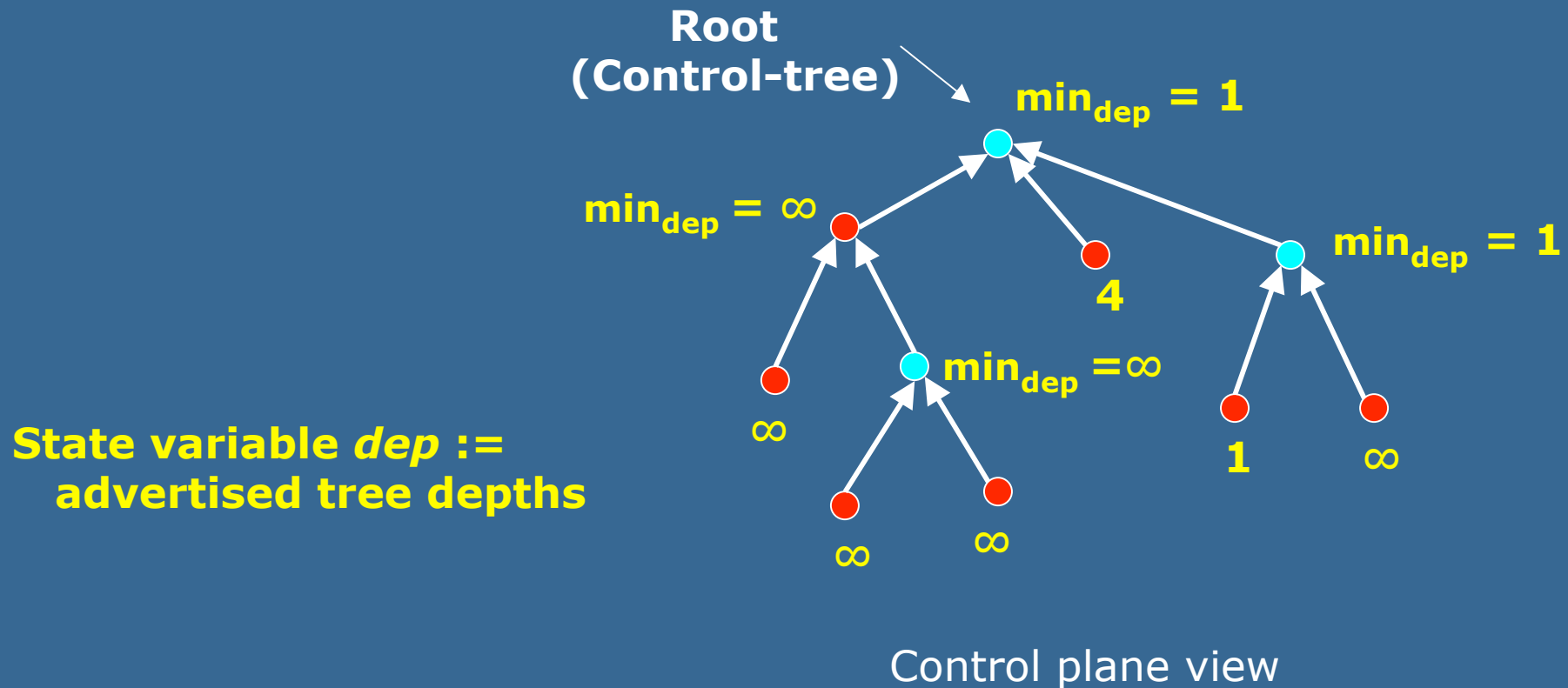
# SAAR: Aggregation



Control plane view

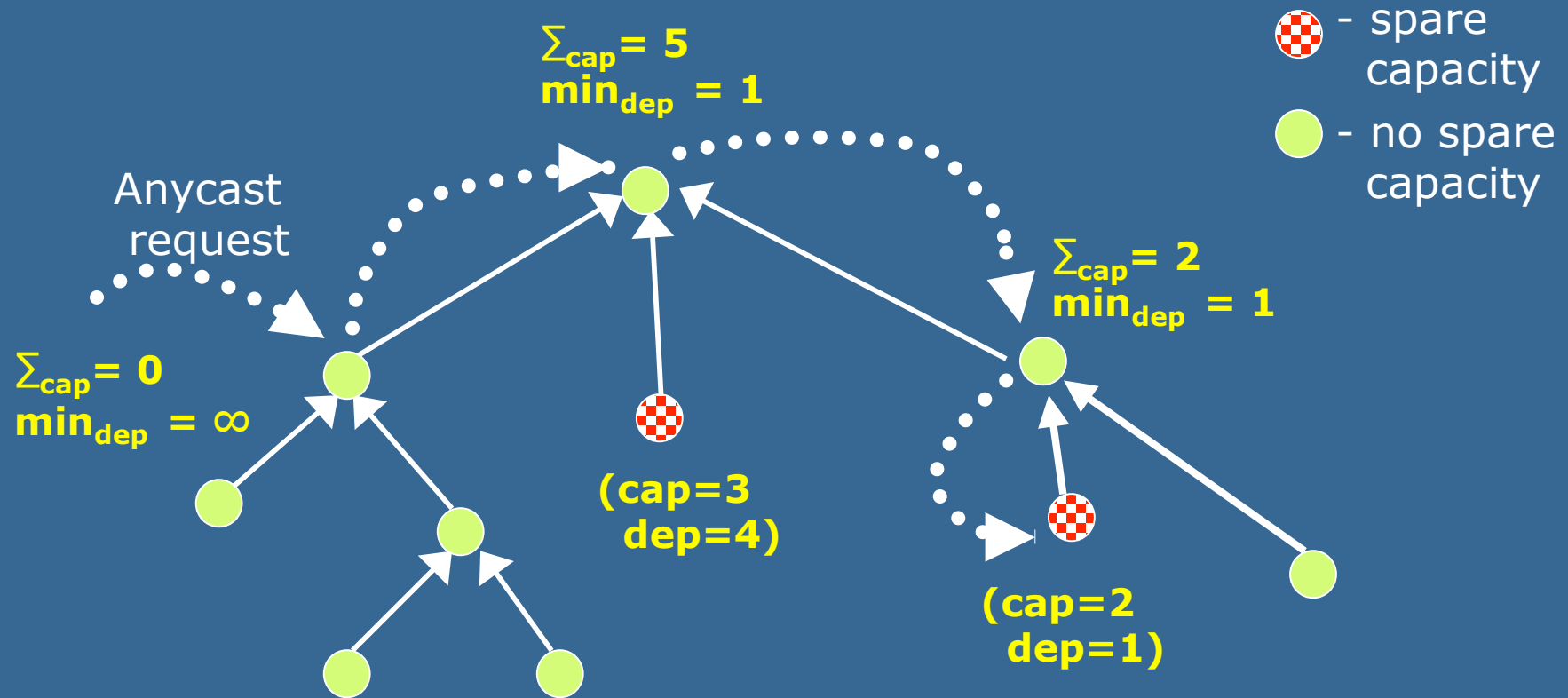
- channel member
- member of other channel

# SAAR: Aggregation



- channel member
- member of other channel

# SAAR: Efficient search

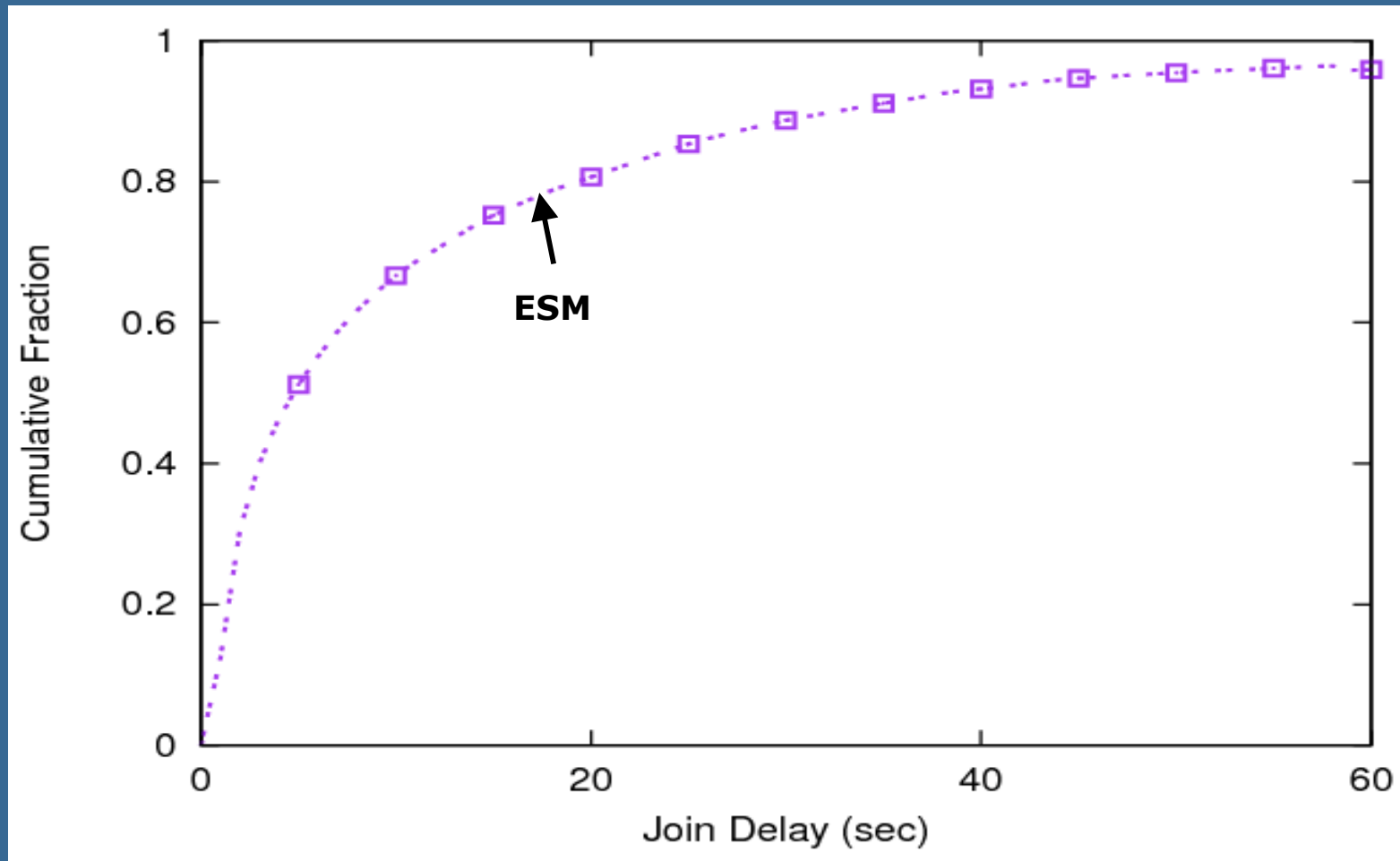


Aggregation enables efficient Depth-First-Search of control tree

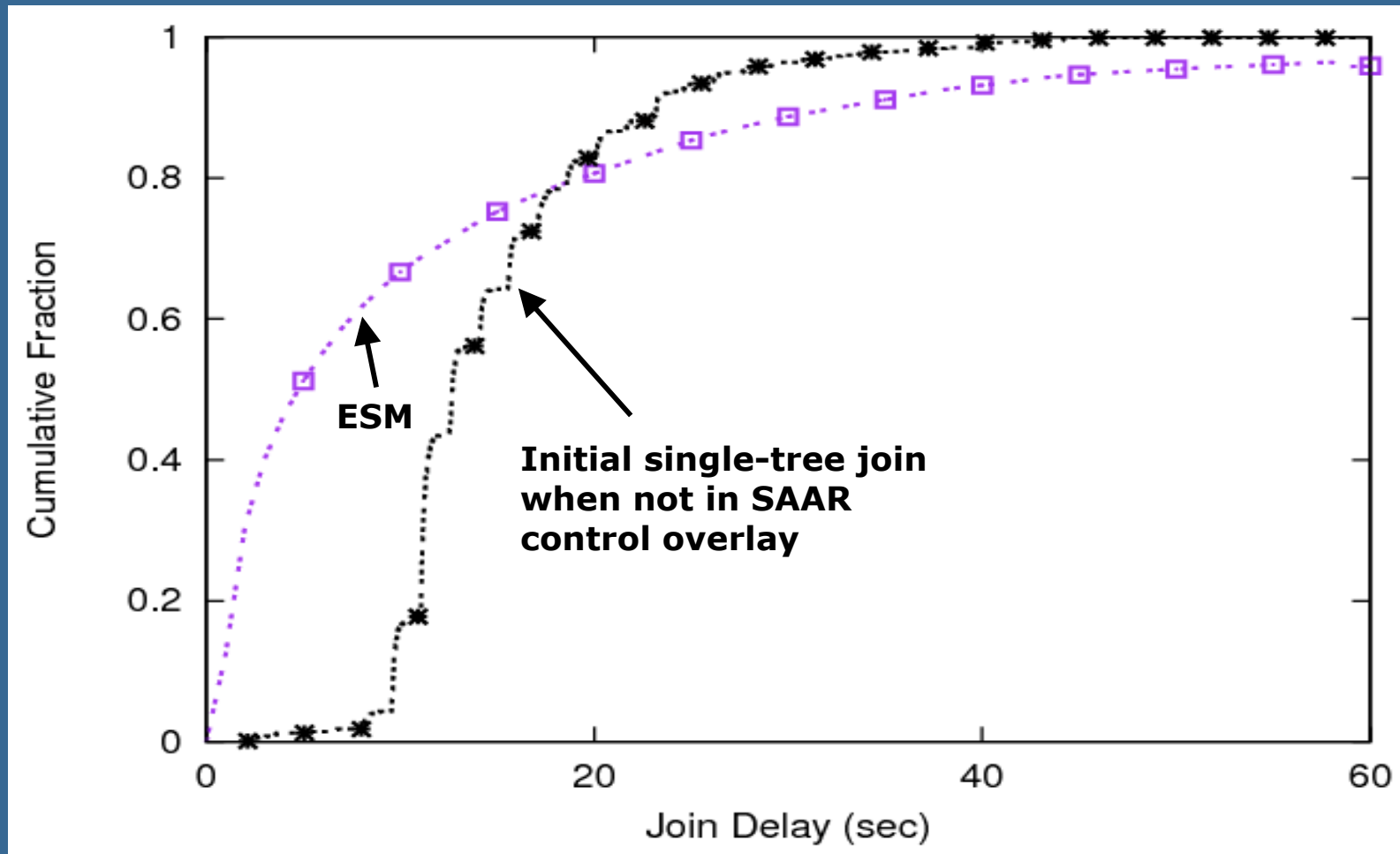
# Evaluation

- Modelnet
- 350 nodes
- heterogeneous forwarding bandwidths
- mean stay time of 5 minutes (exponential distribution)
- Metrics: channel switching delay  
streaming quality  
control overhead

