# Peer-to-Peer Networks

Chapter 5-3: Application Layer Multicast/P2P-based IPTV

Thorsten Strufe

# Agenda

- Motivation for an ALM
- Background
- Exemplary systems
  - Tree-push systems:
    - Narada
    - Banana Tree Protocol
  - Mesh-pull system: DONet
  - Hybrid systems:
    - Coolstreaming
    - mTreebone
- Importance of resilience
  - SplitStream
  - Optimally DoS-resistant P2P streaming

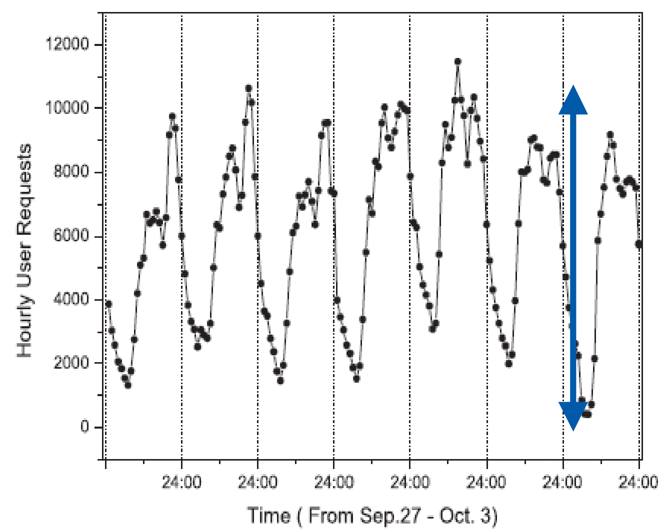
# Video Distribution on the Internet

- Video distribution becomes a "killer application" in the current Internet
  - YouTube: 45 terrabytes of videos and 1.73 billion views by August 2006 [Liu2008]

- The playback rate is still low!
  - 200-400 kbit/s due to resource constraints
  - E.g. DVB-T around 3-3.5 Mbit/s

- High cost
  - E.g. YouTube 10$/Mbps
  - Results in 1 Mio $ per day [Credit Suisse 2009])

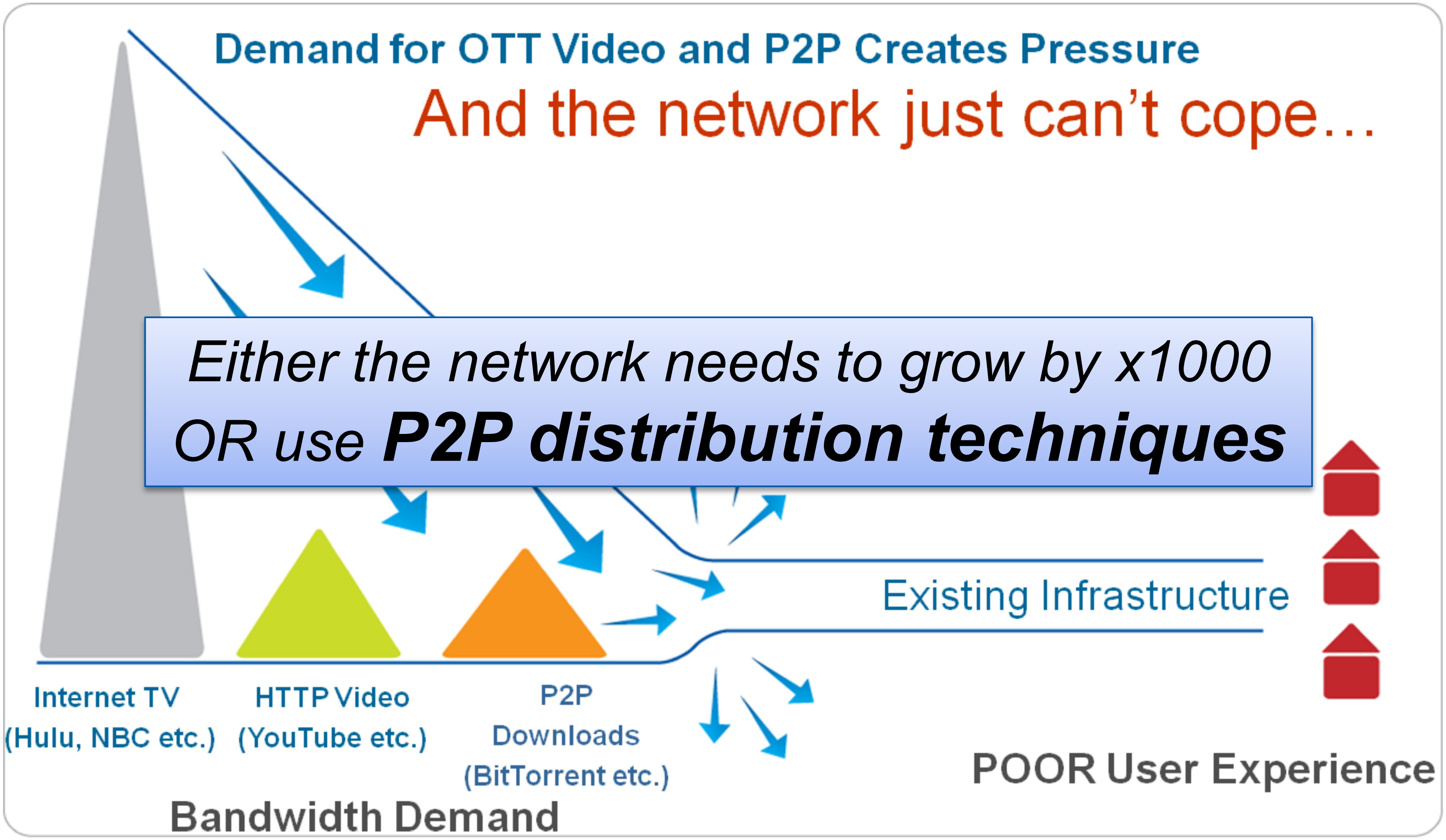
- Flash crowds make content unavailable
  - Especially for small content providers




