P2P Streaming

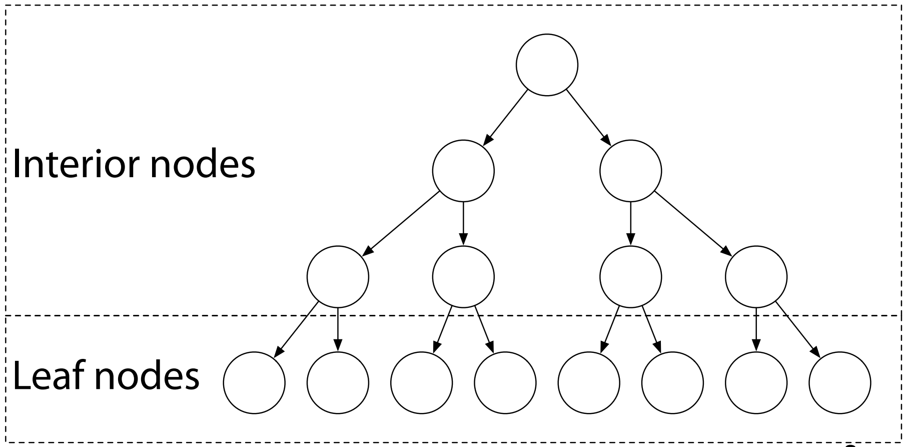
Niels Olof Bouvin

Sharing the cost of streaming

- Multicasting or streaming is data-intensive, timing-critical, and sensitive to data loss
- Centralised or Cloud solutions are expensive, bandwidth- and storage-wise
  - though a fascinating study—Netflix' blog is highly recommended
- Could the burden be shared among the users without affecting the quality?

Characteristics & Terms

- **Time**
  - live vs. stored video
- **Topology**
  - single tree
  - multiple trees
  - mesh
  - combinations hereof



- **Interior node**
  - a node that also forwards data
- **Leaf node**
  - a node than only receives
- **Push**
  - data forwarded as it comes in
- **Pull**
  - data forwarded on demand

Overview

- Tree-based
- Multi-tree-based
- Mesh-based

Single tree multicast

- **Best known example: SCRIBE**
  - based on Pastry, and covered a few weeks past
  - the foundation of SplitStream
- **Fairly rare for streaming purposes, as**
  - leaf nodes contribute nothing, yet consume a stream
  - interior nodes leaving requires costly rebalancing of tree, which results in missed or delayed data delivery
- **(Re)join is either done at the root, or through a walk towards the root (e.g., with a list of (grand)parents)**
  - periodically, the tree will need to be rebalanced for better performance
  - ideally, you would have stronger, more stable peers high in the tree

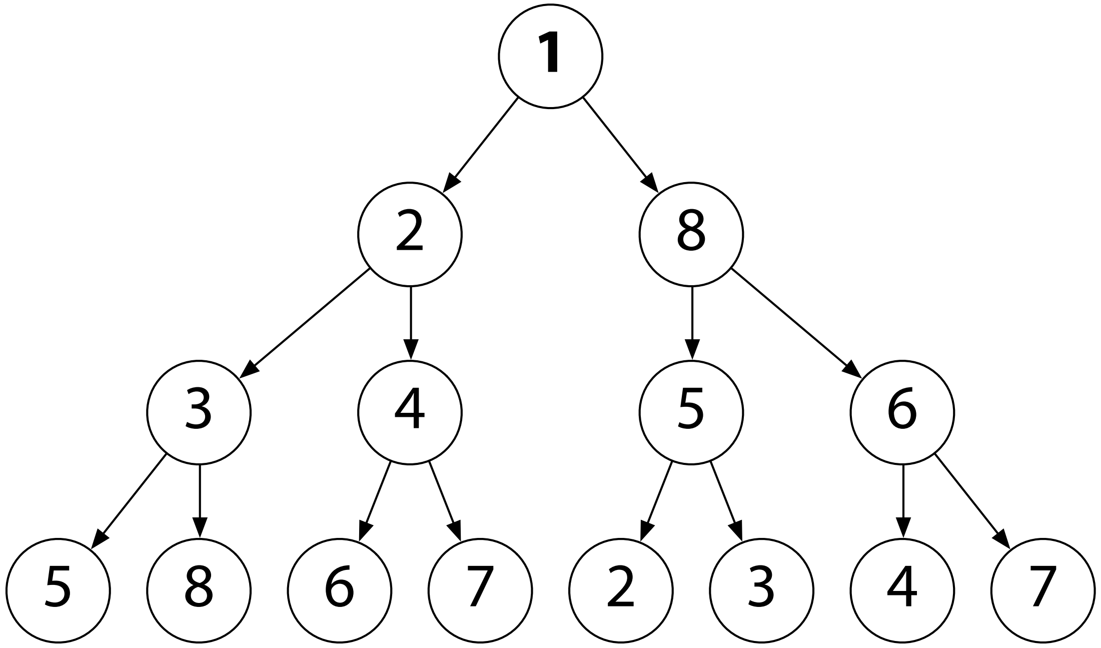
Overview

- Tree-based
- **Multi-tree-based**
- **Mesh-based**

Multiple trees

- **Concept:**
  - divide the data stream into *n* sub streams, usually encoded with redundancy in mind
  - create *n* disjoint trees (one per sub stream) with all nodes
  - every node is an internal node in one tree, and a leaf node in the others
- **All nodes contribute (serve as internal nodes)**
- **All nodes receive the *n* sub streams**
  - or a sufficient number of sub streams to reconstruct the streamed media
- **The disruption of a node leaving will affect at most one sub stream (the one where it is an internal node)**
  - and if the sub streams are encoded using some error correcting method, the missing data can still be recovered from the received sub streams