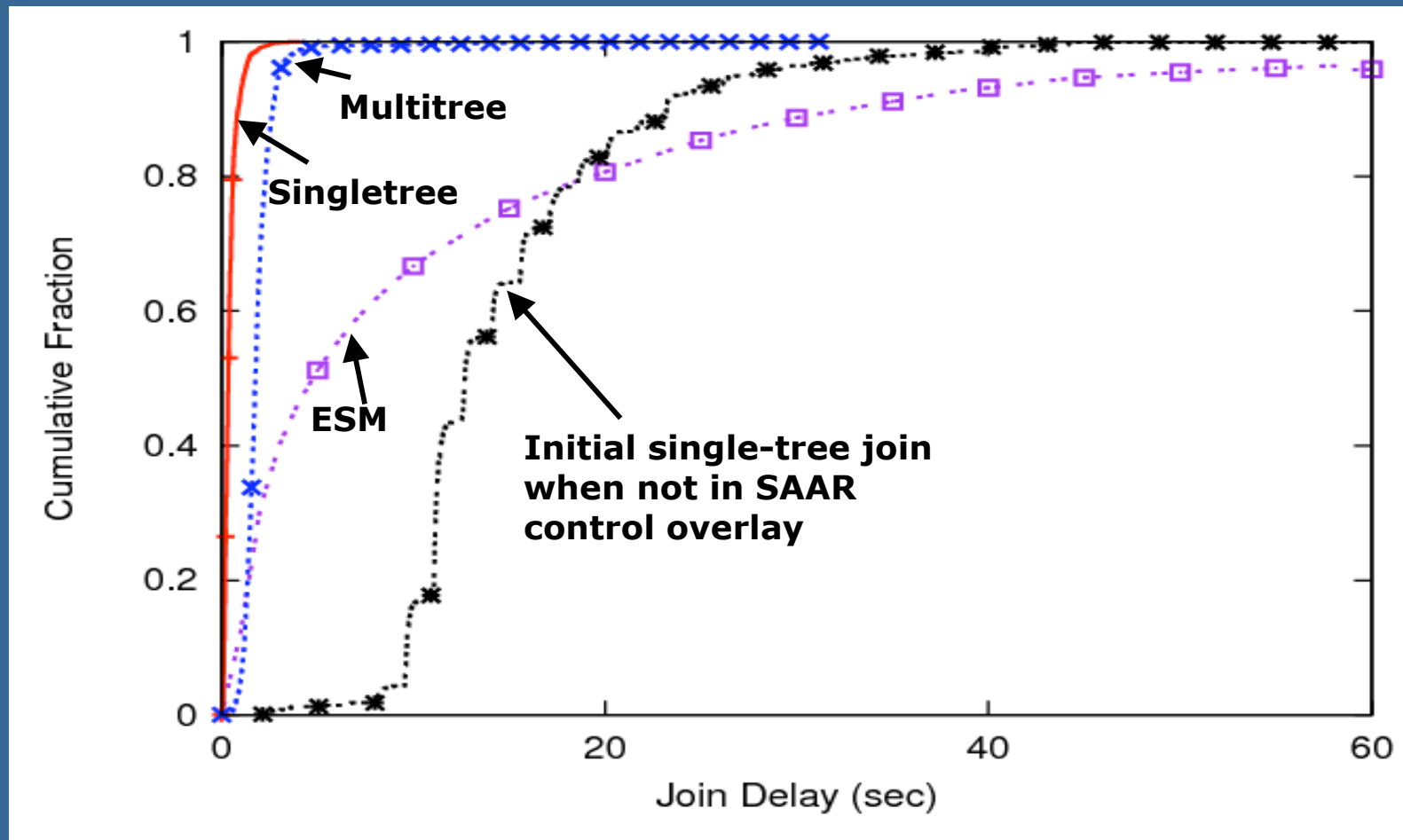
# SAAR enables low join delays



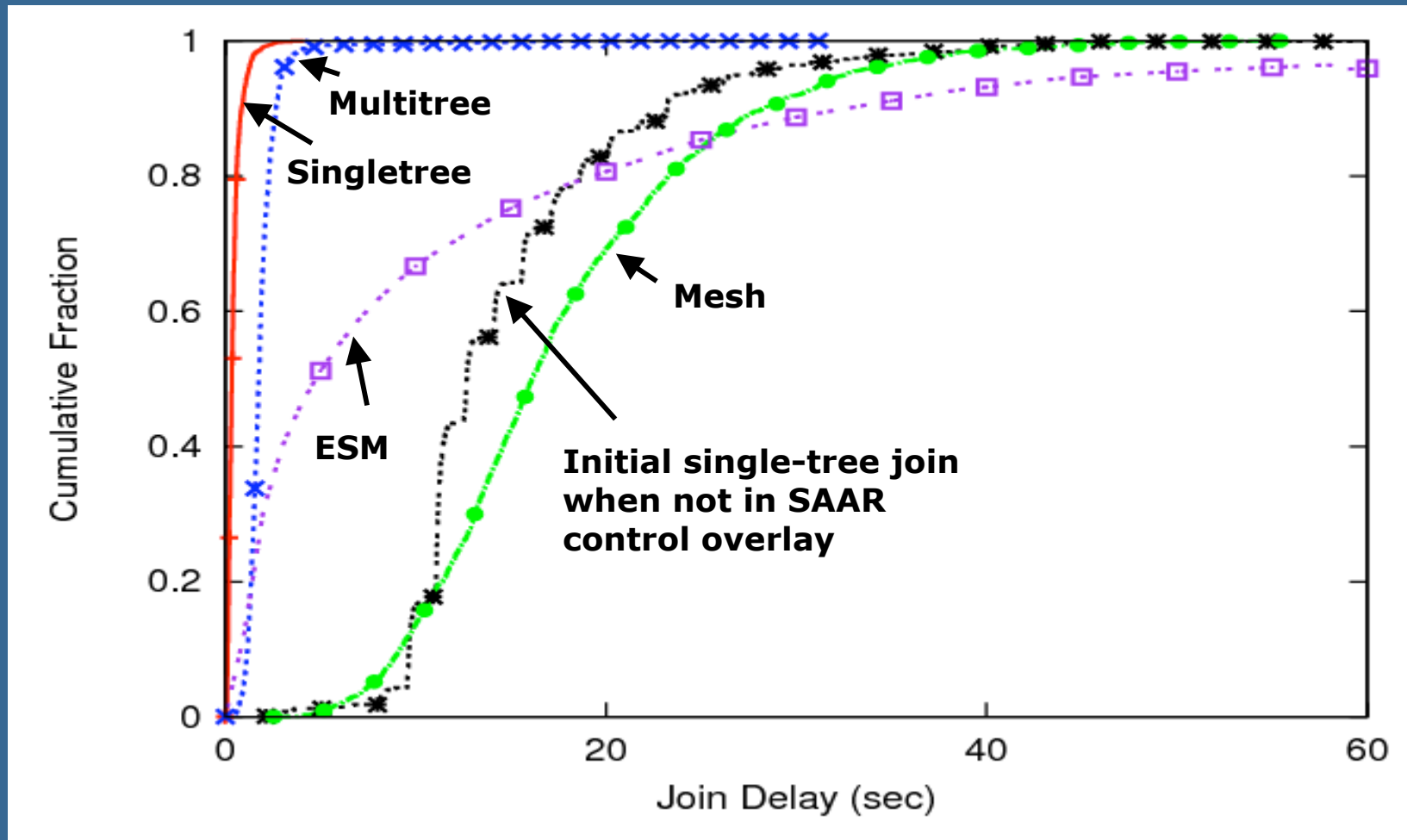
# SAAR enables low join delays



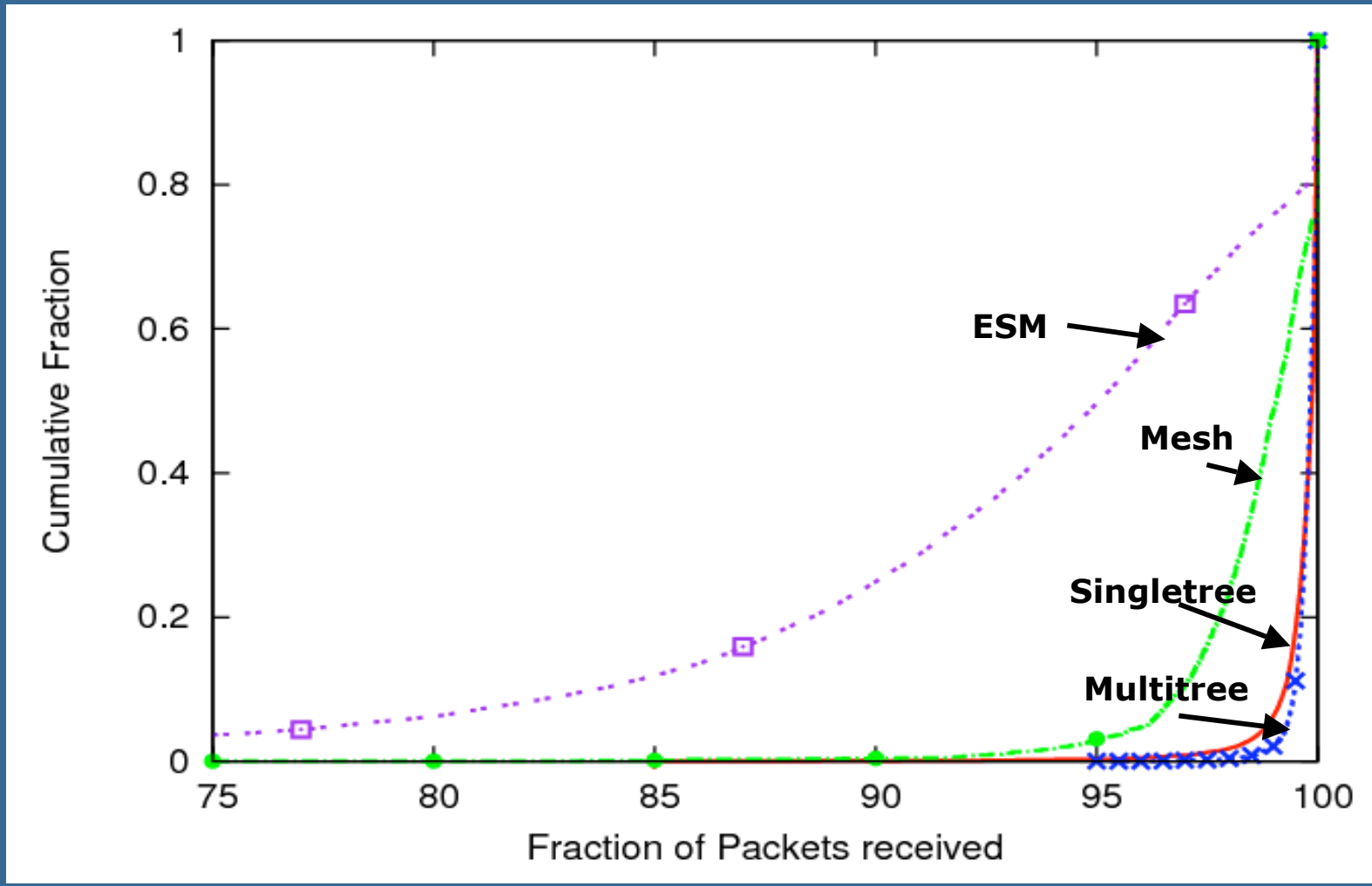
# SAAR enables low join delays



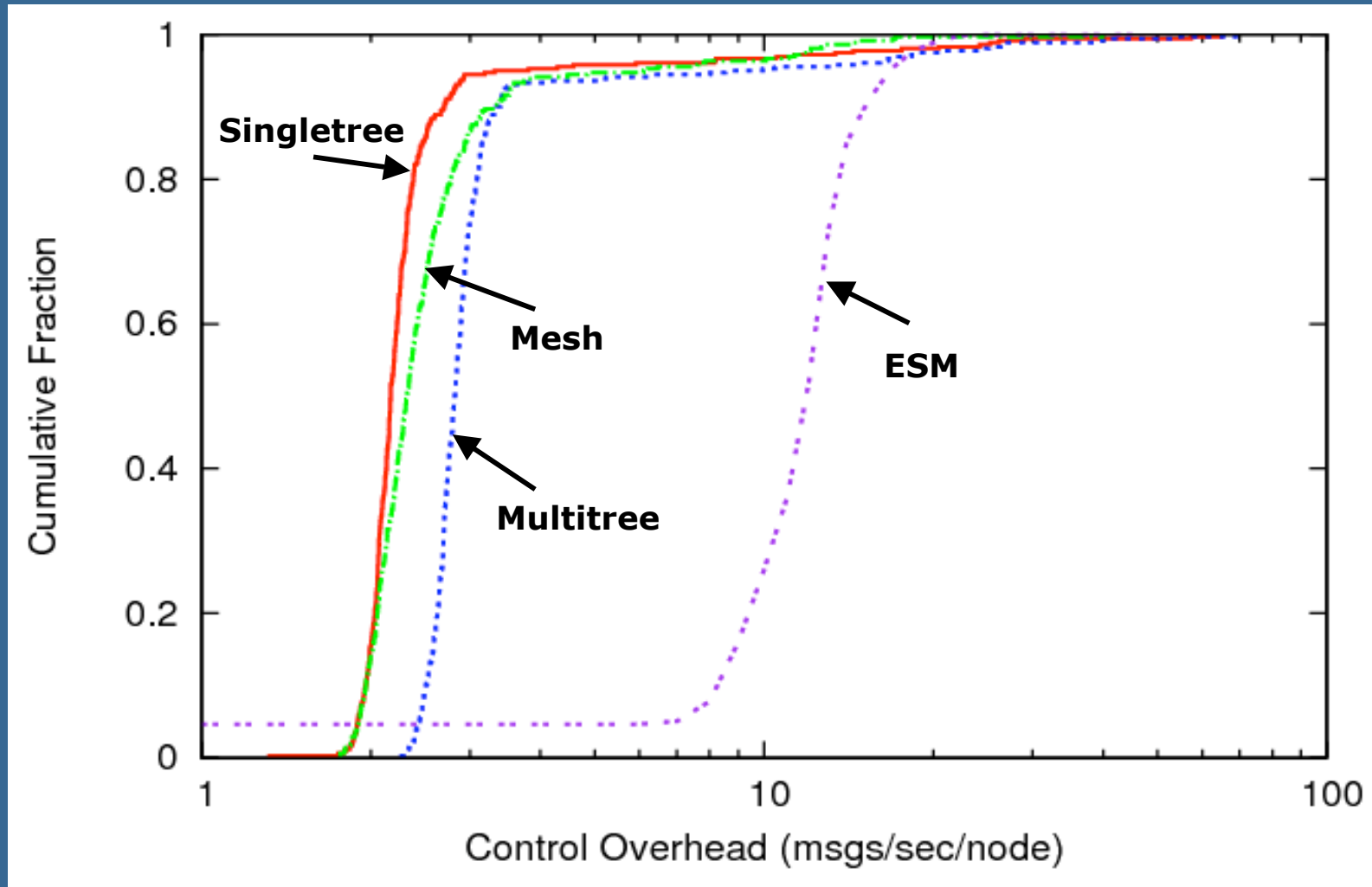
# SAAR enables low join delays



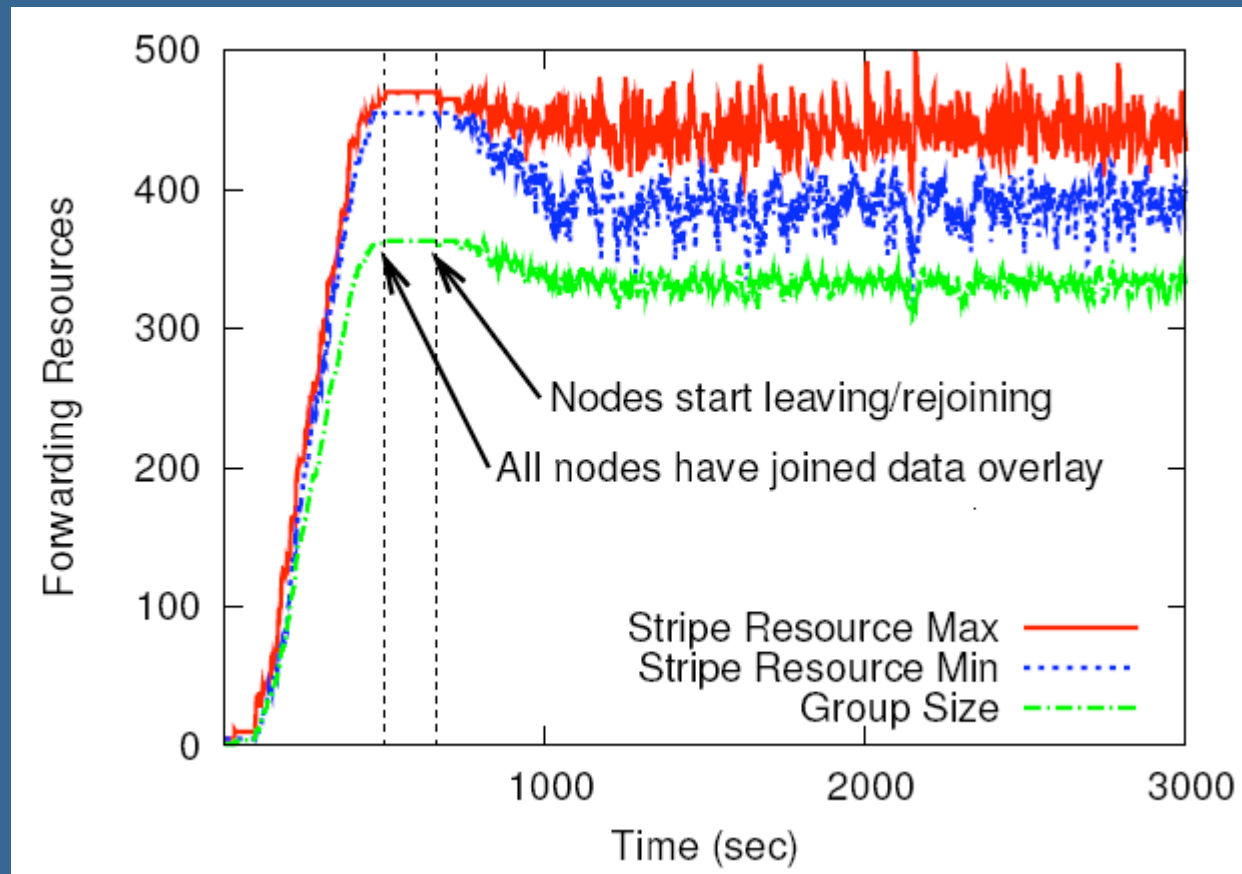
# SAAR: good streaming quality



# SAAR: low control overhead



# SAAR supports complex overlays



Multi-tree, 5 stripes, 350 nodes, 2-min mean churn (exp.), RI=1.23