- *How to distribute a large number of large files in a cost- and resource-efficient manner?*

source: Yu2006

# The Bandwidth Challenge



# Internet Multimedia Streaming

- Possible types of providers
  - ISP independent streaming services (youtube, vimeo,…) „Web-TV“?
  - TV infrastructure inside single ISP (t-home, …) „IPTV“
  - Citizen reporters (`http://www.ustream.tv/ http://bambuser.com`)
  - V-logs (Dr. Horribles Sing-Along Blog)
  - ***Possibly:*** *Virtual Living Room (distributed public viewing)*

- Types of video streaming
  - Classified by time of capturing
  - Classified by audience

# Types of Media Streams

| Streaming Approach | **Live** | **Recorded** |
|---|---|---|
| **Selected Audience** | Video Conferences | Video on Demand |
| **Disperse Audience** | Live TV | Recorded Programs |

# Properties wrt. Time of Capturing

- Access patterns of live streams
  - object-driven (user has passive role)
  - depends on content's schedule
  - e.g., boxing match at 4 a.m.
- Access patterns of recorded streams
  - user-driven (user has active role)
  - depends on preferences and user's schedule
  - e.g., recording of favorite TV show after work
- Correlation between various variables differs
  - e.g., length of viewing time and QoS
  - e.g., arrival frequency vs. time of day

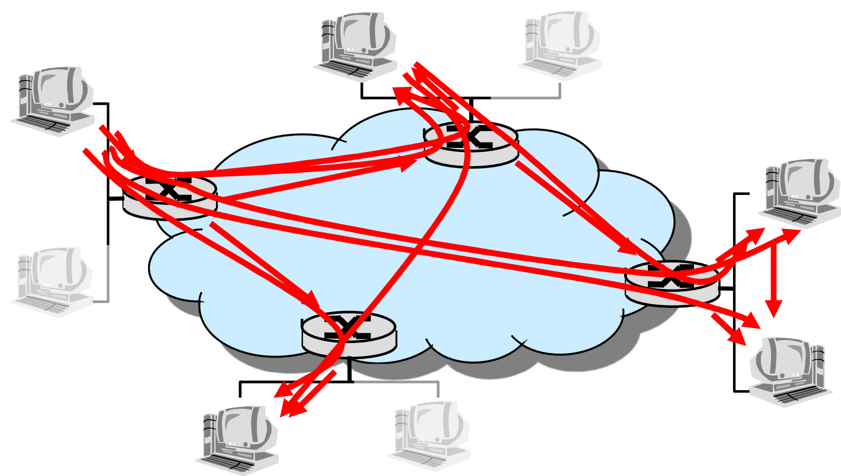
# Rough Requirements of Streaming

- Functional
  - Broadcast of channels (many)
  - Location of content (channels, groups, public discussions)
  - Group creation and management
  - Location of friends/contacts
  - Direct local multicast (group-/multi-party communication)
    - Video
    - (Voice)

- Non-Functional
  - Low distribution cost
  - responsiveness (channel surfing)
  - delay bounds (interactive tv)
  - availability (commercial deployment)
  - robustness
  - resilience
  - access control and accounting
  - privacy preservation (personal information!)
  - scalability…