Multiple trees



- **A multi-tree topology usually seeks to have every node in every subtree, and, ideally, have every node be internal in one tree, and leaf node in all others**
    - this is known as 'interior-node-disjunct'

(Error correcting codes)

- **Encode the signal so that data loss can be recovered**
- **Trivial example using XOR ($\veebar$)**
  - given (a piece of) of a data stream 31415926, divide it into sub blocks: 3141 5926 and create a $\veebar$ block: $3141 \veebar 5926 = 7011$. Transmit the 3 blocks along 3 sub streams
  - if either of the data blocks is lost, it can be reconstructed by the remaining data block and the $\veebar$ block: $3141 \veebar 7011 = 5926$; $5926 \veebar 7011 = 3141$
  - if the $\veebar$ block is lost, there is no data loss, and it can trivially be recreated and transmitted to child nodes
- **Far fancier, more robust, and more efficient methods exist, such as Reed-Solomon and Fountain codes — see https://en.wikipedia.org/wiki/Erasure_code**

Challenges

- **Construct the *n* trees satisfying the requirement**
- **Timing becomes critical: a node must receive all *n* packets within a narrow time span for good playback**
  - nodes close in IP space should be close in the tree to minimise latency
- **All the complexities of maintaining a single multicast tree $\times$ *n* + some extra overhead**

SplitStream

- **Yet another system based on Pastry (and SCRIBE)**
- **The stream is split into *k* stripes, encoded such that only a subset is sufficient for playback**
- **A joining peer specifies its in- and outbound capacity**
  - that is how many stripes it wishes to receive and how many it can transmit

Quick Pastry recap

- **Pastry is a structured P2P network**
  - supports effective, distributed object location and routing in $O(\log_{2^b} N)$
    - 128 bits ID space, encoded as digits in base 2^b
    - routing tables with $O(\log_{2^b} N)$ rows and 2^b columns
    - the *n*'th row in the routing table share the first *n* digits with the local node's ID
- **Locality awareness**
  - Pastry maintains a "neighbourhood set" of the $|M|$ nodes that are closest by some proximity measure (e.g., routing hops)

Creating interior-node-disjoint trees

- **Pastry routes messages by finding longer and longer common shared prefixes, at least one digit per hop**
- **SCRIBE trees are created by a joining peer forwarding a 'join' message towards the groupID. Join is achieved, once a peer, either a member or a forwarder, is found**
    - any peer along the way is added to the multicast tree as forwarders
    - thus, all interior nodes in a SCRIBE tree will share at least one common digit in their ID
- **Therefore, if we create stripeIDs with different first digits, we ensure that nodes are interior in only one tree, namely the one where they have the first digit in common with the stripeID, and leaf nodes in all others**

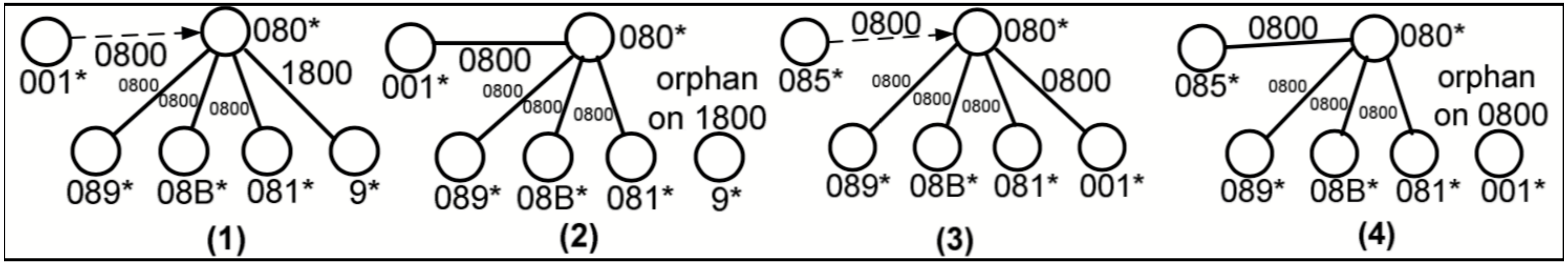
Memberships

- **All peers are members of the *k* stripes**
  - where they will be leafs in most
- **If *k* = 2^b, every node will have the possibility to be an interior node in one of the *k* stripe trees**
  - every node is expected to be able to at least forward its own stripe
- **Any joining node subscribes to the number of stripes it has incoming capacity for**
  - and always the stripe with which it shares the first digit common prefix
- **If it has unused outbound capacity, it will also join the *spare capacity group***

Joining a peer at capacity

- **If a peer attempted to join an already full peer in SCRIBE, the peer would be rejected and sent to one of the children instead, where the process would be repeated until the new peer found a capable peer**
  - this will work because every SCRIBE peer is required to accept at least one child
- **In SplitStream, things are a bit more complicated, because a leaf peer might already be at capacity from being an interior node in another stripe tree**
  - so we cannot just "push-down" the joining peer