# More results in the paper

- Flash crowd conditions
- Higher churn (2 min mean, exp. dist)
- Control churn
- Planetlab experiments

# Conclusions

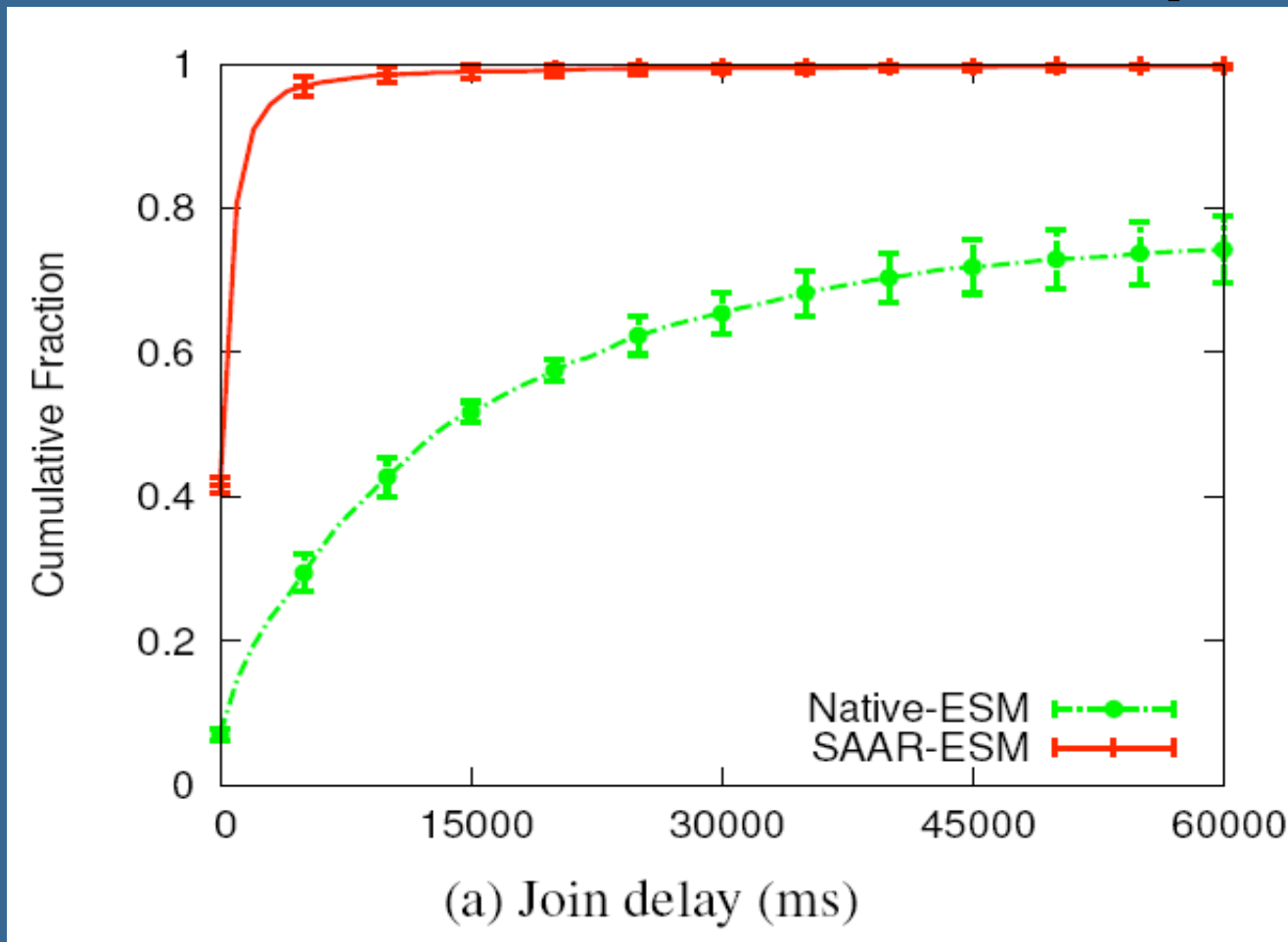
- Control efficiency is key to performance of overlay multicast systems, especially tree-based systems.
- Anycast based on structured overlays enables powerful, efficient control for tree-based systems.
- Fast channel switching can be achieved with tree-based overlay multicast systems.