# Transmission of Live Multimedia Streams

- Client/Server
  - Duplication of the packets at server
  - No scalability to groups / sessions
  - Vulnerable to attacks
- Network-Multicast
  - Duplication at routers
- Application Layer Multicast*
  - Duplication at end hosts
  - *Are there possibilities to build scalable and robust solutions?*



*„Application Layer Multicast“ termed by Biersack et al. (1999) originally not in the context of P2P*

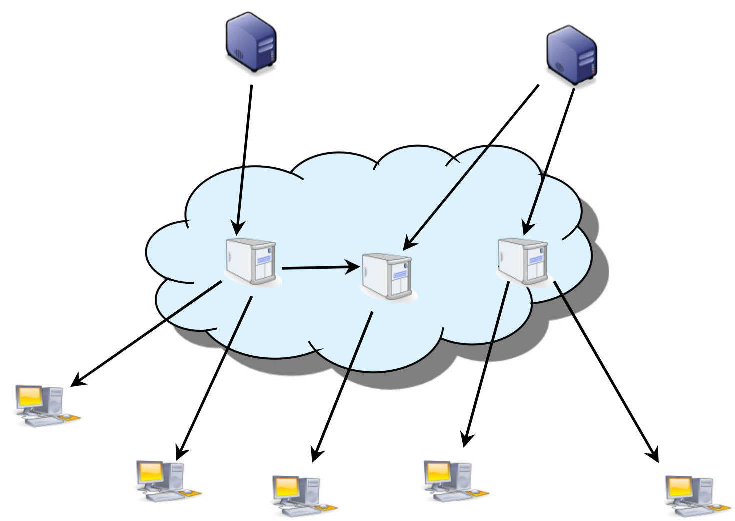
# Key Concerns with IP multicast

- Scalability to number of groups
  - Routers maintain per-group state
  - Aggregation of multicast addresses is complicated
- Supporting higher level functionality is difficult
  - IP Multicast: best-effort multi-point delivery service
  - End systems responsible for handling higher level functionality
  - Reliability and congestion control for IP Multicast complicated
- Inter-domain routing is hard
- Deployment is difficult and slow
  - ISP's reluctant to turn on IP Multicast
- ***IP multicast is vulnerable to a plethora of attacks***

# First Approach: Content Delivery Networks (CDNs)

- First (existing) possible solution

- Managed network of servers
  - Distributed across the Internet
  - Host content on demand for paying customers (content providers)



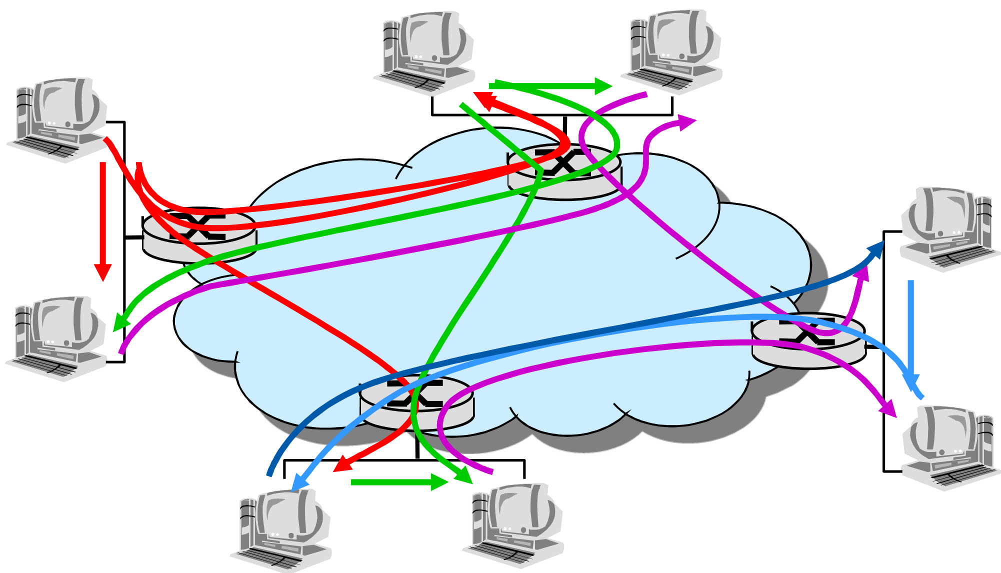
- How does it work?
  - The user request is redirected to a *close* CDN server
  - The content is served from servers caches

- Benefits
  - Faster response time (for cached content)
  - Less transit traffic
  - Better load balancing / scalability

- But
  - High cost
  - Limited flexibility / dynamics
  - Peer assistance can reduce server load by 66% (estimated in [Huang 06])