Joining a full peer in SplitStream

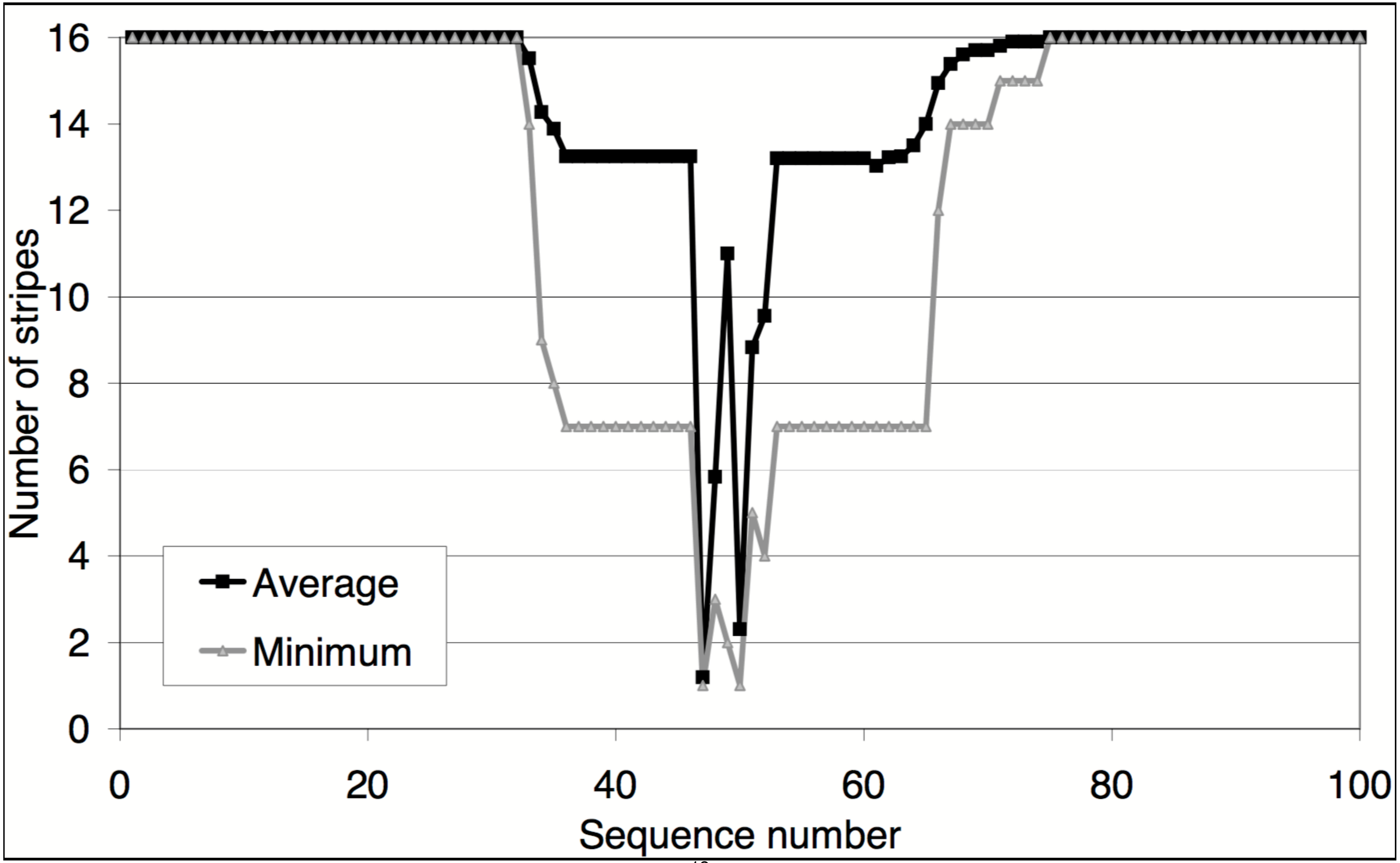


- **Which peer should be orphaned?**
  - If a peer joins, and there are children whose stripeID does not match the node's own, orphan one of those (2)
  - if there are no peers not matching the stripeID, orphan the peer with the shortest prefix match (4)
  - the orphan first seeks to attach itself to a former sibling with a stripe matching prefix
- **Where do rejected orphans go?**
  - the orphan sends a message to the (IP) nearest peer in the spare capacity group
  - the peer searches for a child that can supply the desired stripe—if there is no such child, the search is sent to a parent until a match is found and joined by the orphan

