

Streaming Content Delivery In Peer-to-Peer Networks



Daniela Gavidia and Michal Szymaniak

Vrije Universiteit Amsterdam

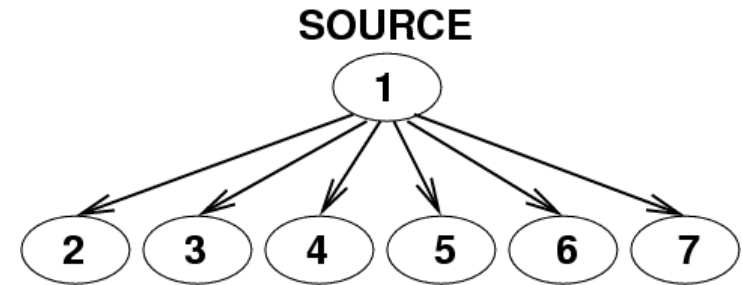
{daniela,michal}@cs.vu.nl

Context: Cooperative Environments

- (Large) computer network
- Participating nodes contribute resources to join the system
- Contributors expect gain proportional to their contribution
 - e.g., incoming bandwidth == outgoing bandwidth
- Highly-dynamic structure
 - nodes come and go all the time
- Examples:
 - peer-to-peer file sharing platforms (BitTorrent, Kazaa, etc.)

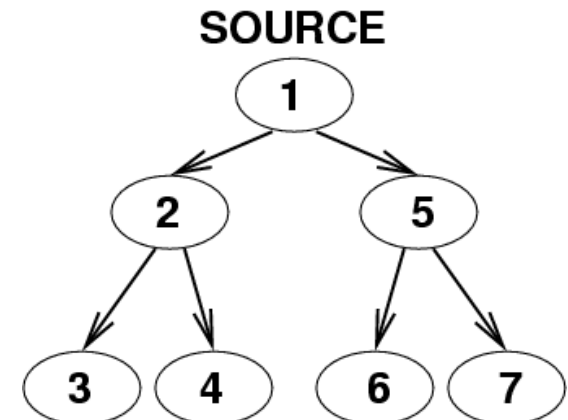
Goal: Multicast

- Multicast: one-to-many
 - single node acts as a content sender (1)
 - multiple nodes receive content (2-7)
 - e.g., video streams, bulk file archives



- Many sender-to-receiver connections inefficient
 - **redundancy** (source repeatedly transmits the same data)
 - **physical constraints** (source has limited bandwidth)

- Instead: multicast tree
 - organize nodes into a tree
 - content sender == root (1)
 - internal nodes receive and forward content (2,5)
 - leaves only receive (3,4,6,7)



Problem: Cooperative Multicast

- Fundamental conflicts between multicast and cooperative networking:

- **unbalanced node responsibilities**

- a few nodes forward, the others just receive

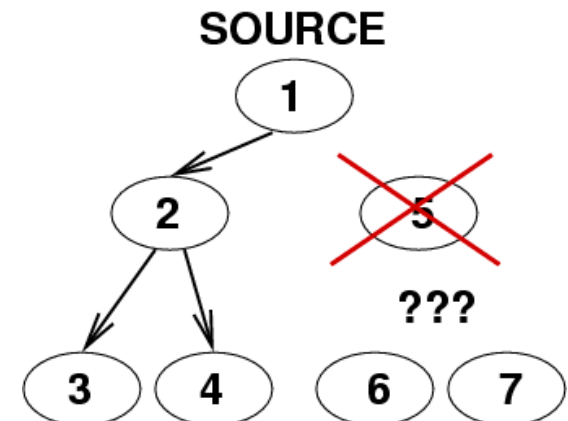
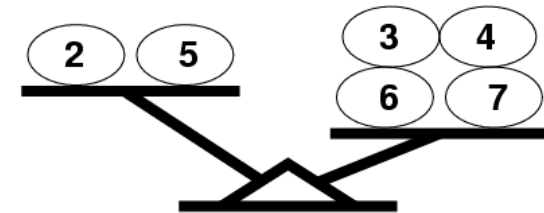
- **unbalanced contribution and gain**

- tree-internal nodes: bandwidth contribution \gg bandwidth gain
- leaves: bandwidth contribution $== 0$

- **leaving tree-internal nodes spoil the tree**

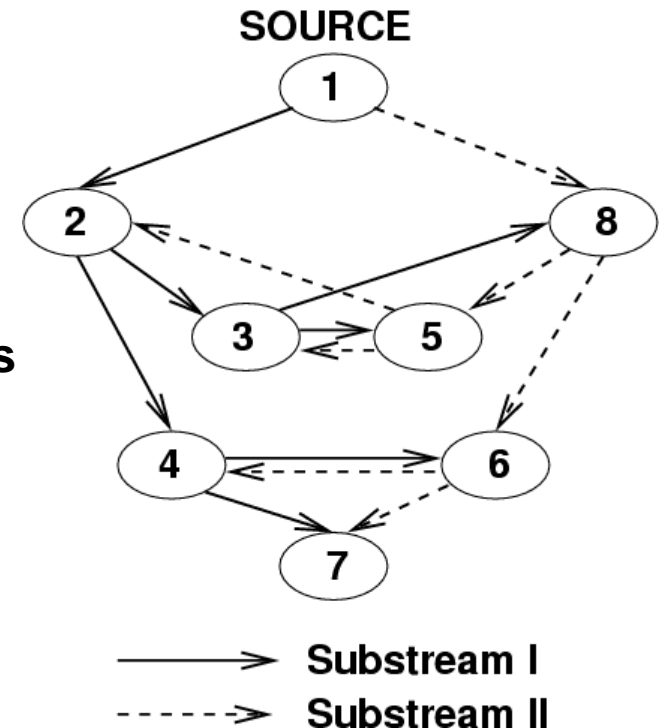
- ..and all the nodes below become disabled

- What can we do about it?