**THANK YOU**

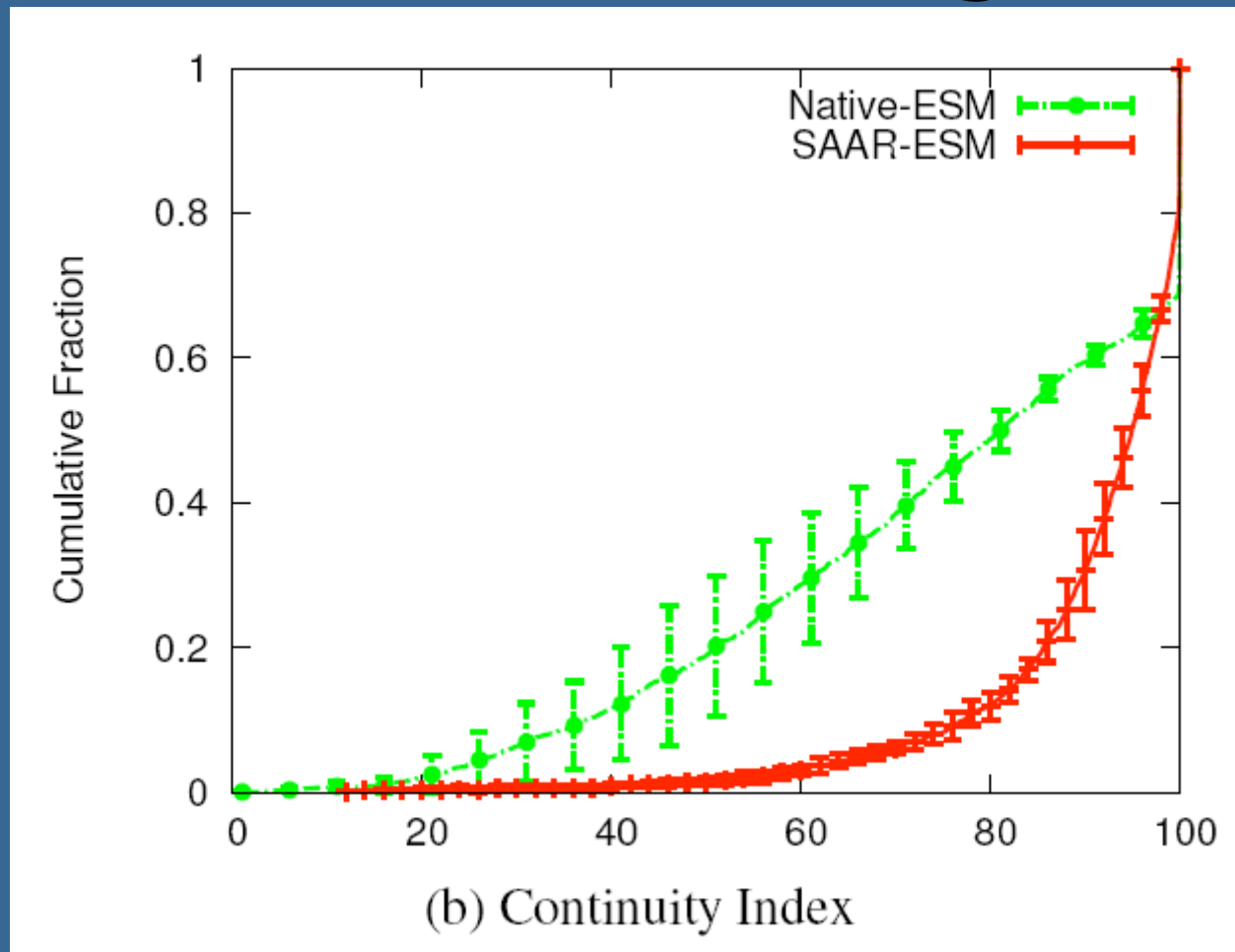
Animesh Nandi

# Backup Slides

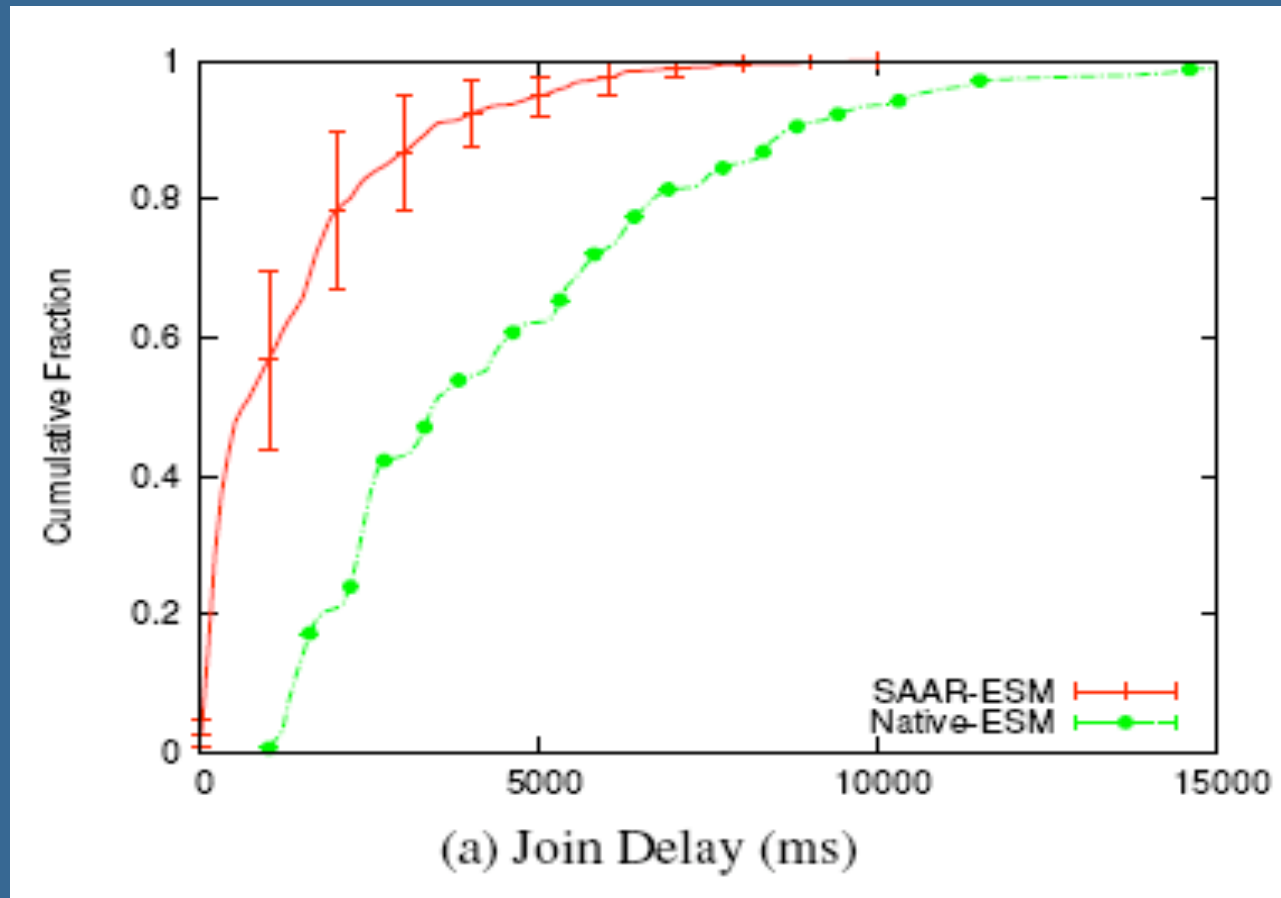
# Planetlab: Join delays



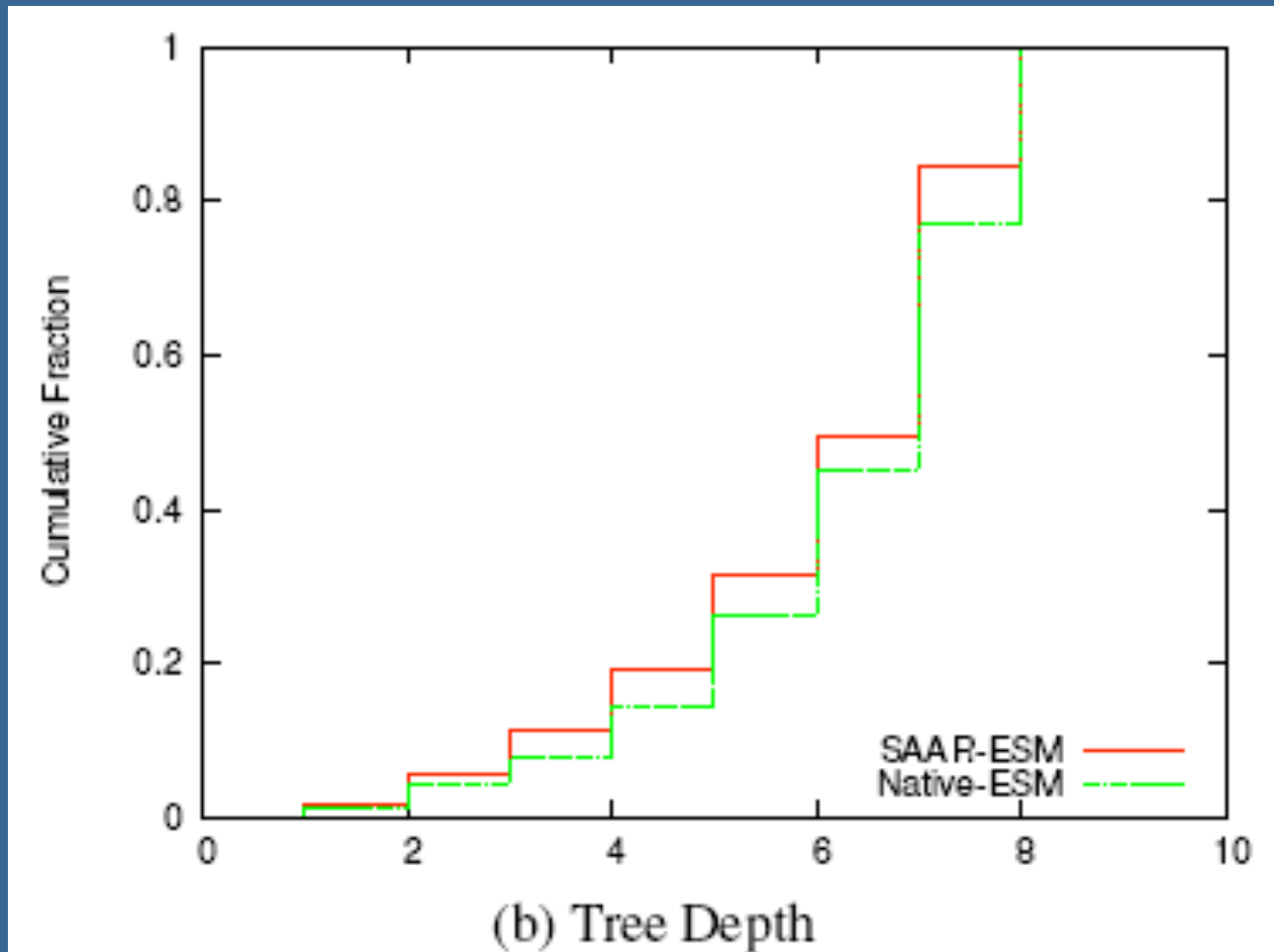
# Planetlab: Streaming Quality



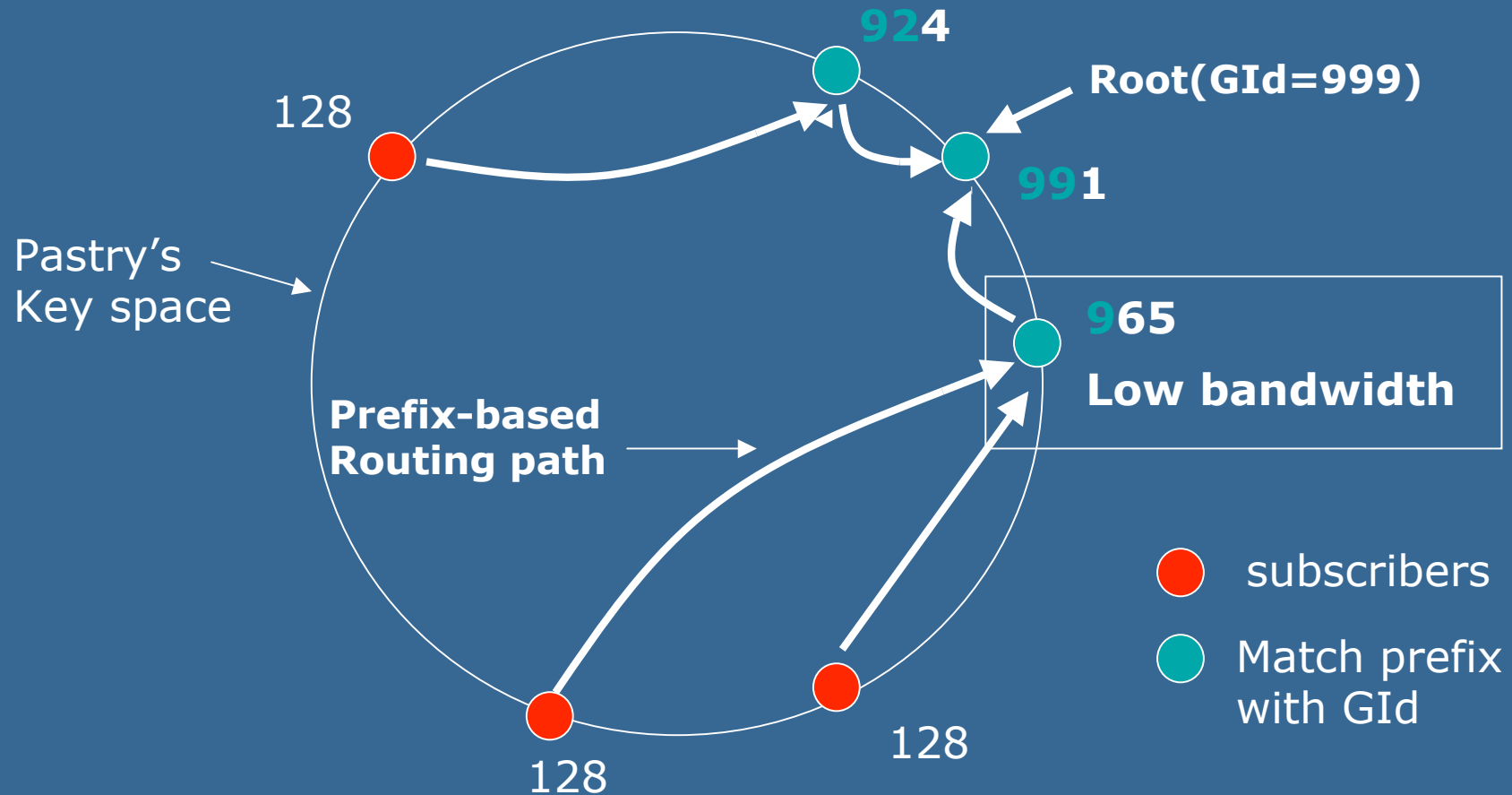
# Flash crowd: Join Delays



# Flash crowd: Tree Depths

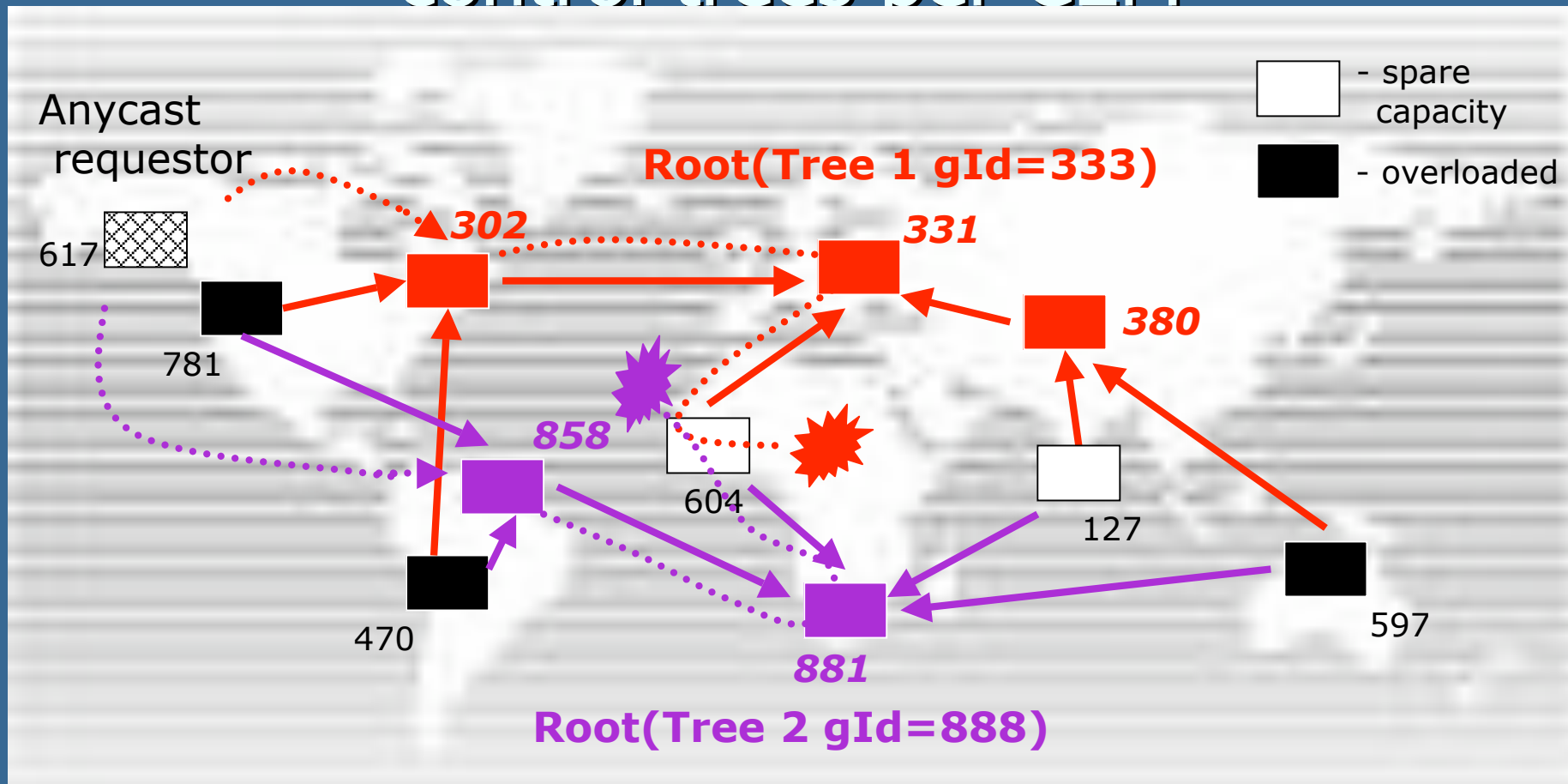


# eg. Scribe's Control/Data



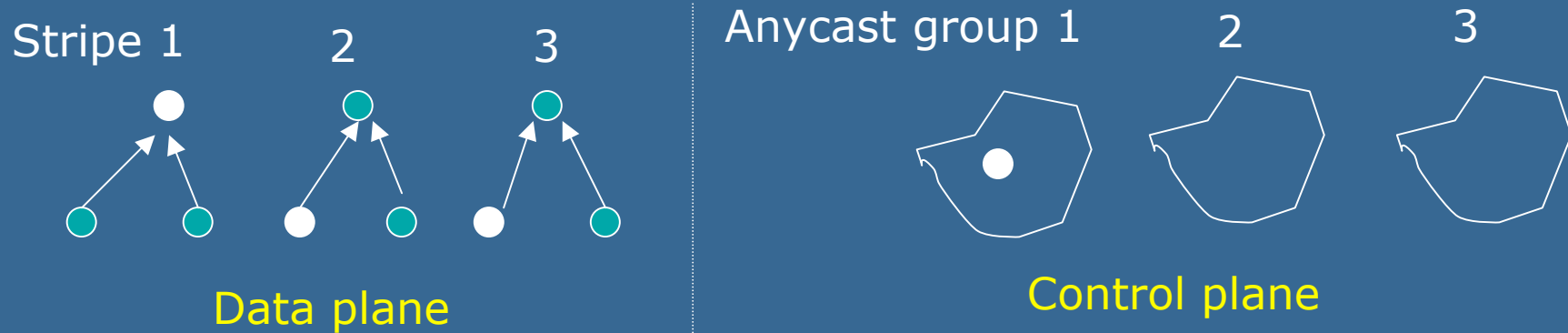
**Scribe has efficient control but inefficient data paths**

# SAAR Anycast: Interior-node-disjoint control trees per CEM



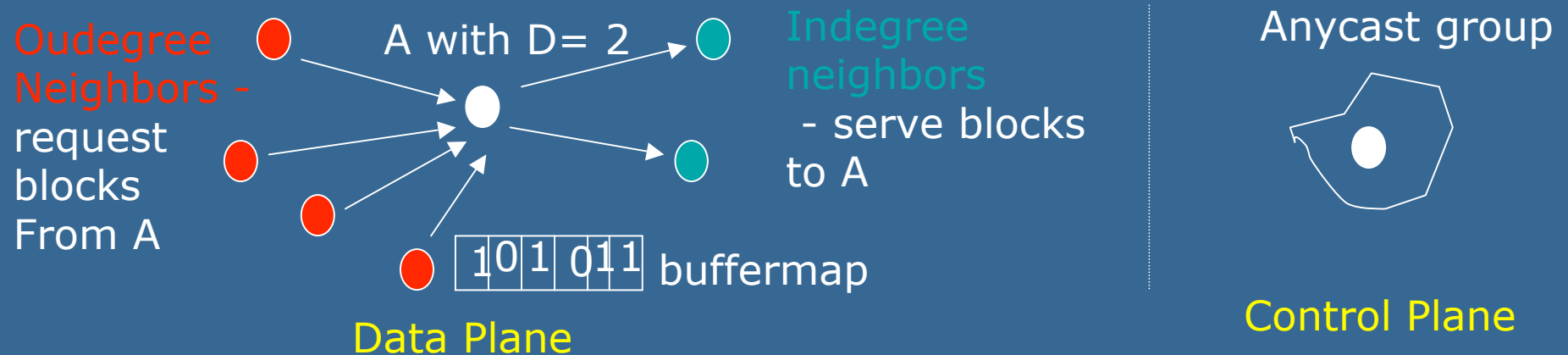
Summary: SAAR provides uninterrupted control in the face of control overlay churn.

# Enabling multi-tree data plane



- Attempts to build  $k$  interior-node disjoint stripe trees.
- Node anycasts to establish parents in each stripe, but joins only one anycast group corresponding to its primary stripe.