# Overall Goal: Cost Efficient Delivery of Multimedia

- *Harness resources of the participating end hosts*
- *Lookup of content and potential serving nodes*
- *Scalable construction of ALM overlays based on local knowledge*



***Common metrics:***

***Stress** (per link)**:** amount of Identical packets traversing the same physical link*

***Stretch** (per packet)**:** ratio of the overall sum of hops on the **overlay path**, divided by the number of hops on the **unicast path***

*Requirements for Streaming Topologies*

*Network Efficiency:*
- *Low redundancy: Stress*
- *Low delay: Stretch*

*Robustness in case of:*
- *Random node failures*
- *Deliberate attacks*

# General Functionality in ALM

- Two general functions: ***Location*** and ***Content Distribution***

- Location
  - Content (Streams / channels / programmes)
  - Possible sources:
    - neighbors (location)
    - „parents“ (content)

- Neighbor Selection
  - In signalling / location overlay
  - In transmission overlay

- Routing
  - Next-hop decision for location / signalling
  - Multicast routing (constructing / optimizing the distribution tree)

- Transmission (/Presentation)

# Classification of Different Approaches

- Communication mode
  - Video on demand (tweak BitTorrent, etc.) → ***Peer-to-Peer streaming***
  - Live streaming → ***Overlay live streaming***
  - Dialogue/conferences → ***Application layer multicast*** (is super group, too…)
- Signalling overlay
  - „Mesh-first“ vs. „Tree-first“
- Topology types
  - „Trees“ vs. „Meshes“ (multitude of covering trees)
- … or transmission paradigm
  - Pull-based vs. Push-based => „mesh-push“ or hybrid

# Different Signalling Overlays

- How is signalling traffic transferred?
- How is the streaming topology created?

- „Mesh-first“
  - Create an explicit signalling „mesh“ (overlay)
  - Streaming
    - in separate streaming overlay
    - along selected links of signalling mesh

- „Tree-first“
  - Plain (streaming-) neighbor selection
  - Signalling along same links
  - Only for single „channel“ topologies

# Dffrnt. Transmission Paradigms: Mesh/Pull

- Pull-based streaming *(„Mesh")*
  - Streams are divided into chunks
  - Each peer requests the chunks it needs
  - Simple example: BitTorrent, rarest first exchanged for request in order
  - → naive file-sharing like distributed download of stream
  - (+) very robust
  - (-) slow, high delays, significant signalling overhead (request each packet, but little compared to stream)
  - ***When can this be done?***
  - ***Video on Demand*** *(why?)*

  - Btw:

*Each packet still distributed along tree(s) (from virtual source before seeders)*

# Dffrnt. Transmission Paradigms: Tree/Push

- Pushed (subscription based) streaming
  - Explicit construction of treaming topology (explicit parent <-> child relations)
  - Parent forwards packet to child(ren) on reception
  - (+) little overhead, small delays
  - (-) less robust (topology repair on node failure/departure)

- Extended thoughts…
  - The whole stream is transferred along a tree (cmp. ip multicast)
  - Stream is split into partial streams („stripes“, descriptions), one tree each
  - ***When can this be done?***
  - ***Live-streaming, broadcast-like streaming…***
  - What about asynchronous access?

# Hybrid systems

- Motivation
  - Both Tree-push and Mesh-pull have pros and cons
  - Question: Can we combine them?
- Principle
  - Overlay: MESH
  - Transmission paradigm: PULL and PUSH
  - With buffer map exchange
  - ***What is the condition to switch?***
    - *As soon as possible, or*
    - *When peers are „stable"*
- Common properties
  - Lower delay and overhead
  - More robust in dynamic environments
- Is it really the answer? (resistant against attacks?)

# Potential Benefits of App.- Layer Multicast

- Scalability to group sizes
  - *Cheap!* (They come with their own resources!)

- Scalability to number of sessions in the network
  - Routers do not maintain per-group state
  - End systems do, but they participate in very few groups

- Easier to deploy

- Potentially simplifies support for higher level functionality
  - Leverage computation and storage of end systems
  - For example, for buffering packets, transcoding, ACK aggregation
  - Leverage solutions for unicast congestion control and reliability

# General Issues of P2P Streaming

- Topology needs to be
  - loop-free (but we don‘t even know who‘s online!?)
  - and „alive“ (enough BW to serve everyone)
- Reliability of peers
  - Peers subject to „churn“ (arrival/departure) due to decisions of user or crashes (network/peer)
  - Peers subject to significant „cross-traffic“ (other apps…! Even congestion…)
- Scalability
  - Peers depend on predecessors, long paths cause failures / delays / jitter
  - Topology needs to be created in a sensible way
  - Signalling overhead needed, but pot. adverse for streaming and scalability

# General Issues… (ctd.)

- Bandwidth variation
  - Cmp. cross traffic: bandwidth of peers varies
  - Difficult to estimate but needed for topology control (or accepting requests..)
  - Very heterogenous nodes and links between nodes
    - Access link: modem, dsl, t1
    - Well connected sub-networks (uni intranet) vs. Dsl (de) to dsl (aus)…

- Access patterns
  - Channel surfing asks for fast channel change
  - Asynchronous access to VoD (how many people watch the same youtube video concurrently?)