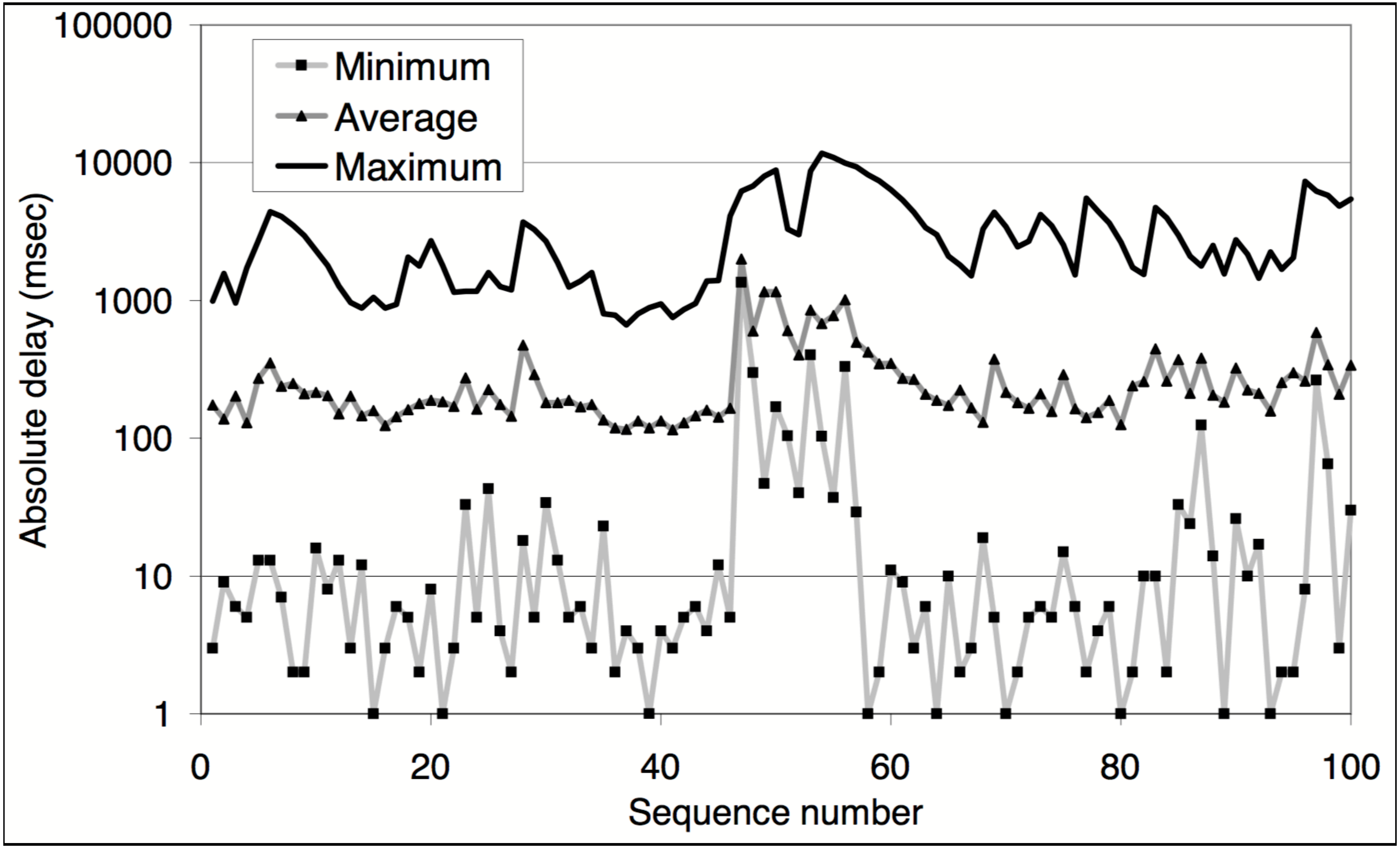
Evaluation

- **Very extensive simulation tests (40.000 nodes)**
- **…as well as live PlanetLab tests (72 nodes)**
  - PlanetLab is a global testbed with ~1300 computers across ~700 sites
  - participating institutions get access to running their experiments across the network

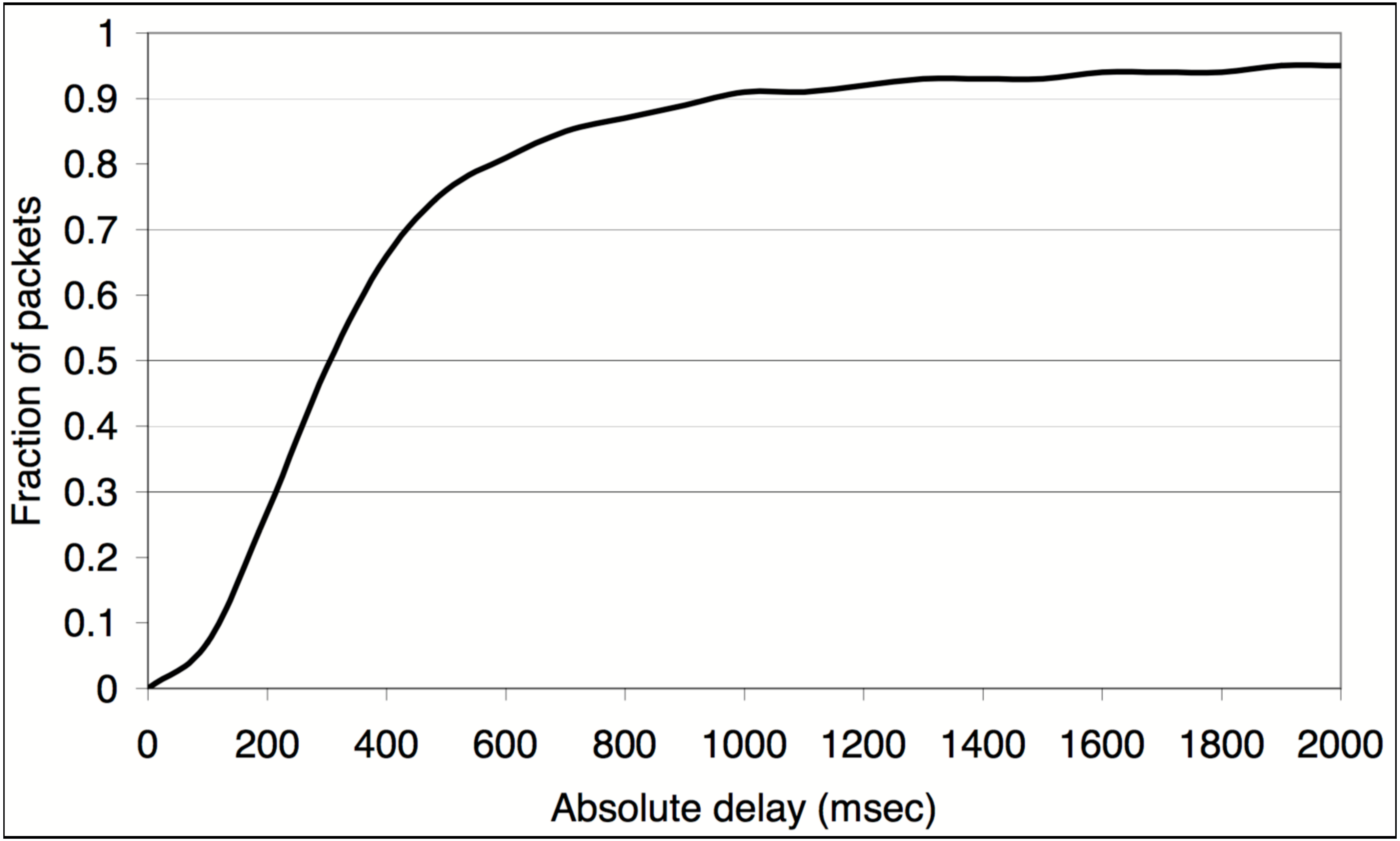
Effect of 8 nodes leaving



Effect of 8 nodes leaving



CDF of packet delay



Summary

- A sophisticated system cleverly built on top of Pastry
- The reorganisation of peers has complications, but the arrangement of stripes also ensures that peers will contribute

