Chainsaw: Eliminating Trees From Overlay Multicast

Vinay Pai Kapil Kumar Karthik Tamilmani Vinay Sambamurthy Alexander E. Mohr
{vinay,kkumar,tamilman,vsmurthy,amohr}@cs.stonybrook.edu

Department of Computer Science
Stony Brook University

International Workshop on Peer-to-Peer Systems 2005

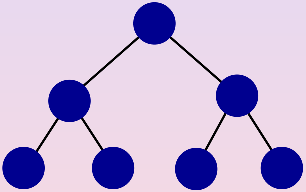
Problem Statement

Problem Statement

Design an overlay multicast system that:

- Delivers high bandwidth
- Supports a large number of simultaneous users
- Incurs little or no packet loss
- Minimizes duplication of data
- Is robust to large-scale node failure

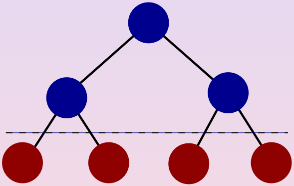
Traditional Approach: Multicast Trees



Shortcoming of Trees

- Rigid structure

Traditional Approach: Multicast Trees

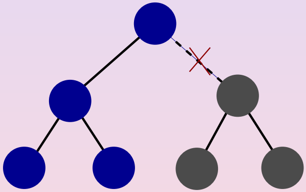


Leaf nodes don't upload!

Shortcoming of Trees

- Rigid structure
- Unfair sharing of load

Traditional Approach: Multicast Trees

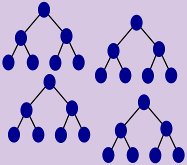


Shortcoming of Trees

- Rigid structure
- Unfair sharing of load
- Error propagation

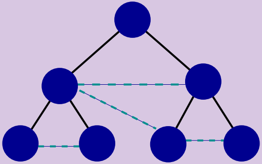
Solutions

Splitstream



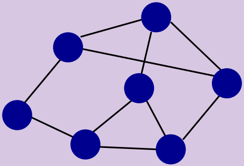
More Trees!

Bullet



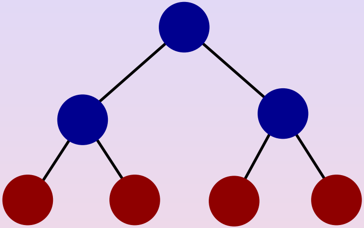
Tree overlaid with mesh

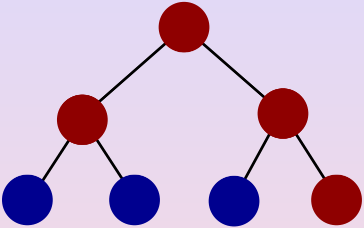
Chainsaw



Get rid of the trees

Splitstream

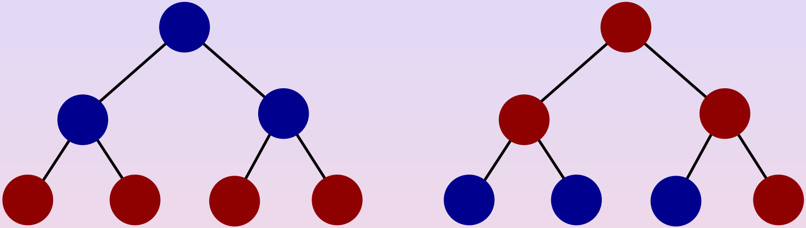




Splitstream: Multiple Trees

- A node is interior in at most one tree
- Improves fairness

Splitstream



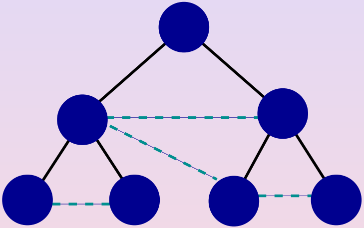
Splitstream: Multiple Trees

- A node is interior in at most one tree
- Improves fairness

Limitation

- Only partially mitigates effect of packet loss/node failure

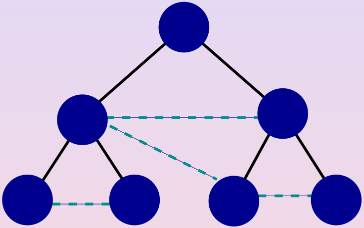
Bullet



Bullet: Tree+Mesh

- Most data sent over the tree
- Missing packets recovered using mesh
- Improves performance vs. pure tree

Bullet



Bullet: Tree+Mesh

- Most data sent over the tree
- Missing packets recovered using mesh
- Improves performance vs. pure tree

Limitation

- Does not fully address the issue of fairness—leaf nodes still likely to upload very little