# Exemplary Systems

- Tree-push systems:
  - Narada
  - Banana-Tree
- Mesh-pull system:
  - DONet
- Hybrid systems
  - Coolstreaming
  - mTreebone

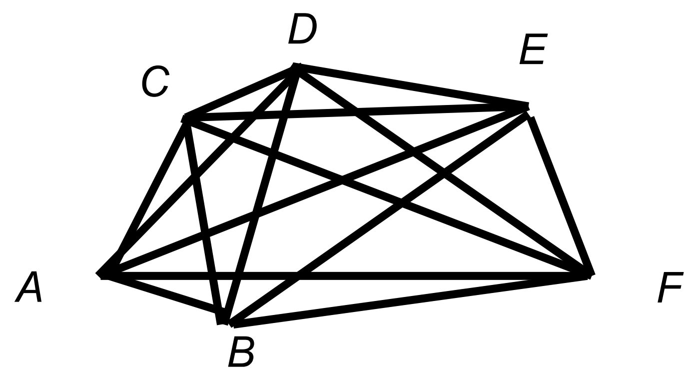
# Narada (End System Multicast)

- Carnegie Mellon University
- Yang-hua Chu, Sanjay G. Rao, and Hui Zhang
- 2000 (2002) SIGMETRICS (JSAC) (*the same time as napster and gnutella!)*
- „Narada“ -> „ESM“ -> „Conviva“ (with Ion Stoica..)

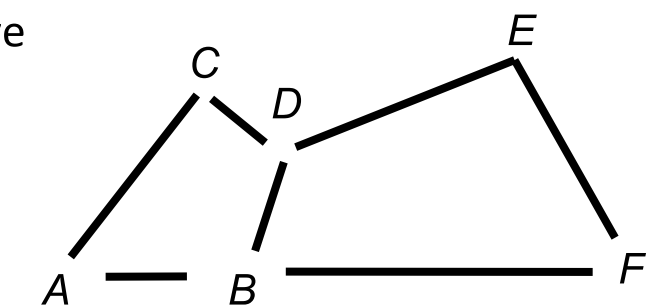
- Aim: implement an ALM for small, sparse groups
- (later: distribute streaming video „to a large number of people“)

# Narada Design

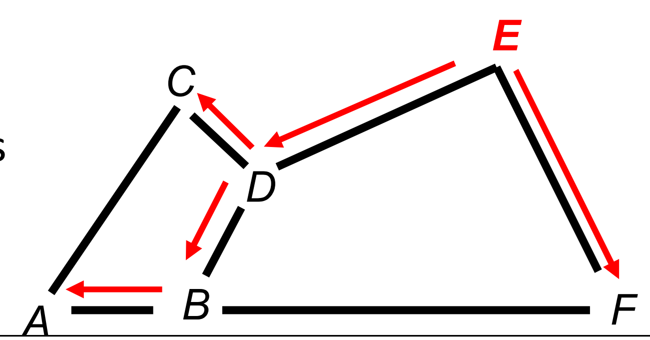
- Step 0
  - Maintain a clique overlay of all group members
  - Links correspond to unicast paths
  - Link costs maintained by polling