- Resource based primary stripe selection: `groupAggregateRequest (groupId, state-variable)`
- Same predicate and objective function as in single-tree for each stripe tree anycasts
- **Summary** : multi-tree policy expressed easily.

# Enabling block-based data plane

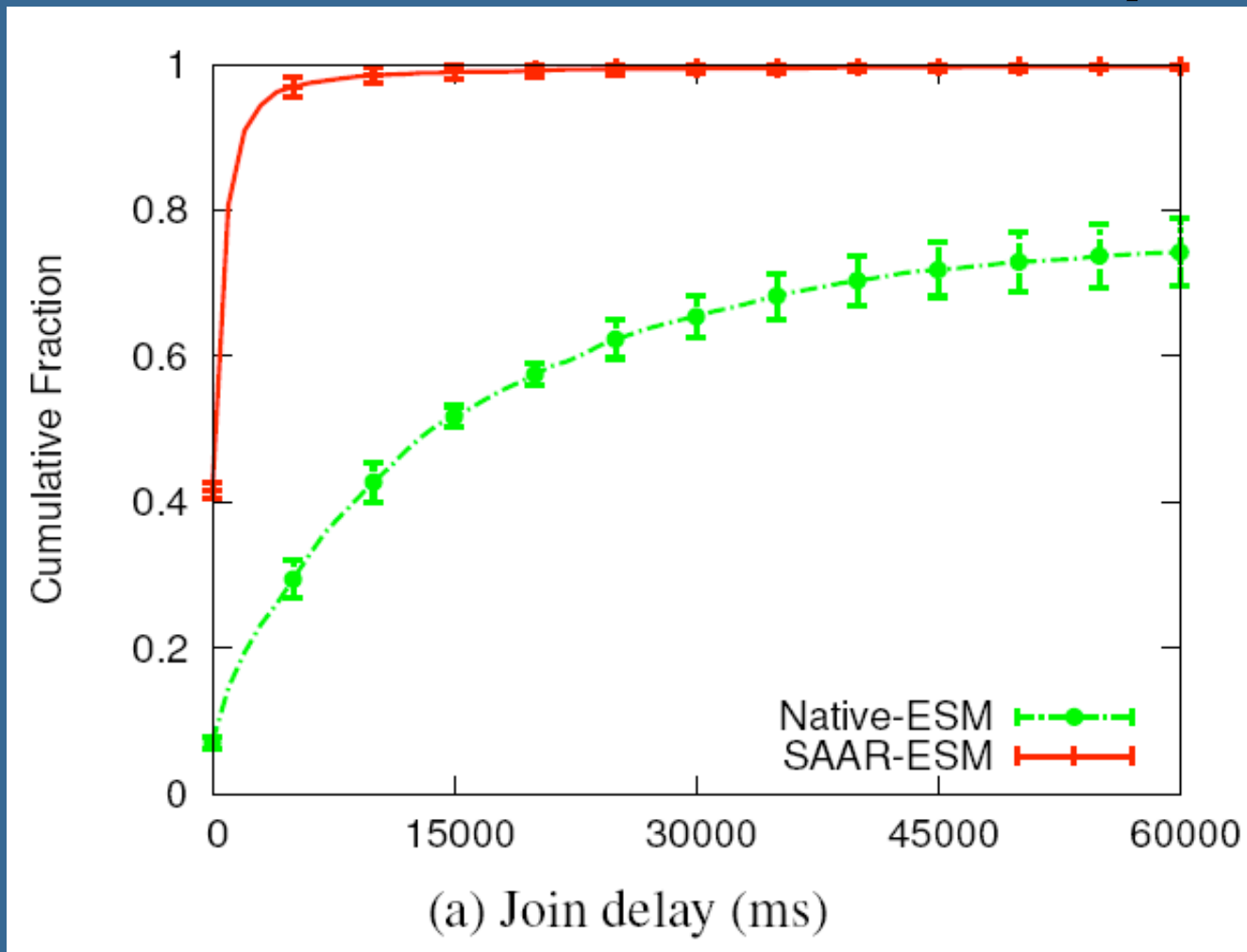


- Node anycasts for indegree-neighbors and employs swarming
- **Group's state variables** -  $g_{capacity}$ ,  $g_{load}$ ,  $g_{loss}$ ,  $g_{buffermap}$ ,  $g_{coordinate}$ ,
- **Predicate:**  $(load < capacity) \ \& \ // \text{ not overloaded}$   
 $(loss < threshold) \ // \text{ low loss}$
- **Objective function:**  
 $cardinality(missingset \cap g_{buffermap}) \ // \text{ recover missing blocks}$   
 $1/g_{loss} \ // \text{ establish a good mesh neighbor}$

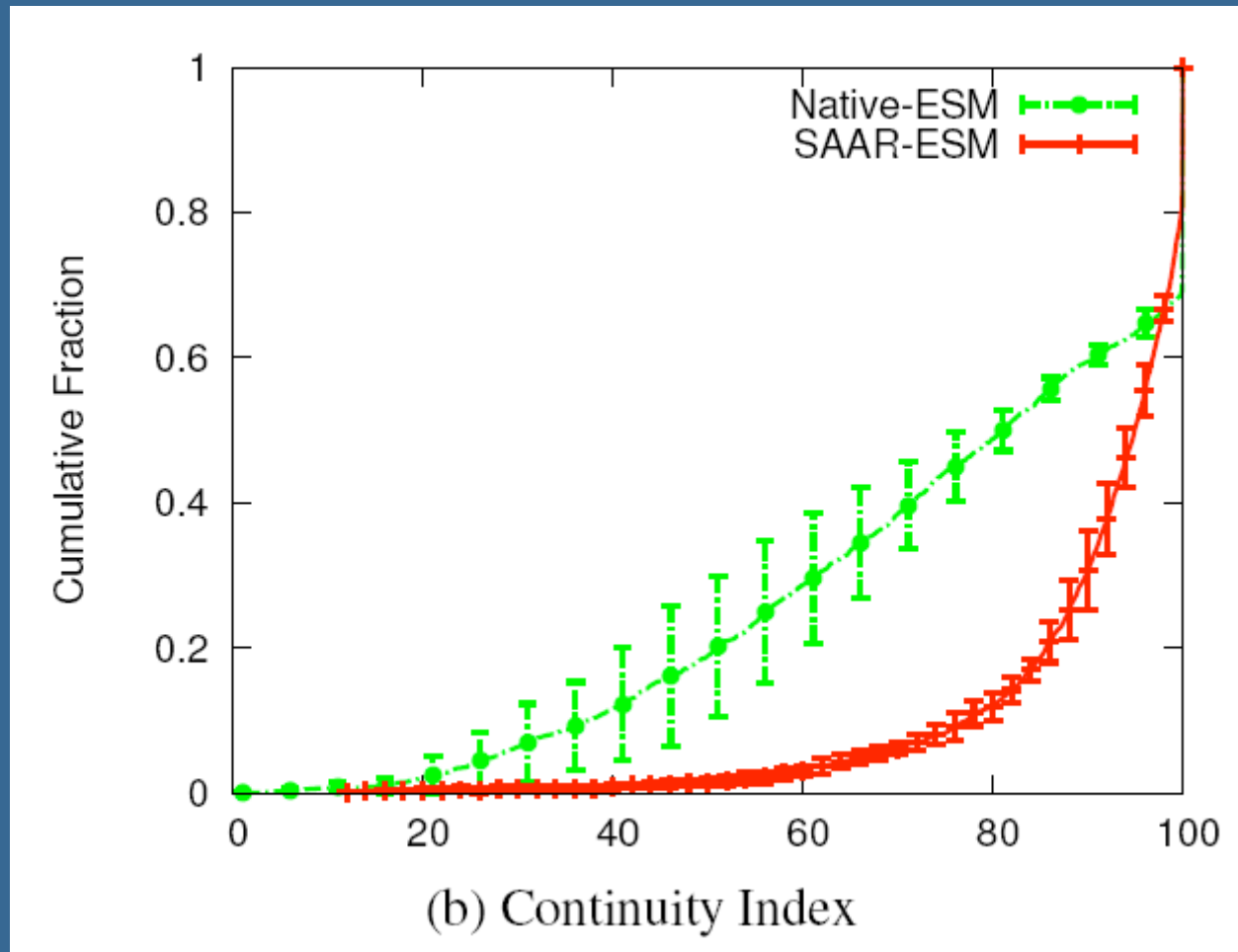
- **Summary:** block-based policy expressed easily

