Trees versus meshes: Is the Debate Over?

Paul Francis
P2P Streaming Workshop, Sep. ‘06
“Mesh” approaches to P2P streaming are popular

- Coolstreaming
- Lots of startups use meshes
  - (as far as I know)

- Simple
- Robust
- Acceptable overhead (high volume apps)
But I’ve been working on “tree” based approaches

- So, motivated to show that tree-based approaches are better than mesh-based
  - Don’t want to have wasted my time!
- Therefore came up with this title of talk when Pablo asked me to speak:
  - Trees versus Meshes: Is the Debate Over?
Some caveats

- Only talking about live streaming
- Not sure I’m really ready to give this talk
  - Haven’t done a good study of trees versus mesh pros and cons
    - Though I plan to
- Therefore may be holes in my logic
  - This is a workshop!
  - Food for thought…
What I have done (with Vidhya Venkatraman)

- Design of an unstructured tree-based P2P multicast protocol
- Chunkyspread
  - ICNP ’06
  - Multi-tree
  - Scalable
  - Supports heterogeneity
    - Good control over transmit load
  - Performs better than Splitstream
Trees versus meshes

- More similarities than differences
Trees versus meshes

- More similarities than differences
- Both approaches can be unstructured
  - Chunkyspread is, but also Yoid (1998)
Trees versus meshes

- More similarities than differences
- Both approaches can be unstructured
  - Chunkyspread is, but also Yoid (1998)
- Both optimize on volume
  - Most bytes follow the path of a tree
Trees versus meshes

- More similarities than differences
- Both approaches can be unstructured
  - Chunkyspread is, but also Yoid (1998)
- Both optimize on volume
  - Most bytes follow the path of a tree
- Both effectively utilize send capacity of all peers
  - Multi-tree
So what is different?
So what is different?

- Data delimiting?
  - Meshes use blocks, trees use slices
  - But both of these are attempts to aggregate
  - This difference isn’t really important
So what is different?

- Data delimiting?
  - Meshes use blocks, trees use slices
  - But both of these are attempts to aggregate
  - This difference isn’t really important

- Trees are push and meshes are pull?
So what is different?

- **Data delimiting?**
  - Meshes use blocks, trees use slices
  - But both of these are attempts to aggregate
  - This difference isn’t really important

- **Trees are push and meshes are pull?**
  - But when a child selects a parent in the tree, it effectively requests (pulls) a slice
The basic difference:

- **Meshes:**
  - *Peers advertise what they already have*

- **Trees:**
  - *Peers advertise what they expect to have in the future*
  - The path in a tree is a “chain of promises”
  - But this doesn’t mean trees are fragile per se: a tree can repair itself
    - Fairly simply…
Evaluation criteria

- **Delay**
  - Rather subtle

- **Overhead**
  - Trees are good…meshes can amortize at high volume

- **Simplicity**
  - Trees not as bad as you might think

- **Robustness**

- **Control over send load**
  - Chunkyspread good…not sure where meshes stand
Causes of delay

- **Mesh:**
  - Sender buffers a block of data
  - Advertises block to neighbors
  - Neighbors request block
  - Does this every hop

- **Tree**
  - When failure:
  - Detect interruption in data flow
  - Repair tree (start data flow from new parent)

- #hops x buffering time
  - Trade-off between overhead and delay
Key observation:

- If tree can repair faster than mesh buffering time (x #hops), **then trees should always perform better than meshes!**

- Why?----worst case, tree nodes always buffer for time of tree repair
  - Play out of buffer when parent is lost until tree repaired
Chunkyspread:
1. Build sparse random mesh

Built scalably with random walks (Swaplinks, Infocom ‘06)
Chunkyspread:
1. Build sparse random mesh

Control over node degree (heterogeneity)
Chunkyspread:
2. Stream source selects random slice sources
Chunkyspread:
3. Each slice source is root of slice tree
Chunkyspread: Loop avoidance and detection

- Each data packet contains path to slice source
  - Parent, parent’s parent, etc.
  - Compressed using Bloom filter [Whitaker ’02]
- Detect loop in one data packet cycle
- Each peer tells its neighbors its current path for each slice
  - Don’t select neighbor if loop would result
Chunkyspread: Parent selection

- For each slice, select a parent from among neighbors based on several criteria:
  - Avoid loops
  - Consider load on parent
    - Peers advertise desired load (heterogeneity)
  - Minimize delay
    - Simple method of estimating delay for each slice
Quality of load balance

Roughly 5:1 ratio of node capacities
Recovery from ancestor failure

CDF of Total Playback Disruption Durations

Pareto churn with 300 sec mean

Size of playout buffer

Total across 1000 second simulation
Some conclusions

- Tree-based protocols not as complex as you might think
- Tree-based has less overhead
- Tree-based probably performs better for latency
- Only useful for live streaming
- More to come....