Other

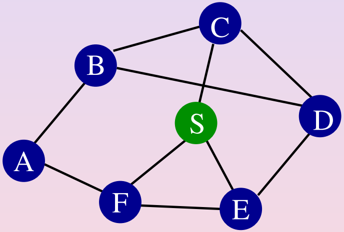
BitTorrent: Mesh-based file sharing

- File sharing not overlay multicast!
- But is similar to our approach

Others

- Gossip-based protocols
- End System Multicast
- TMesh

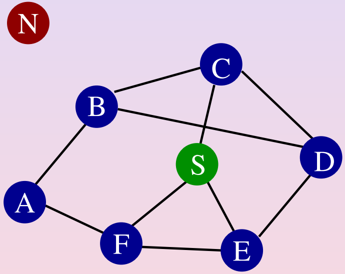
Network Structure



Random Graph Structure

- Nodes are connected randomly with some average degree
- Seed node S injects new data into the network

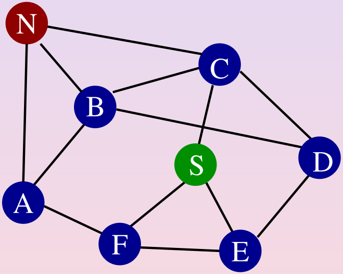
Network Structure



Random Graph Structure

- Nodes are connected randomly with some average degree
- Seed node S injects new data into the network
- New node N joins the system

Network Structure



Random Graph Structure

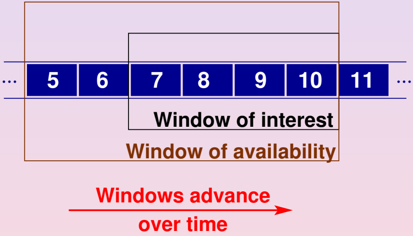
- Nodes are connected randomly with some average degree
- Seed node S injects new data into the network
- New node N joins the system
- N picks a random set of nodes to connect to

Packet Stream

Packet Stream

- Stream is broken up into packets
- Packets are assigned sequence number
- Assume (for this talk) that packets are all equal in size

Packet Stream



Windows

- Attempt to download packets within *Window of Interest*
- Packets that don't arrive before they "fall off the edge" are considered lost
- Offer neighbors packets within *Window of Availability*

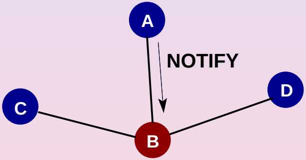
State Maintained

State

Only local state is maintained!

- List of neighbors
- Packets available at each neighbor
- List of potential neighbors (to replace dead ones)

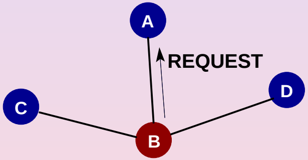
Protocol



Request-Response Protocol

- Node A gets a new packet and informs node B

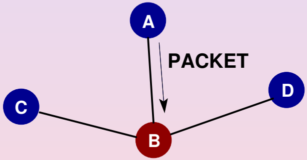
Protocol



Request-Response Protocol

- Node A gets a new packet and informs node B
- Node B makes a list of packet it is interested in
- Node B picks from the list to request

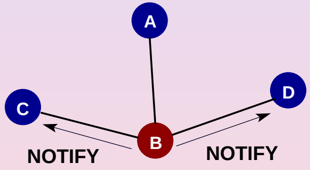
Protocol



Request-Response Protocol

- Node A gets a new packet and informs node B
- Node B makes a list of packet it is interested in
- Node B picks from the list to request
- Node A responds by sending the packet

Protocol



Request-Response Protocol

- Node A gets a new packet and informs node B
- Node B makes a list of packet it is interested in
- Node B picks from the list to request
- Node A responds by sending the packet
- Node B informs its other neighbors C and D

Request Strategy

Which packet to request?

Question: Given the list of packets a neighbor has that you're interested in, which do you request?

Request Strategy

Which packet to request?

Question: Given the list of packets a neighbor has that you're interested in, which do you request?

Potential Choices:

- Random
    - Some packets may not get picked from the seed for a long time

Request Strategy

Which packet to request?

Question: Given the list of packets a neighbor has that you're interested in, which do you request?

Potential Choices:

- Random
  - Some packets may not get picked from the seed for a long time
- Rarest First
  - Biases all nodes towards same set of packets

Request Strategy

Which packet to request?

Question: Given the list of packets a neighbor has that you're interested in, which do you request?