Overview

- Tree-based
- Multi-tree-based
- **Mesh-based**

Mesh-based streaming

- No fragile parent-child relationships
- Peers can send and receive data to/from many other peers
- Traffic is usually *pull* rather than *push*, as one peer cannot know what another peer has or has not got
- Peers share bitfields with their neighbours, and request data blocks that they have not yet received
- More robust than trees

DONet/CoolStreaming

- Mesh-based system
- DONet is the architecture, CoolStreaming an application that saw actual deployment and use
- All DONet peers have a unique ID, and maintains a mCache, a partial list of other peers, maintained through gossiping, as well as a partner list of *M* active neighbours
- Origin node: the source node
- Deputy node: nodes connected directly to the origin
- BufferMaps: bitfields of segments owned or desired

Joining DONet

- A new peer contacts the origin, who forwards the request to a randomly chosen deputy
- The deputy responds with a list of candidate peers from its mCache
- The peer can then contact the candidate peers and register itself as a partner, updating its own and its partners' partner lists and mCaches in the process

Streaming

- **The streamed content is divided into segments**
  - 1 second long in the paper
- **Each peer will continuously exchange BufferMaps of the segments in its possession with its partners**
- **If some segments are desired, a BufferMap expressing this is created and sent to the peer holding them**

Segment selection

- **Each segment has a playback deadline, and partners have varying bandwidth**
- **Given BMs from the partners, select the missing segments, and calculate the #partners per segment**
  - segments held by one partner: start downloading, if there is still time
  - segments held by several partners: select the partner with highest available bandwidth and sufficient time to send the desired segments
  - no point in trying to download what will arrive too late
- **BMs matching the desired segments are sent, and the partner will start transferring the segments**

Partners leaving

- Graceful leave: send departure message to partners
- If a partner leaves, its neighbours will delete their BufferMap of that partner
- Hard leave: a neighbour will detect the failed peer, and gossip a departure message in its stead

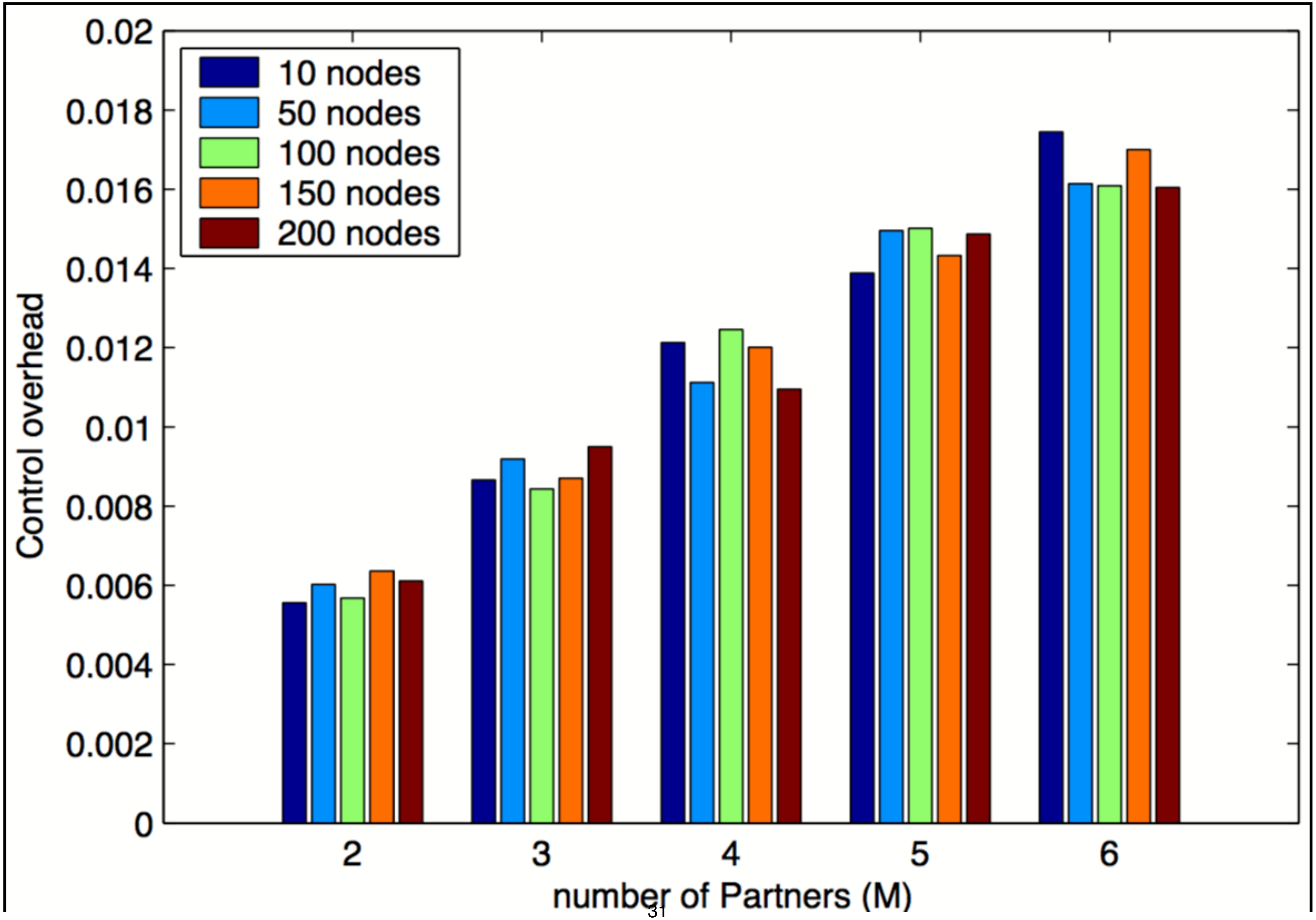
Maintaining partner lists and mCaches

- Peers periodically gossip about known peers
- New peers are added to the mCache, and may experimentally be added to the partner list
- Partners are periodically evaluated on the number of segments shared, and the lowest scoring partner is rejected