Solution: Forest Multicast

- Forest: multiple multicast trees
 - The original content stream split into **k** substreams
 - All substreams **equally important**:
 - Any $m < k$ substreams give the original stream
 - Erasure Coding, Multiple Description Coding
 - Substream trees have **different internal nodes**
 - Solves single-tree limitations:
 - **Load distribution**
 - different forwarders in different trees
 - **Resilience to forwarder failures**
 - redundant content coding



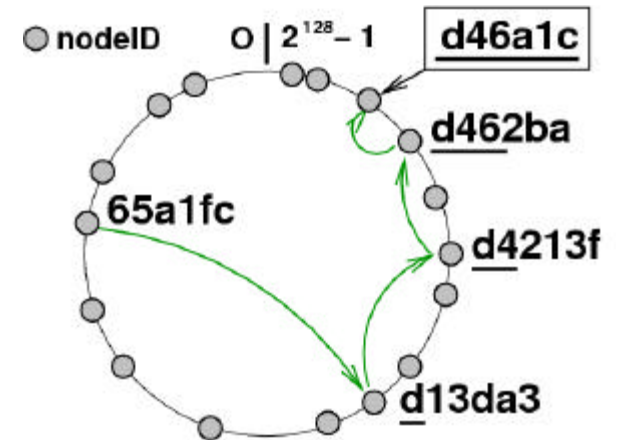
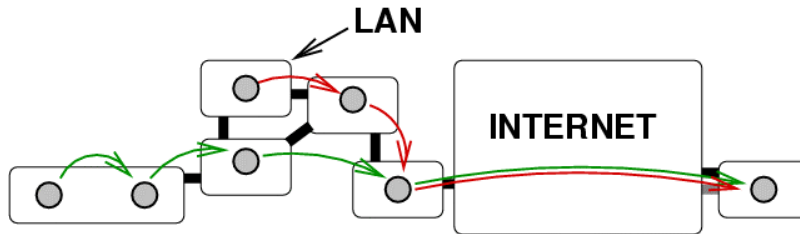
- **How to build a forest of internal-node-disjoint trees?**

Forest Multicast: Implementation

- Distributed forest construction using Pastry/Scribe

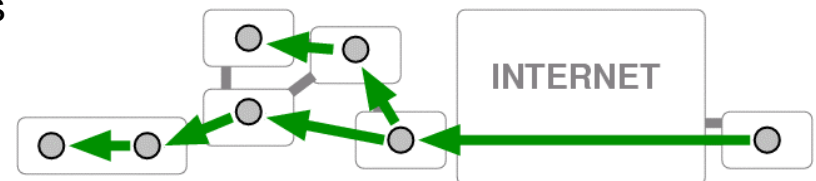
- Pastry:

- self-organizing peer-to-peer overlay
 - nodes have random numerical nodeIDs
 - one routing step ~ one digit in target nodeID
 - **low route delay penalty**
 - **local route convergence**



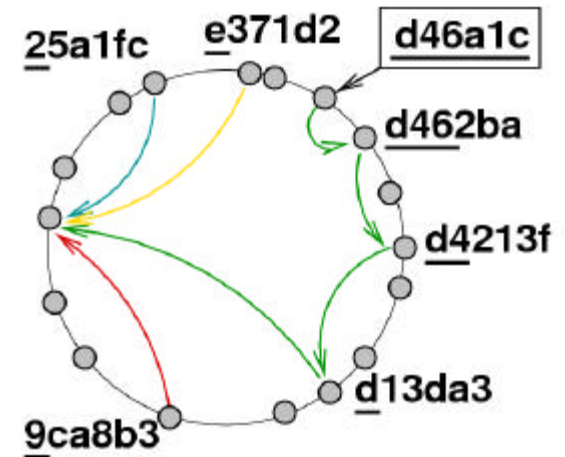
- Scribe:

- single-tree multicast over Pastry
 - reverse-path forwarding
 - So: NodeIDs of root and internal nodes have **the same 1st digit**



SplitStream

- SplitStream -- Forest multicast based on multiple Scribe trees
 - Different trees -- **different roots**
 - Different roots -- **different 1st digits** in NodeIDs
 - Scribe:
 - 1st digit the same for root and internal nodes
 - So: **node can be a forwarder of at most one substream**
- Properties:
 - **global function/load balancing**
 - **resilience to forwarder failures**
 - **higher propagation delay** than in Scribe (more synchronization needed)
 - **uses more network links** than Scribe (multiple trees)
 - ..but **incurs lower per-link load** (less data/stream)



Is SplitStream really that great?

- SplitStream tweaks multicast to work in cooperative environments:
 - **balanced responsibilities and load**
 - **resilience to node failures**
- But it also assumes that:
 - **nodes contribute bandwidth** (otherwise new nodes cannot join)
- Meanwhile, the reality is that:
 - **nodes are not willing to contribute**
 - in 2000, about **70%** of Gnutella users shared no files (and hence no bandwidth)
 - **50%** of all requests were serviced by **1%** of (highly-altruistic) nodes

[Freeriding on Gnutella]
- So how reasonable is it to rely on node-contributed resources?
- Can we run resource-demanding apps in peer-to-peer systems?