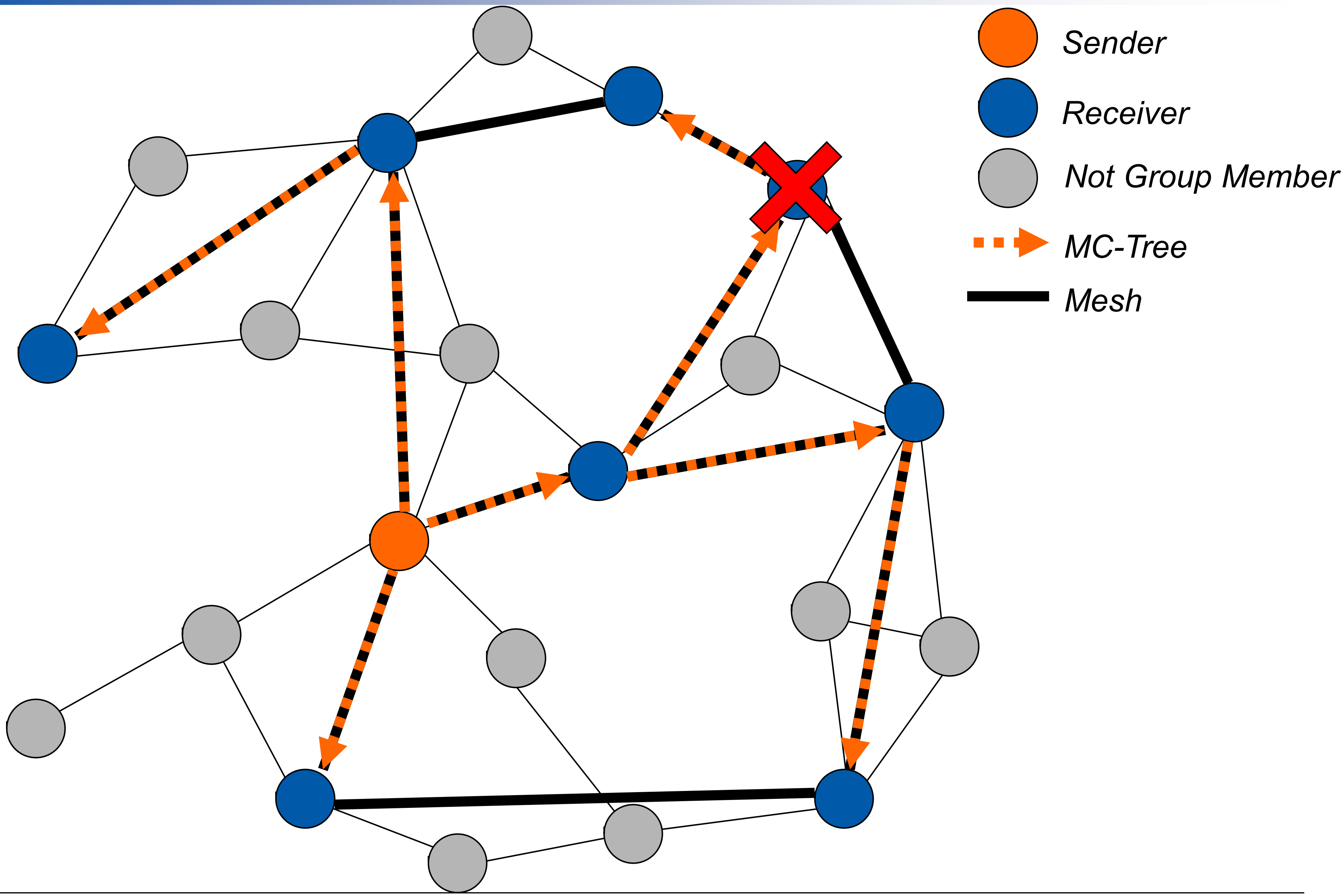

- Step 1
  - Create "mesh": Subset of complete graph (may have cycles and includes all group members)
  - Decrease degree of members
  - Shortest path delay between any pair of members along mesh is claimed to be small