# Graphs from paper

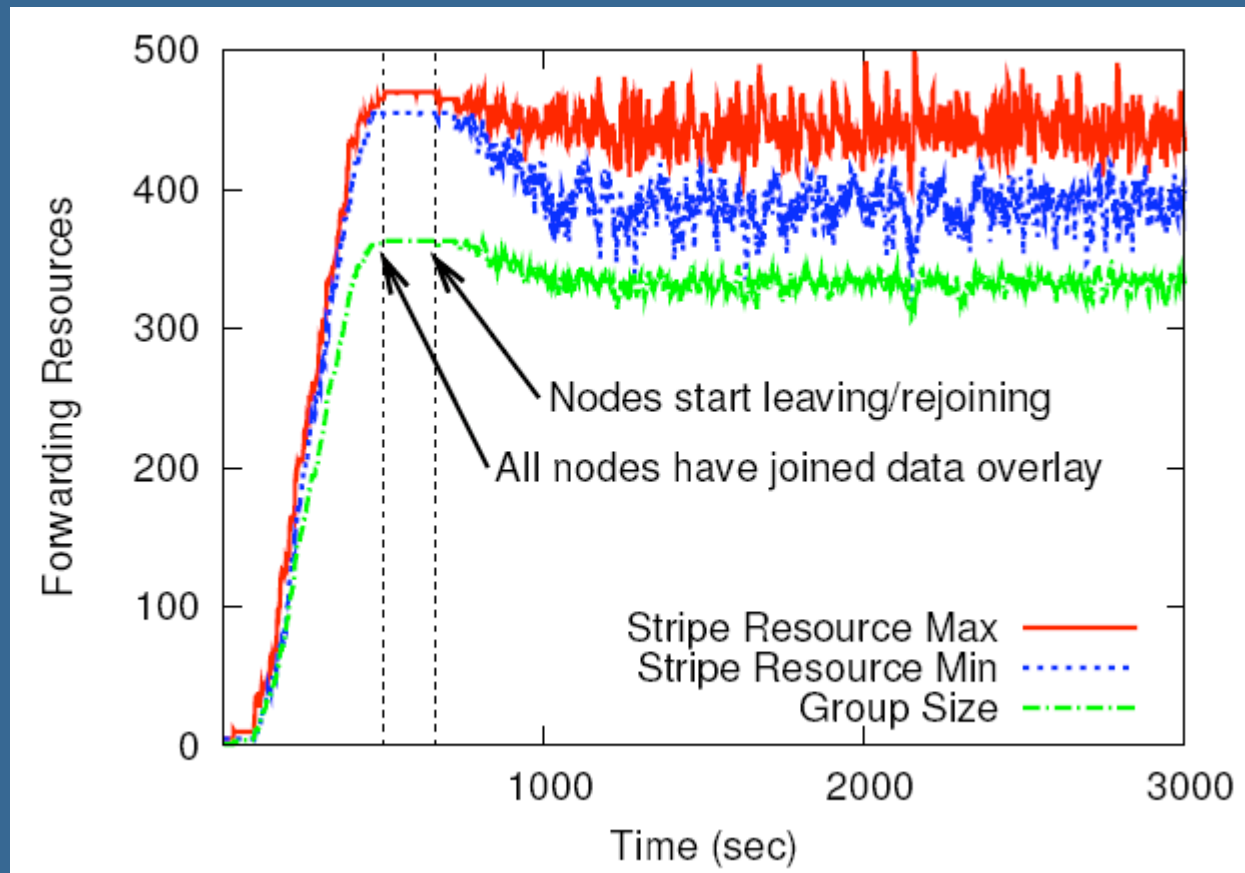
# Planetlab: Join delays



# Planetlab: Streaming Quality

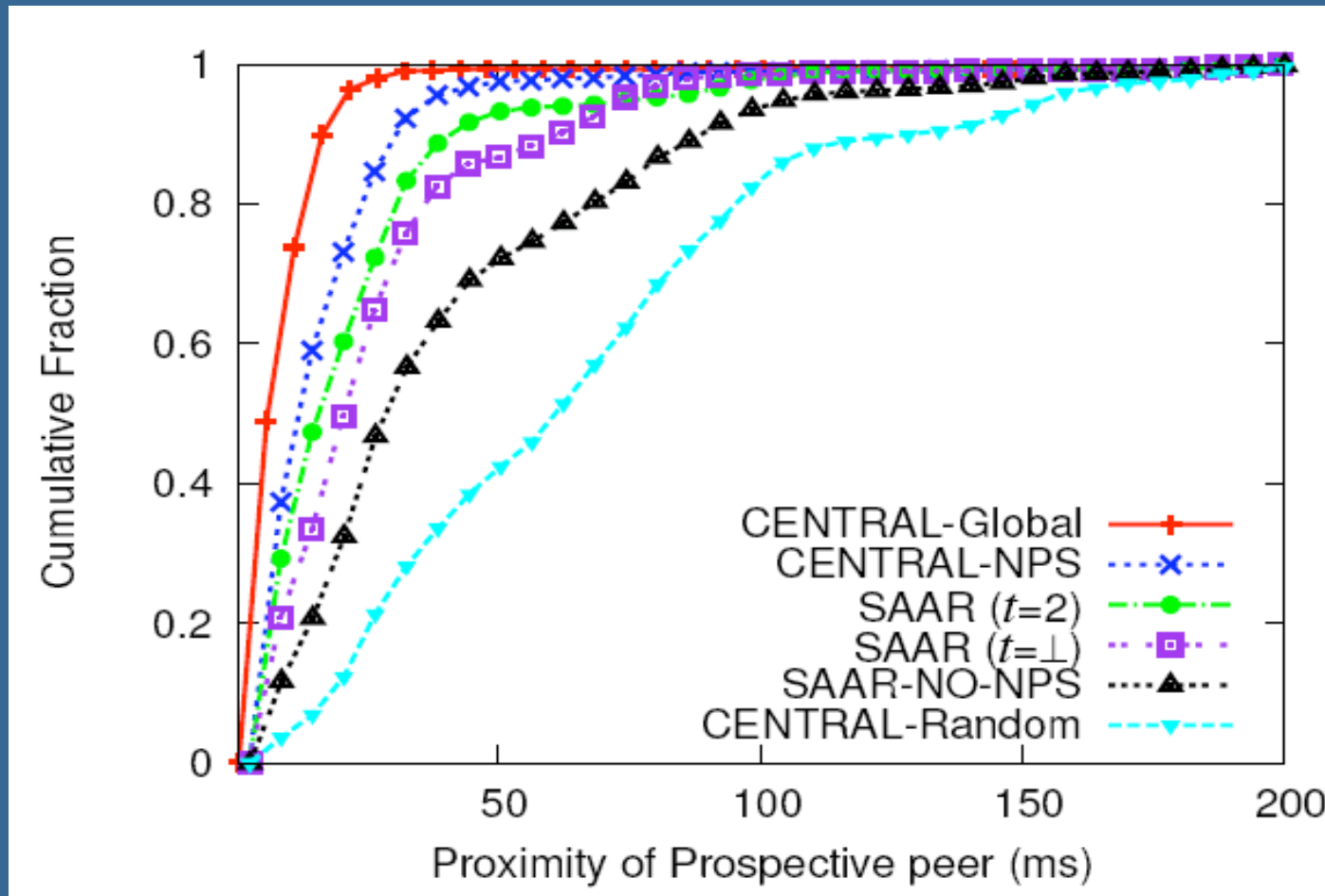


# Multi-tree: Resource Balance

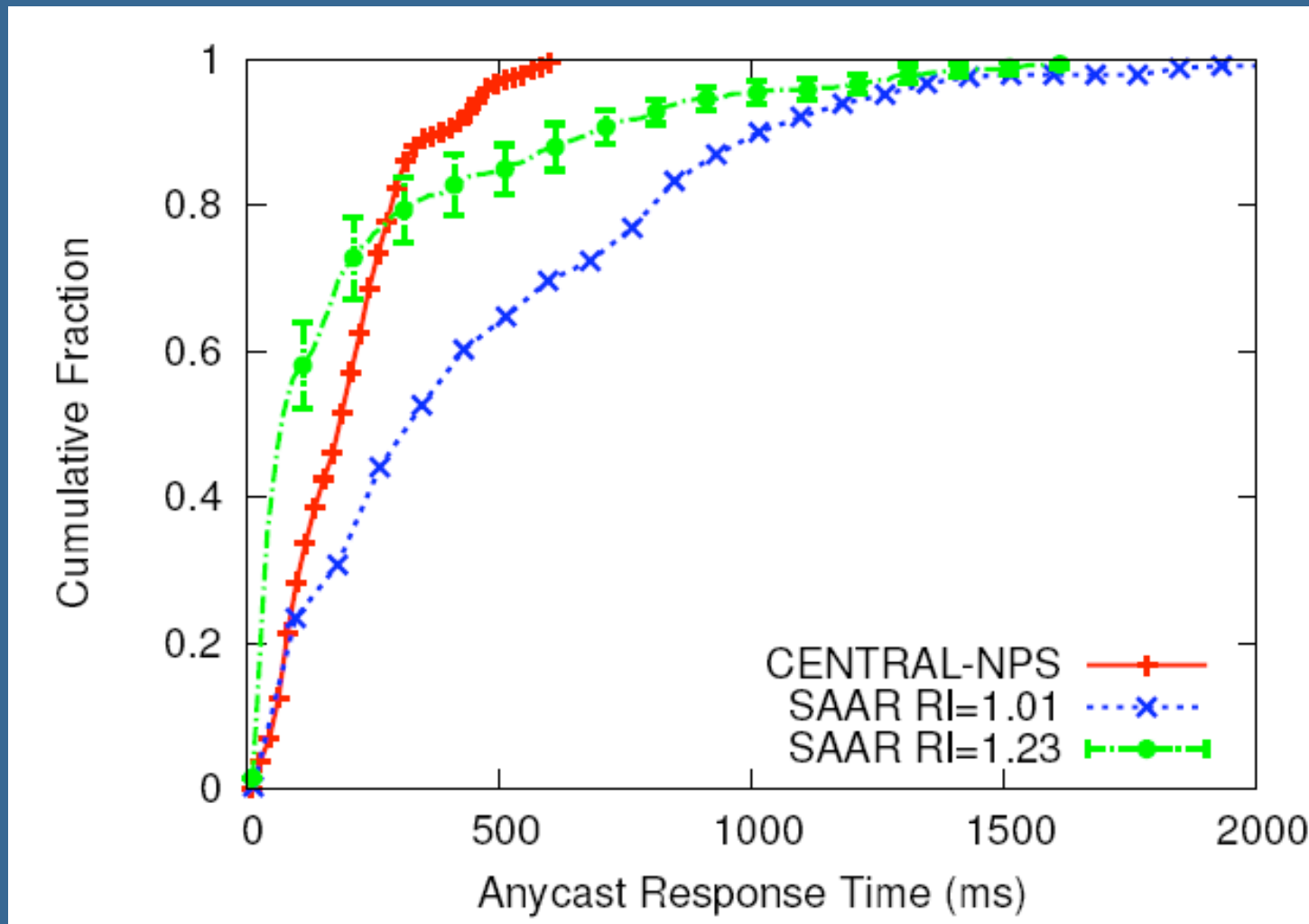




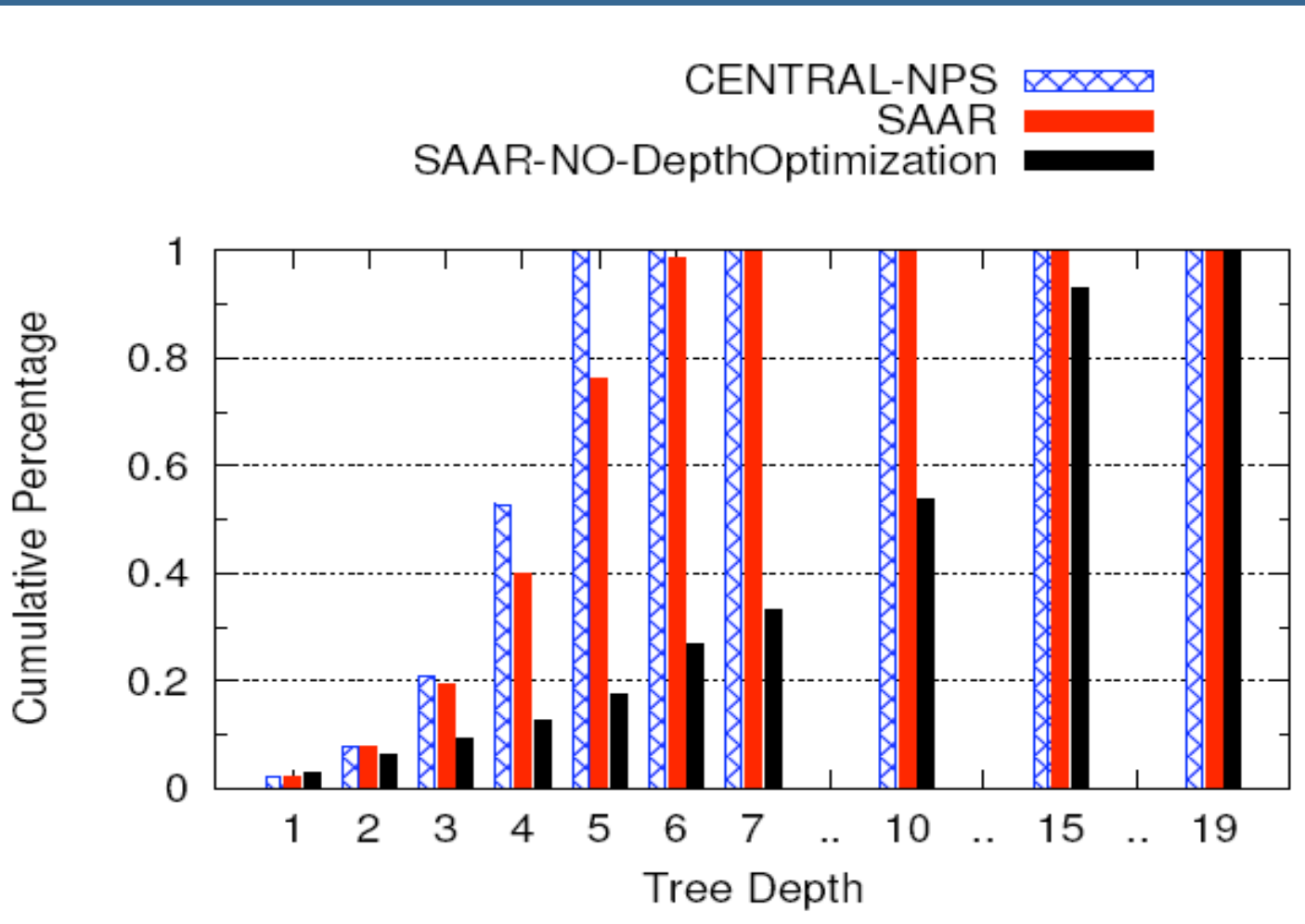