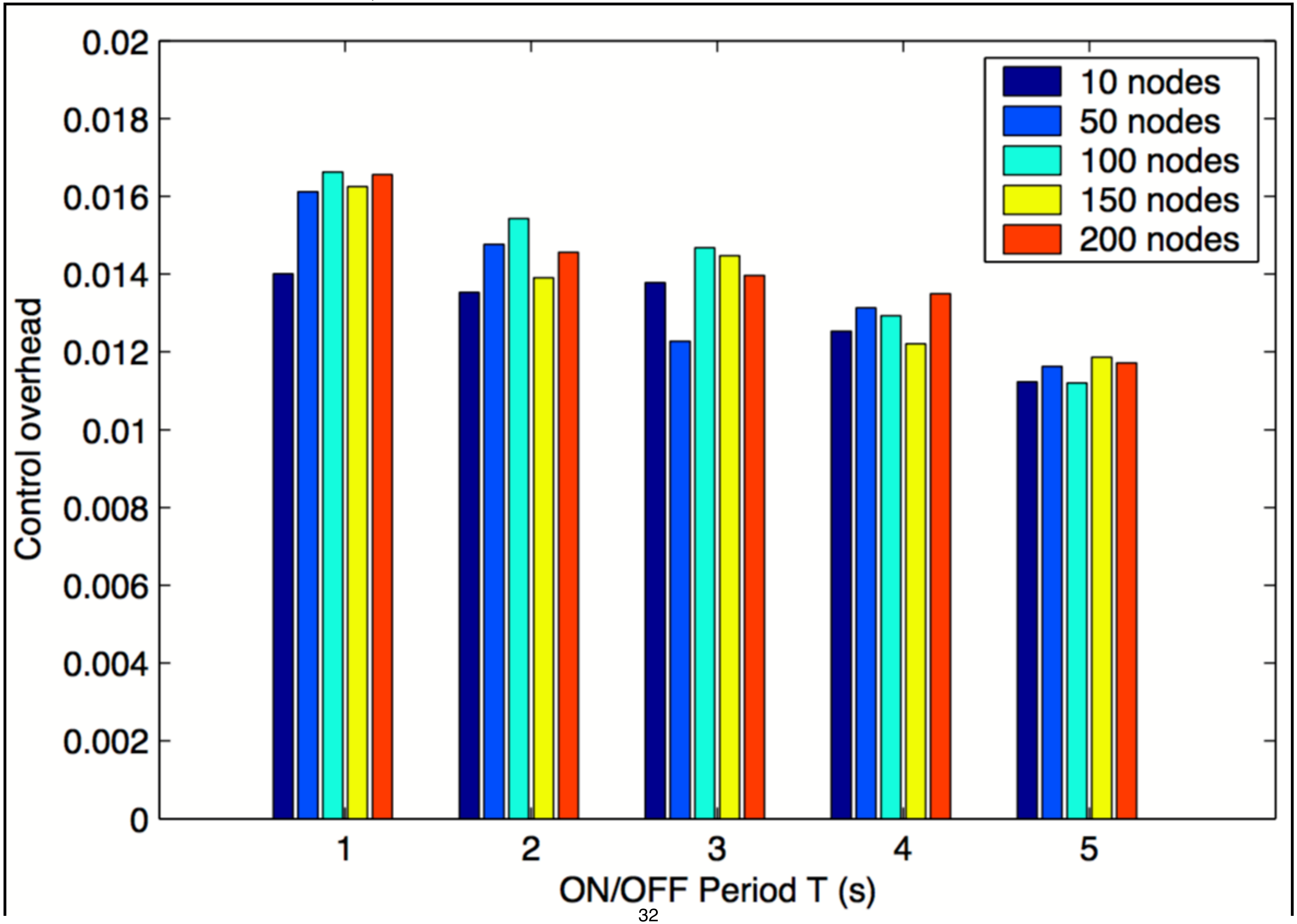
Evaluation

- PlanetNet: 200 peers
- CoolStreaming: deployed with up to 4000 users