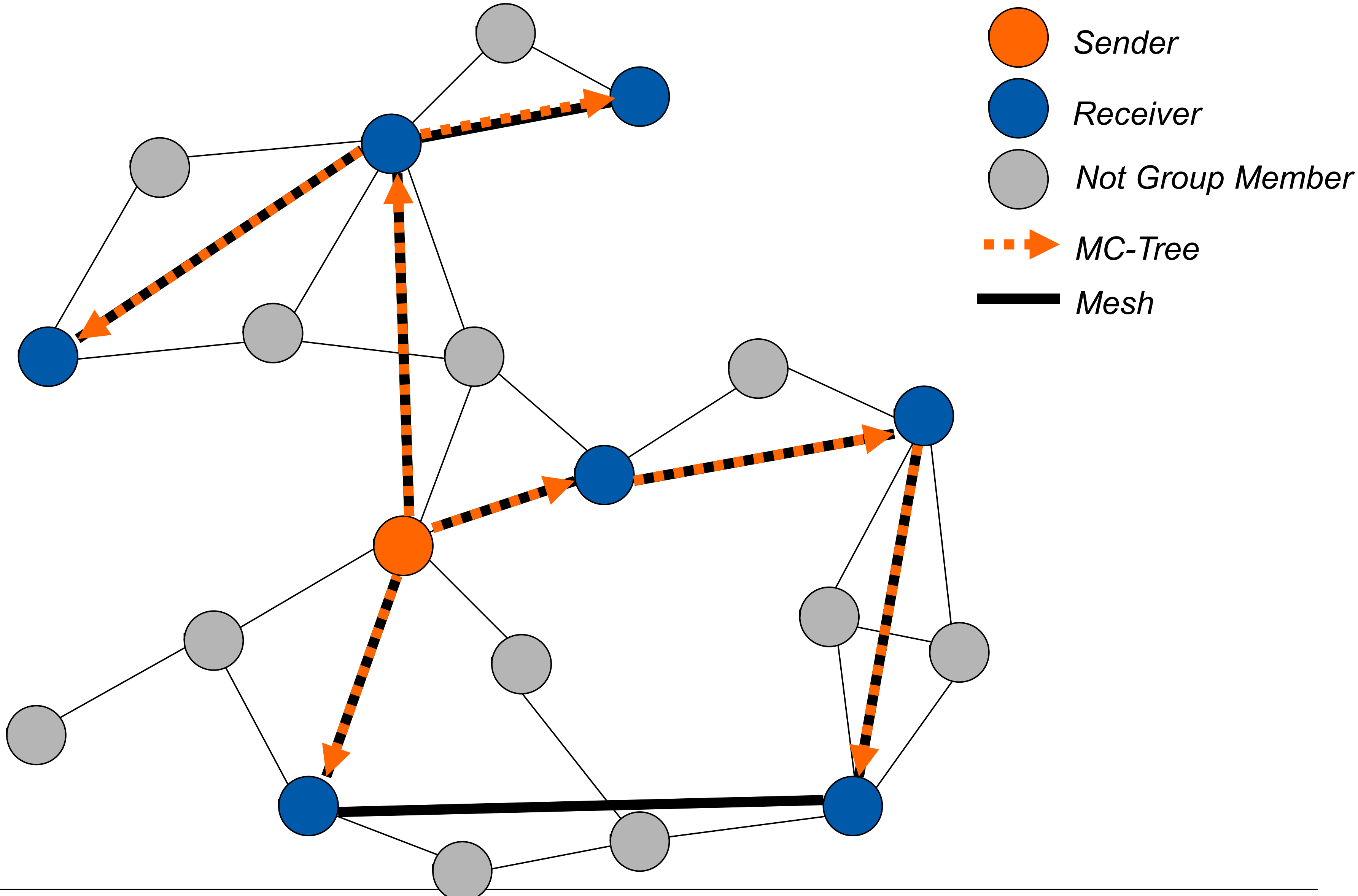

- Step 2
  - Build spanning tree within the mesh
  - Constructed using well known routing algorithms
  - Members have low degrees
  - Small delay from source to receivers



# Narada Example



# Narada Example



# Narada Components

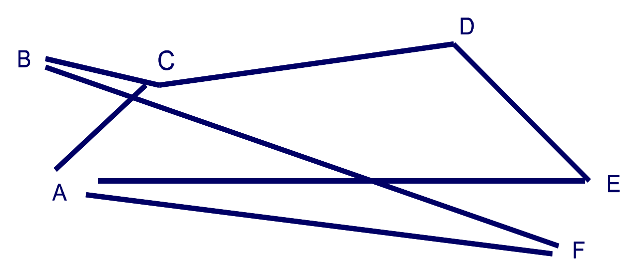
- Mesh Management
  - Ensures mesh remains connected in face of membership changes
- Mesh Optimization
  - Distributed heuristics for ensuring shortest path delay between members along the mesh is small
- Spanning tree construction
  - Routing algorithms for constructing data-delivery trees
  - Distance vector routing, and reverse path forwarding

***Reverse Path Forwarding works /exactly/ how?***

***Which information does /every/ peer need?***

# Optimizing Mesh Quality

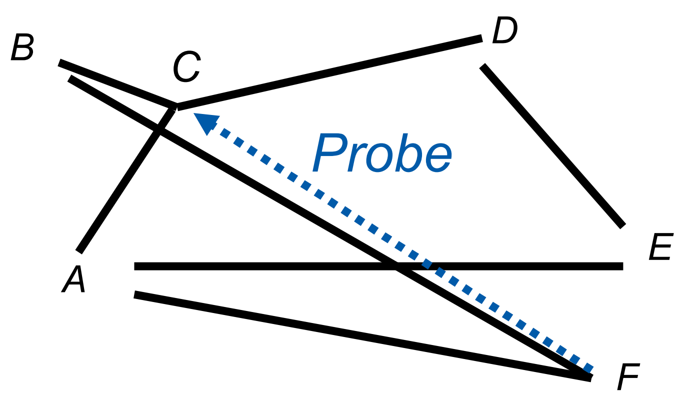
- Members periodically probe other members at random
- New link added if utility gain of adding link > add threshold
  - Based on: number of members to which routing delay improves, how significant the improvement in delay to each member is
- Members periodically monitor existing links
- Existing link dropped if cost of dropping link < drop threshold
  - Based on number of members to which routing delay increases, per neighbor
- Add/Drop thresholds are functions of:
  - Member’s estimation of group size
  - Current and maximum degree of member in the mesh