# Locality-awareness



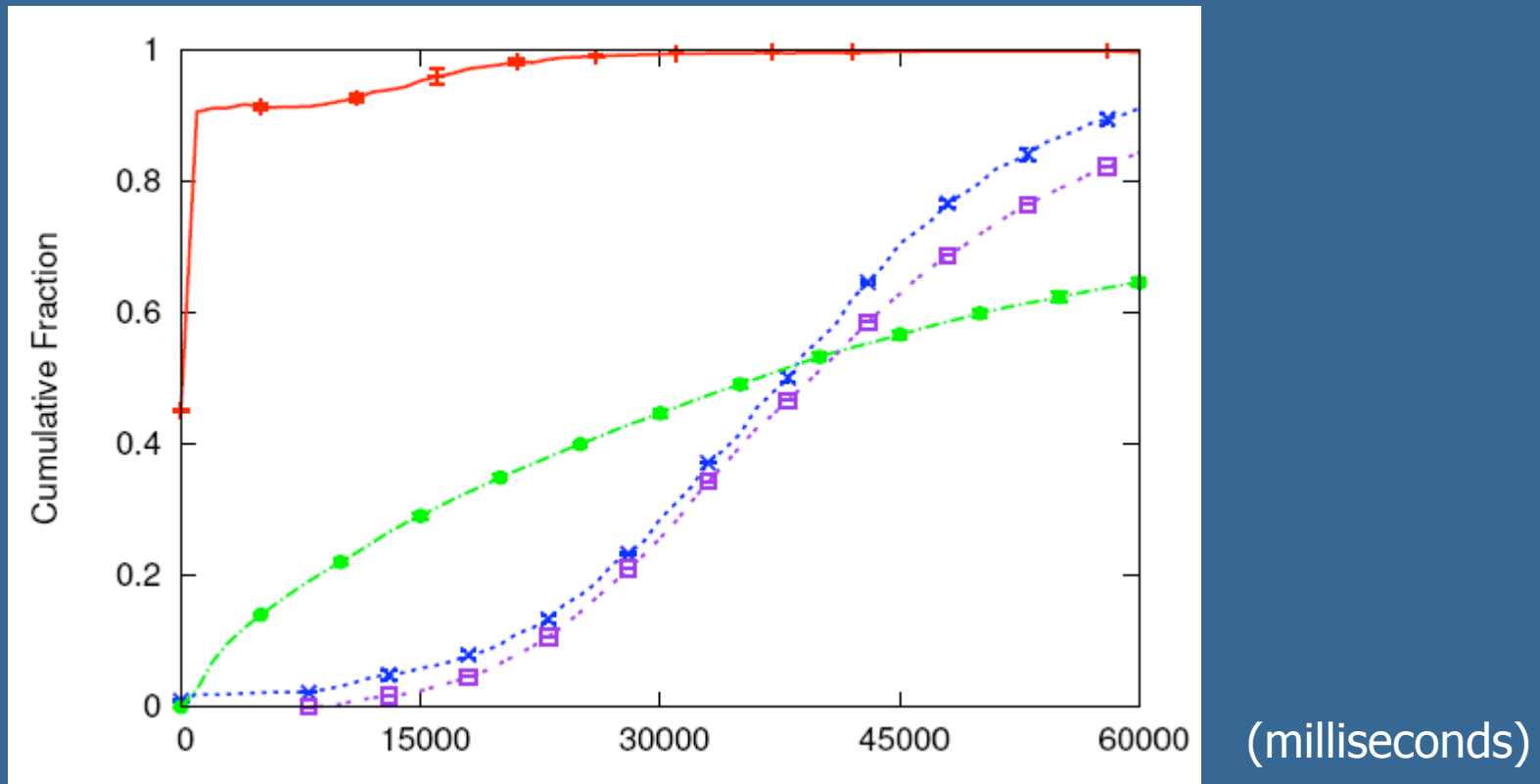
# Load-awareness



# Depth-Optimization

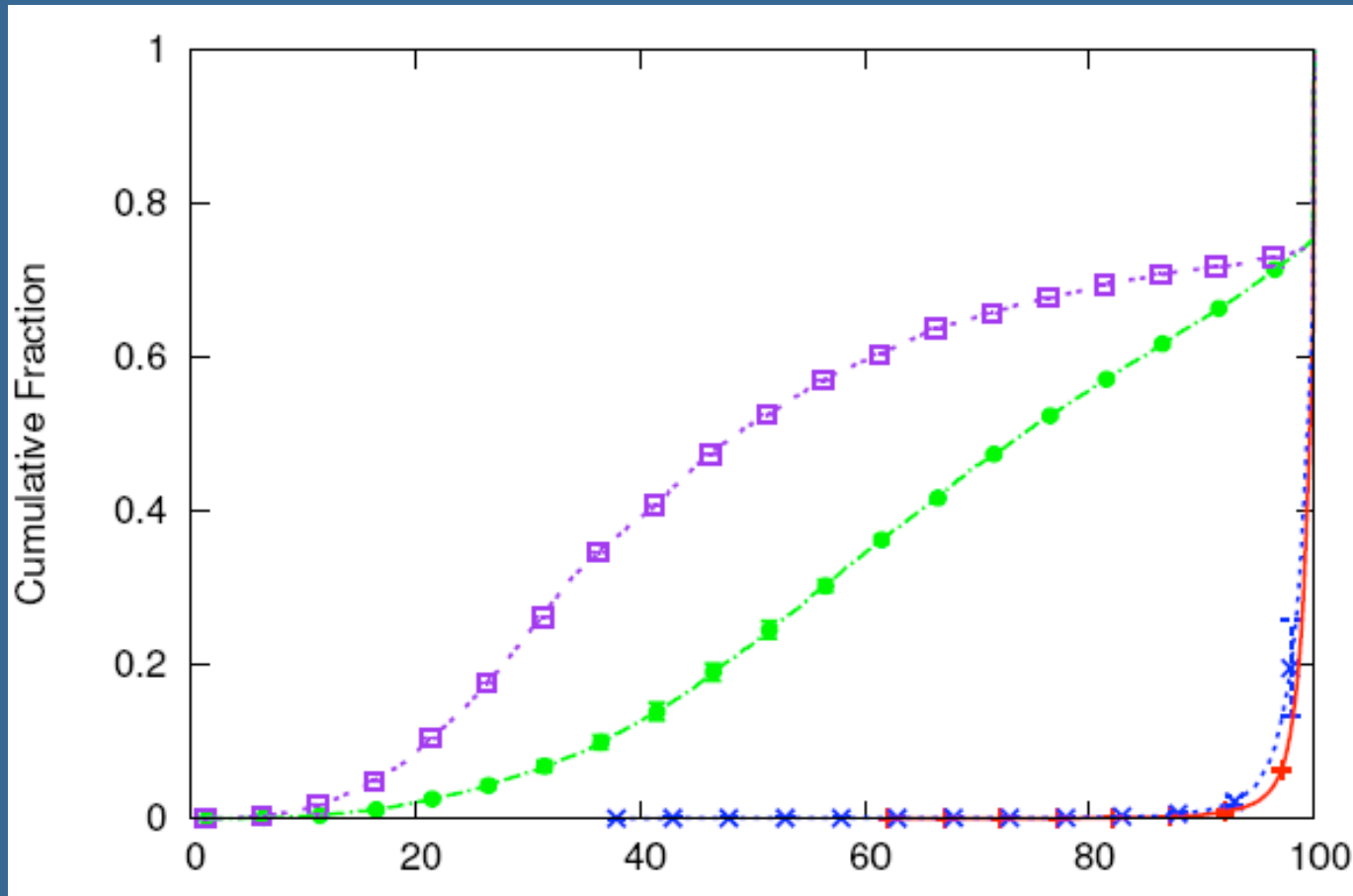


# Single-tree: Join Delays



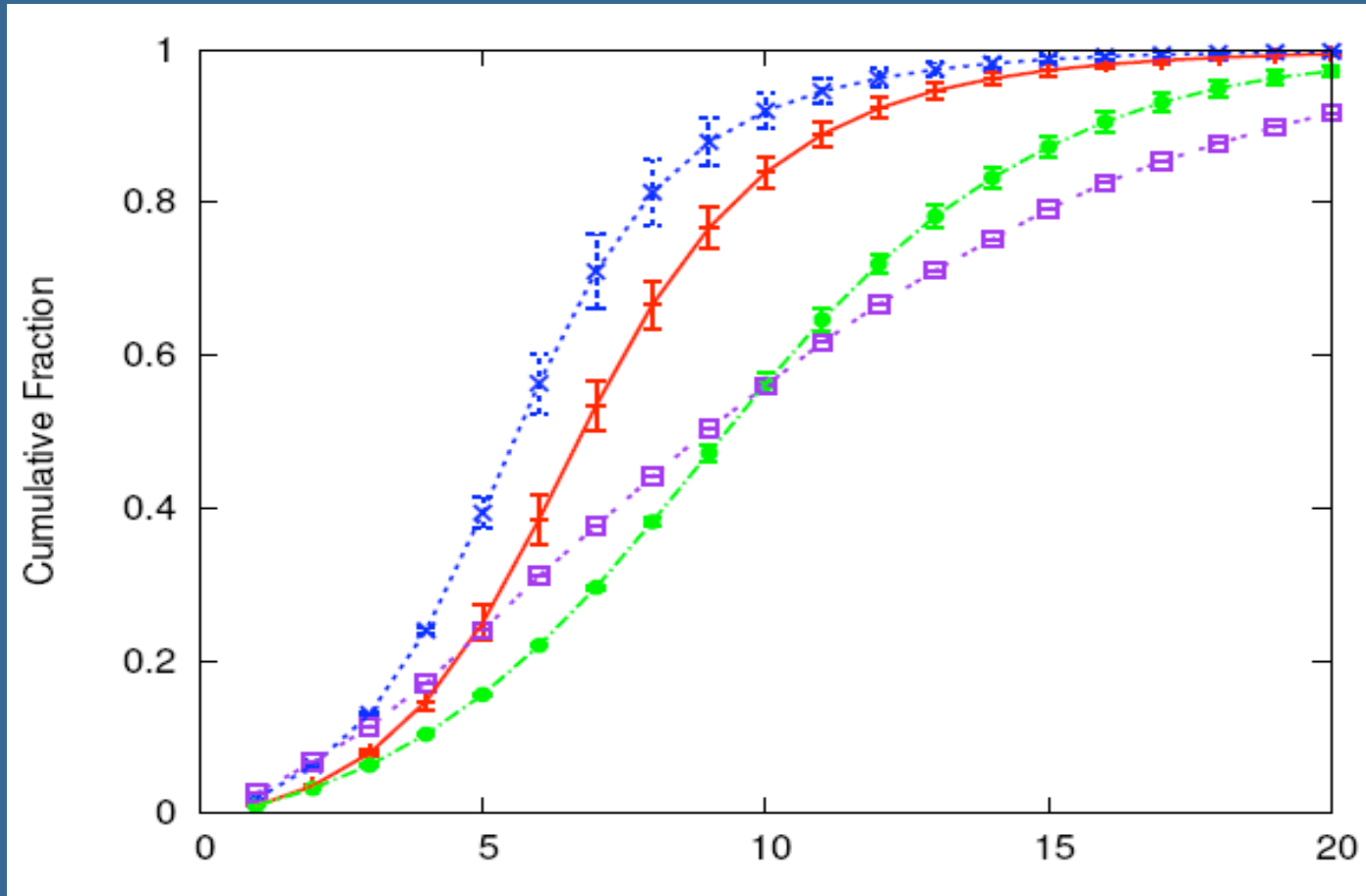
SAAR-ESM    SAAR-ESM-Unshared    Native-ESM    Scribe