Potential Choices:

- Random
  - Some packets may not get picked from the seed for a long time
- Rarest First
  - Biases all nodes towards same set of packets
- Earliest First
  - Biases all nodes towards same set of packets
  - Increases delay by not picking new packets

Request Strategy

Successful Strategy

Nodes use random strategy, seed is smarter.
If the seed has unsent packets and it receives a request for a previously sent packet, it answers the request with an unsent packet instead.

Chainsaw scales to 10,000 nodes

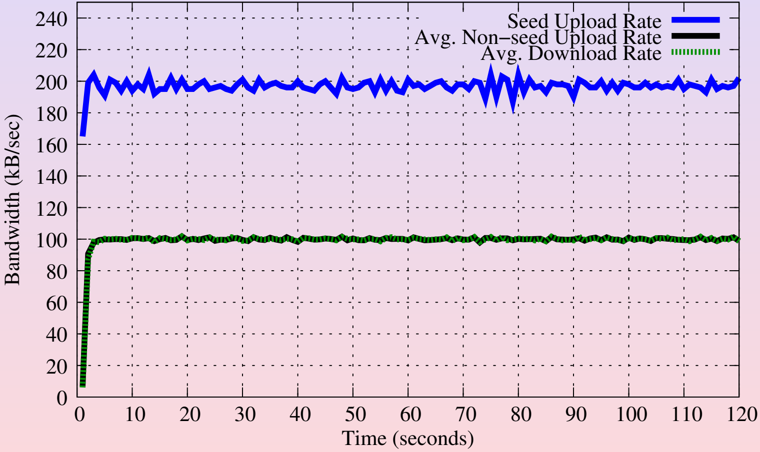
Claim

- Chainsaw consistently delivers high bandwidth to a large number of nodes

Simulator Setup

- Stream: 100 kB/sec with 1000 byte packets
- 10,000 node graph with minimum degree 30
- Seed capacity: 200 kB/sec
- Non-seed capacity: 120 kB/sec
- Round-trip time between nodes: 50 ms
- Buffer size: 500 packets (5 sec)

Bandwidth


Seed Upload Rate
Avg. Non–seed Upload Rate
Avg. Download Rate
Bandwidth (kB/sec)
0
20
40
60
80
100
120
140
160
180
200
220
240
0
10
20
30
40
50
60
70
80
90
100
110
120
Time (seconds)

No Packet Loss

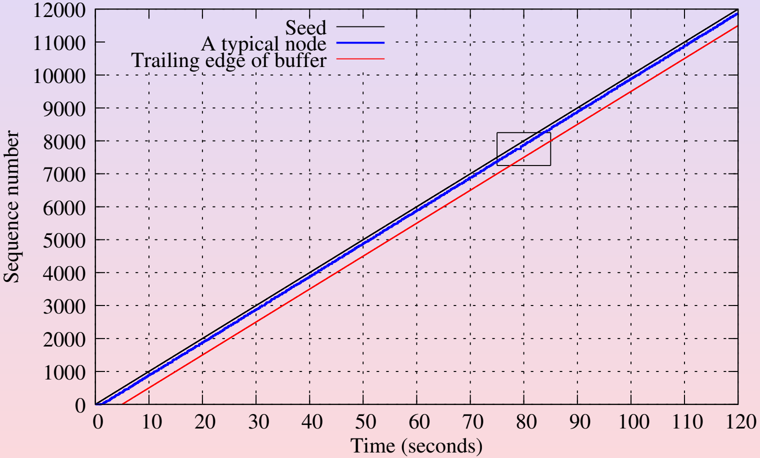
Claim

- Chainsaw loses virtually no packets

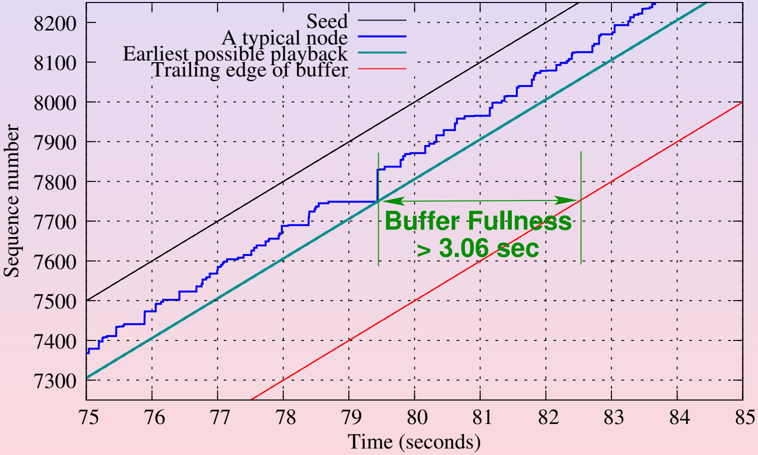
Simulator Setup

- Same as before

Progress Graph



Progress Graph (Zoomed)