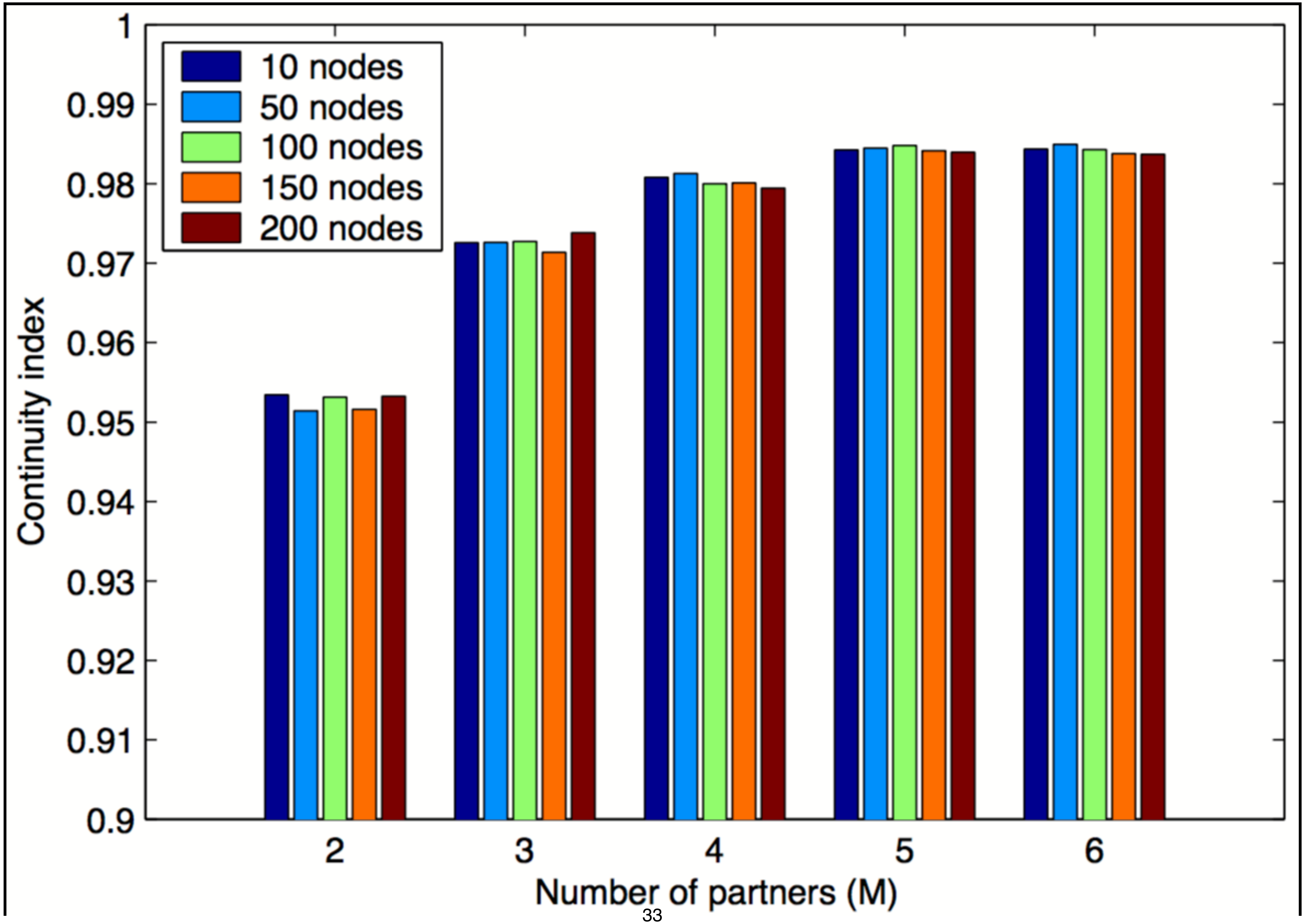
Overhead: static network



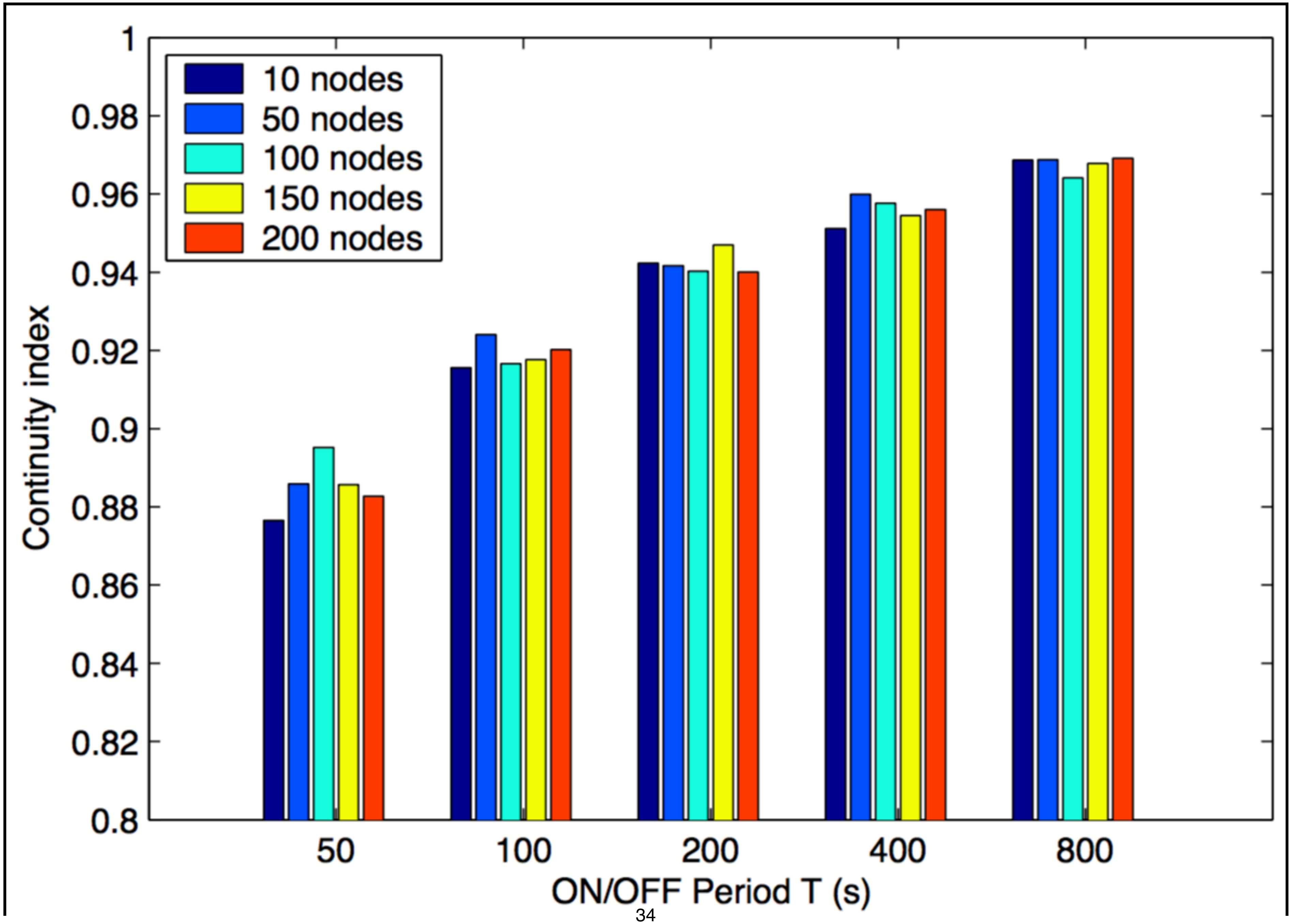
Overhead: dynamic network (4 partners)