# Single-tree: Continuity



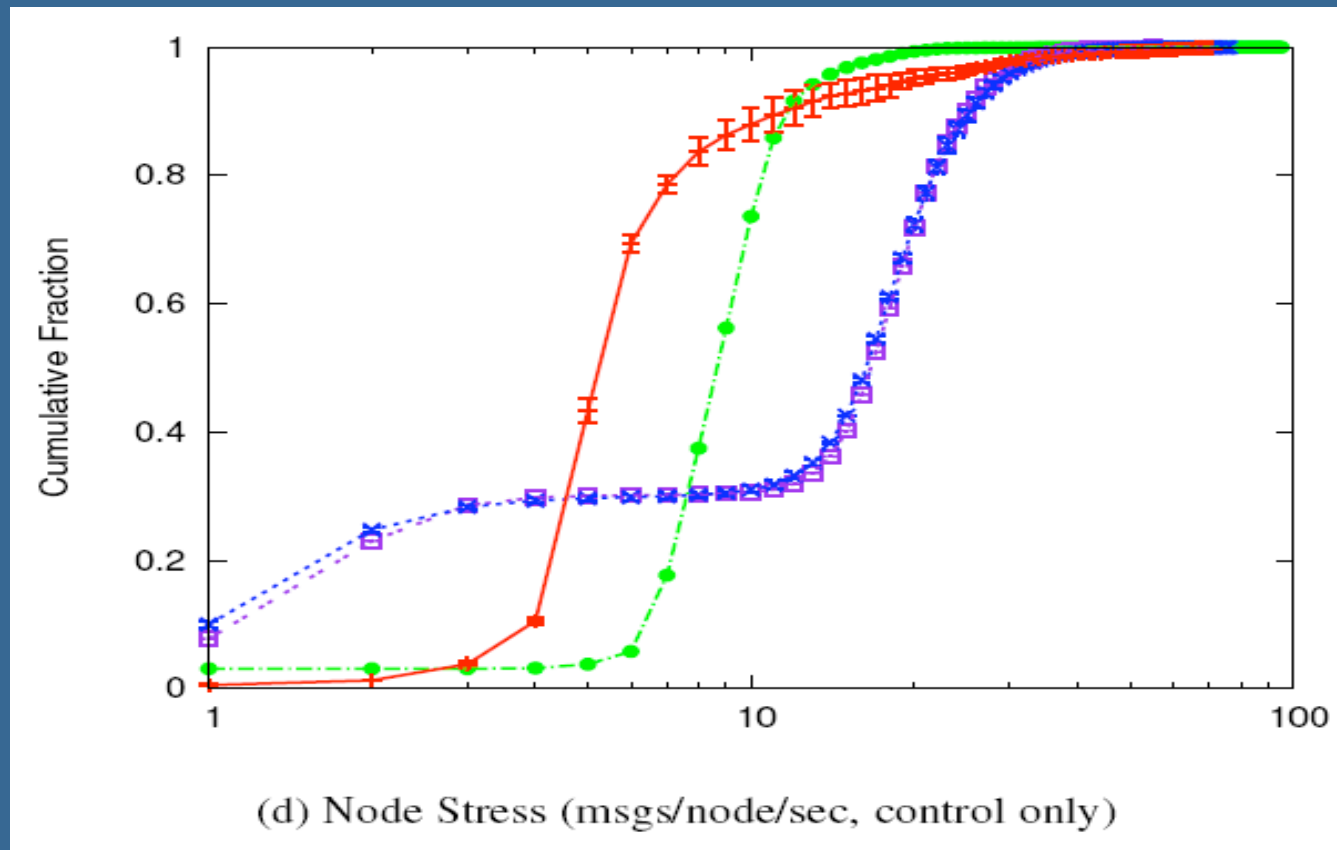
SAAR-ESM (red '+') SAAR-ESM-Unshared (blue 'x') Native-ESM (green 'o') Scribe (purple '□')

# Single-tree: tree-depths



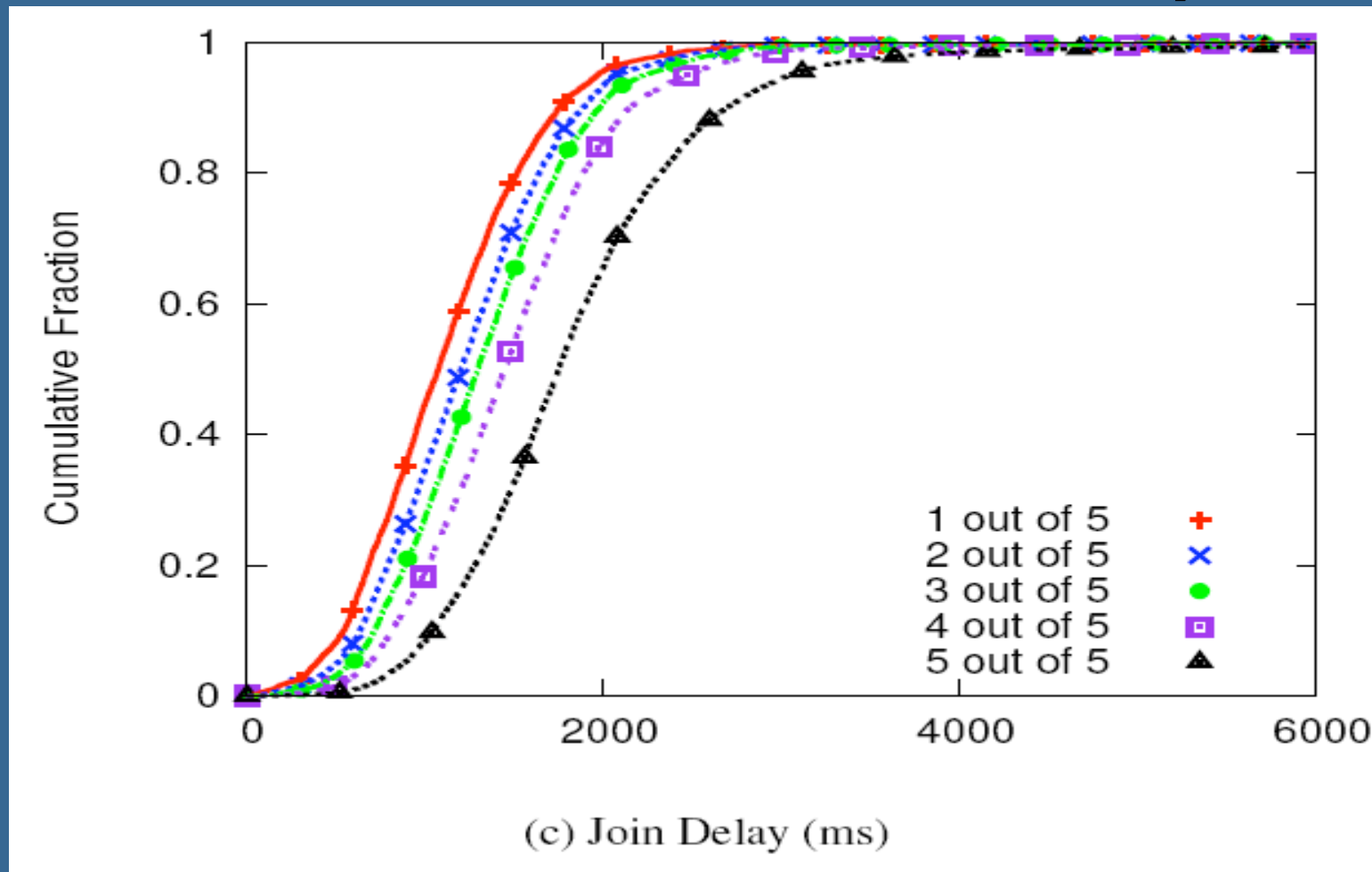
SAAR-ESM SAAR-ESM-Unshared Native-ESM Scribe Scribe

# Single-tree: Nodestress

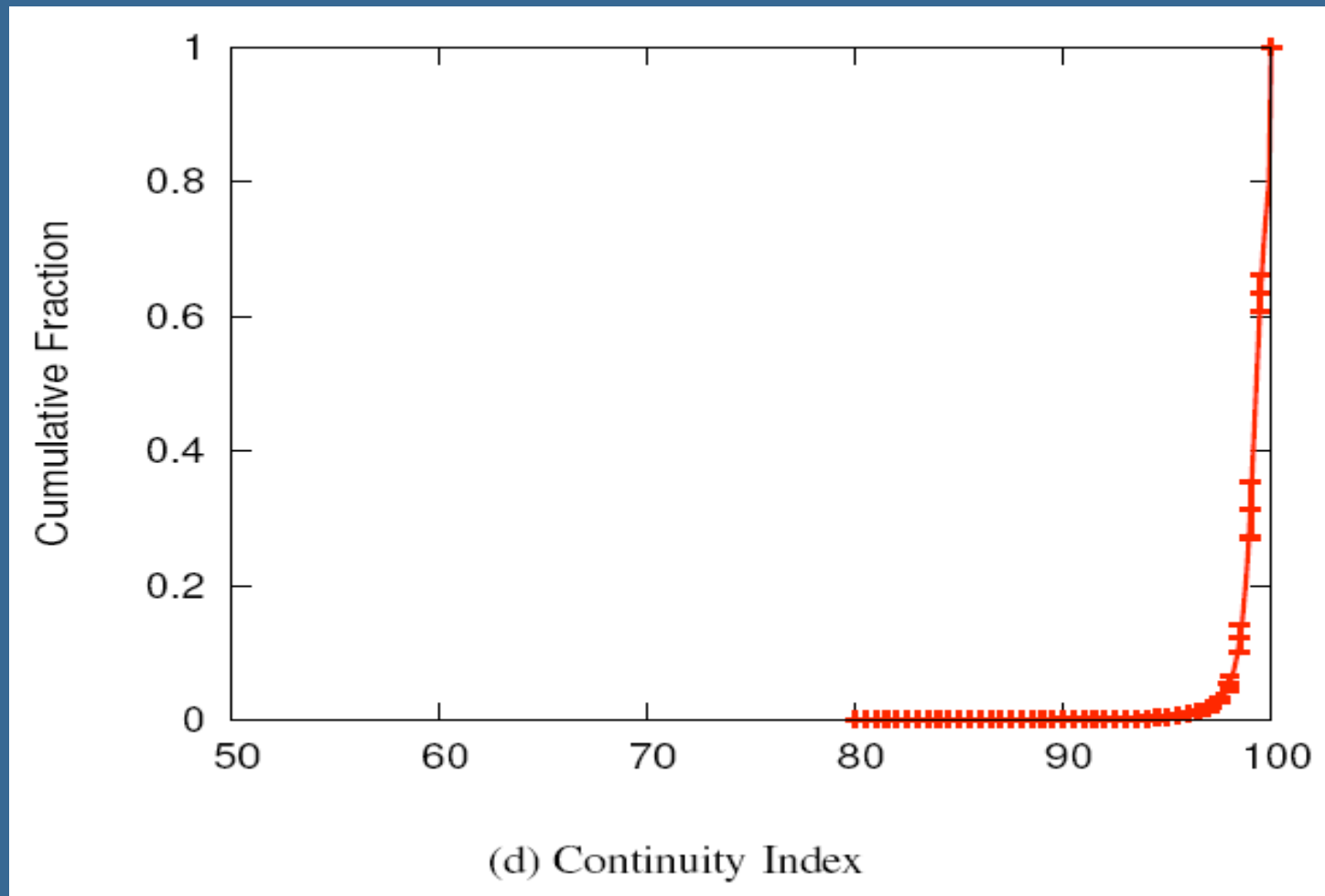


SAAR-ESM  SAAR-ESM-Unshared  Native-ESM  Scribe 

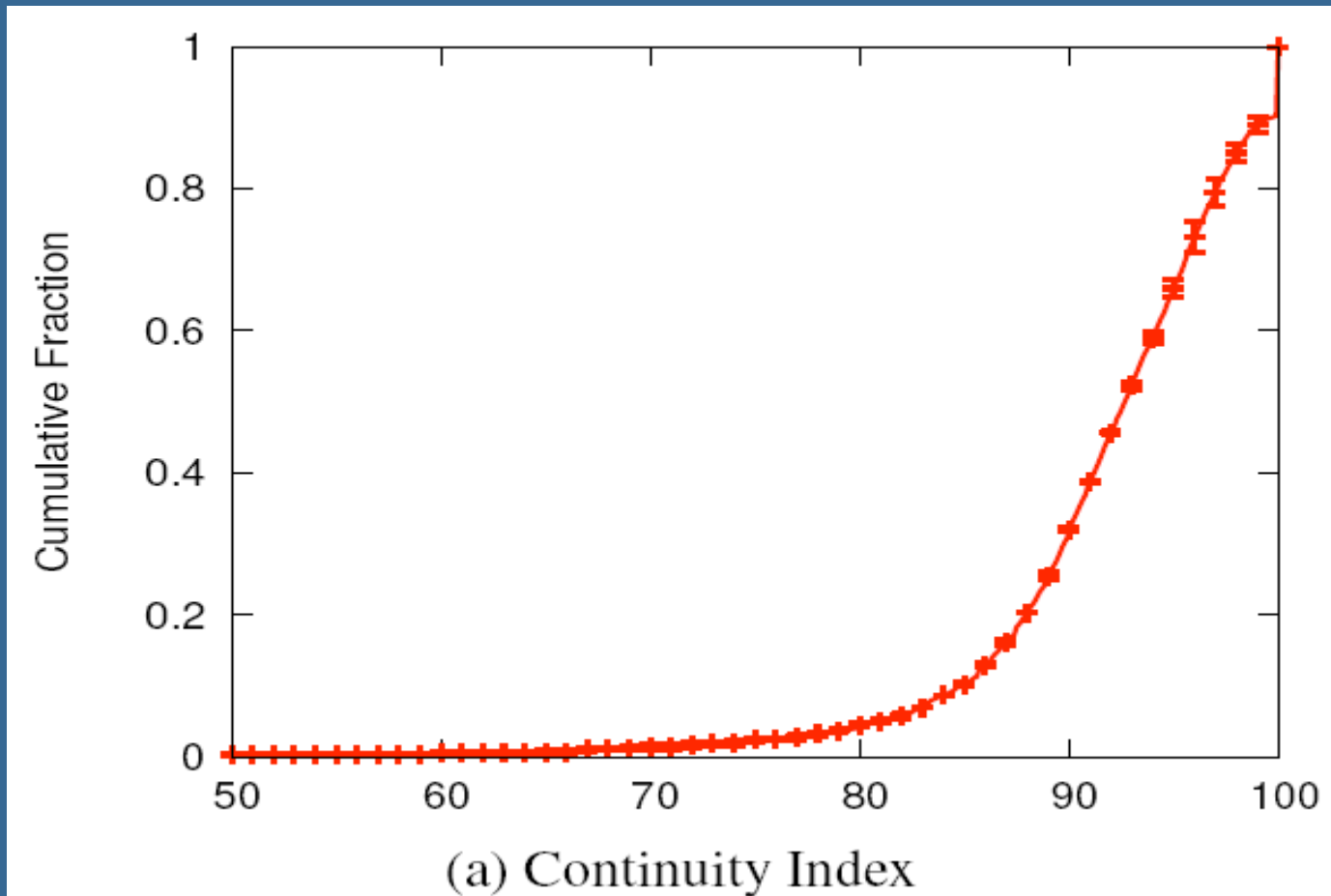
# Multi-tree: Join delays



# Multi-tree: Streaming Quality



# Mesh-based: Streaming Quality



# Mesh-based: Path lengths