Low Startup Delay

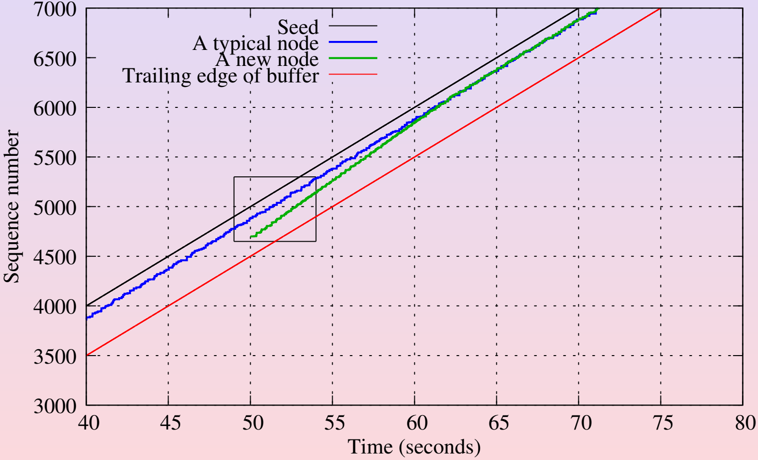
Claim

- A new node can start downloading quickly

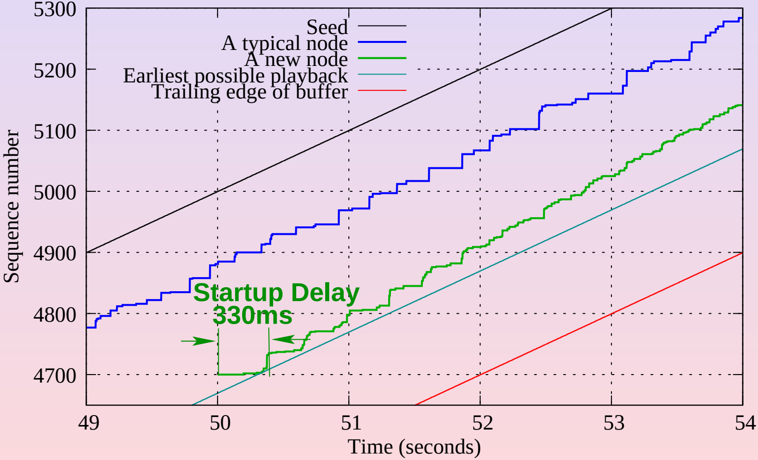
Simulator Setup

- Basic setup identical to previous experiment
- One node started 50sec later than the rest
- Startup strategy:
  - Begin requesting 3 sec old pieces
  - Request sequentially

Behavior of New Node



Behavior of New Node (Zoomed)



Real-world Comparison to Bullet and Splitstream

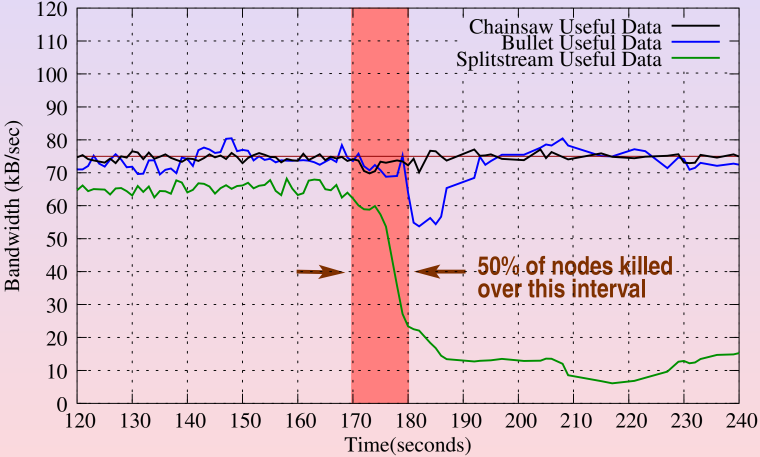
Experimental Setup

- Implemented Chainsaw using Macedon
- Macedon includes implementations of Bullet and Splitstream
- Ran trials on 174 Planetlab nodes
- Stream rate: 75 kB/sec
- 50% nodes terminated after 3 min to simulate failure

Limitation of Splitstream Implementation

At the time we ran our experiments, Macedon's Splitstream implementation did not implement the recovery mechanism. Therefore the behavior of Splitstream following the failure is not due to the limitations of the protocol.