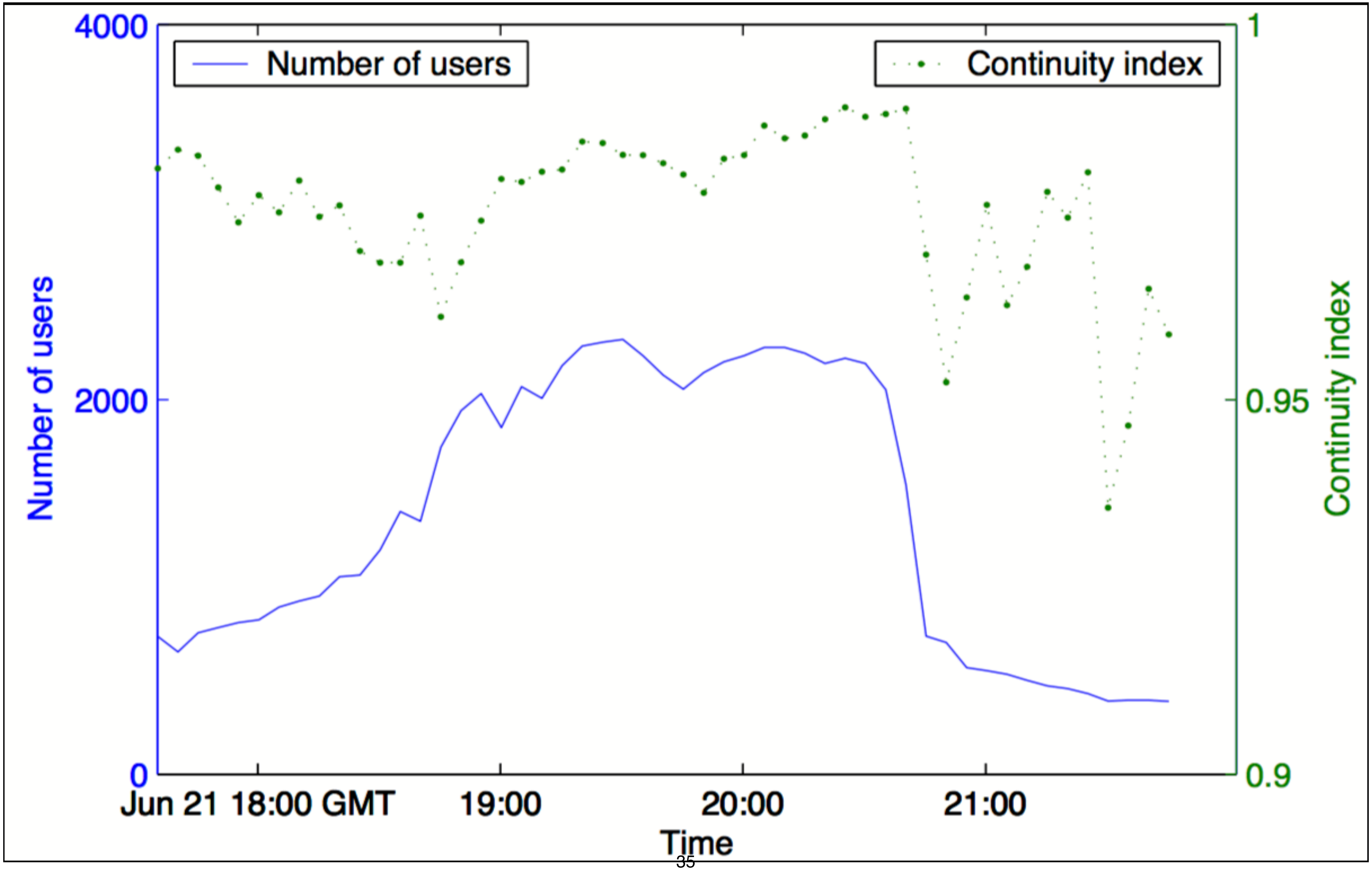
Continuity: static network



Continuity: dynamic network (4 partners)



CoolStreaming: actual users



Summary

- **Far simpler than SplitStream**
  - this makes for more robust code—tree balancing on a live system is challenging
- **Remiscent of BitTorrent—Popcorn Time is BT-based**

P2P streaming

- Is absolutely feasible
- Robustness requires encoding to handle data loss, as it is inevitable in a distributed system
- Streaming pre-recorded content is easier, as deep buffering can handle most irregularities
  - with a BT approach, the trick is then creating a specialised piece selection strategy, as well as having the tracker select peers roughly at the same playback time