Bandwidth


Chainsaw Useful Data
Bullet Useful Data
Splitstream Useful Data
50% of nodes killed
over this interval
Bandwidth (kB/sec)
Time(seconds)
0
10
20
30
40
50
60
70
80
90
100
110
120
120
130
140
150
160
170
180
190
200
210
220
230
240

Packet Loss

Splitstream

- All nodes suffered packet loss
- Average packet loss rate: 14%
- (ignore behavior after nodes fail)

Bullet

- All nodes suffered packet loss
- Packet loss rate: 0.88% - 3.64%
- Loss rate unaffected by node failure

Chainsaw

- 98% of the nodes had zero packet loss
- Two nodes suffered <0.05% packet loss
- One overloaded node suffered 60% loss

Duplicate Data

Splitstream

- Tree-based design eliminates duplicate data
- No duplicate packets observed

Bullet

- RanSub algorithm makes duplication likely
- Nodes received 5-10% duplicate data on average

Chainsaw

- Spurious timeouts may cause duplicate requests
- Observed duplicate data rate $<1\%$

DVD Streaming Over Planetlab

Experimental Setup

- Native C implementation
- 8kB packet size
- 4 Mbit stream (Comparable to DVD rate)

Results

- 230 nodes received stream with zero loss
- 920 Mbit aggregate bandwidth!
- Protocol overhead (including TCP/IP headers, etc.): 10%

Future Work

Future Work

- Churn
  - We have not experimented with dynamic joins and leaves
  - Unstructured architecture: expected to work even with a fraction of neighbors are working at a given time

Future Work

Future Work

- Churn
  - We have not experimented with dynamic joins and leaves
  - Unstructured architecture: expected to work even with a fraction of neighbors are working at a given time
- Non-cooperative Environments
  - So far we assume nodes upload willingly
  - When total capacity is scarce, we wish to penalize those that don't upload first

Future Work

Future Work

- Churn
  - We have not experimented with dynamic joins and leaves
  - Unstructured architecture: expected to work even with a fraction of neighbors are working at a given time
- Non-cooperative Environments
  - So far we assume nodes upload willingly
  - When total capacity is scarce, we wish to penalize those that don't upload first
- Piece-picking Strategy
  - Being smarter than Random may yield better performance
  - Better ways of taking delay and rarity into account

Conclusion

Conclusion

We have shown that:

- Overlay multicast over an unstructured network is feasible
- The architecture can scale to a large number of nodes
- Packet loss can be virtually eliminated
- The system is resilient to catastrophic node failure
- Real-world tests corroborate simulation results

Questions?

Questions?

Robust to Catastrophic Failure

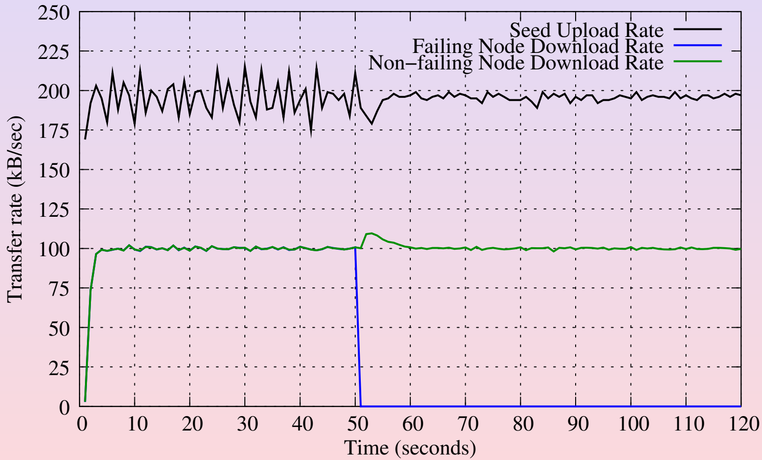
Claim

- Chainsaw continues to work even in the face of massive simultaneous node failure
- Nodes do not lose packets so long as they have enough neighbors

Simulator Setup

- Stream: 100 kB/sec with 1000 byte packets
- 10,000 node graph with minimum degree 40
- Seed capacity: 200 kB/sec
- Non-seed capacity: 120 kB/sec
- Round-trip time between nodes: 50 ms
- Buffer size: 1000 packets (10 sec)

Bandwidth


Seed Upload Rate
Failing Node Download Rate
Non–failing Node Download Rate
Transfer rate (kB/sec)
Time (seconds)
0
25
50
75
100
125
150
175
200
225
250
0
10
20
30
40
50
60
70
80
90
100
110
120

Progress Graph