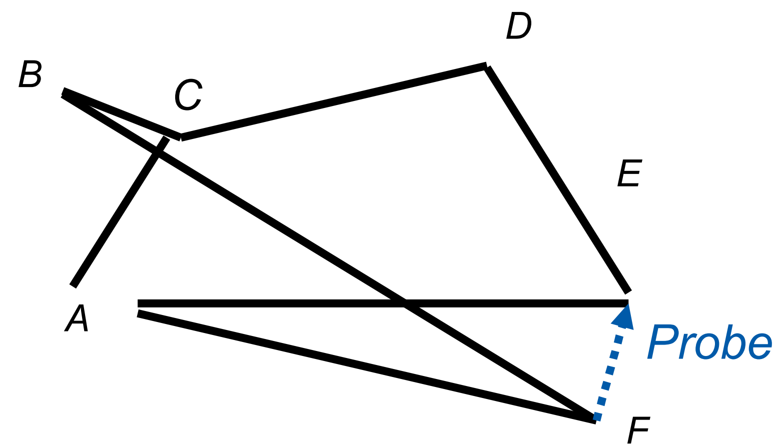

*A poor overlay topology*

# Desirable Properties (Requirements) of Heuristics

- Stability
  - A dropped link will not be immediately re-added
- Partition Avoidance
  - A partition of the mesh is unlikely to be caused as a result of any single link being dropped

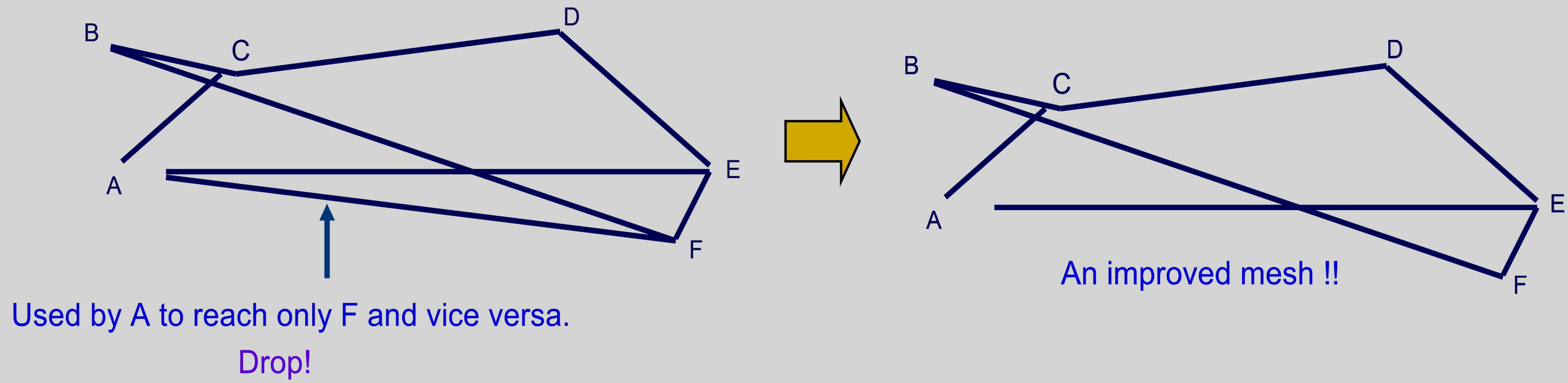



*Delay improves to C, D*
*but marginally.*
*Do not add link!*




*Delay improves to D, E*
*and significantly.*
*Add link!*

# Adding / Dropping Links contd., Overhead




***So the topology is efficient…***

***Can you imagine drawbacks of Narada?***
***What about despiteful parties?***

- Two sources of overhead
  - pairwise exchange of routing and control information
  - polling for mesh maintenance
- Claim: ratio of non-data to data traffic grows linearly with group size.
- Narada is targeted at small groups

# End System Multicast (ESM)

- Goal: enable conferencing on the Internet based on Narada
  - Study in context of real-world applications
  - Achieve acceptable performance, even in a dynamic/heterogeneous environments

- ESM = first detailed Internet evaluation to show the feasibility of ALM

- Why conferencing?
  - Important and well-studied
    - Early goal and use of multicast (vic, vat)
  - Stringent performance requirements
    - High bandwidth, low latency
  - Representative of interactive applications
    - E.g., distance learning, on-line games

# Banana Tree Protocol

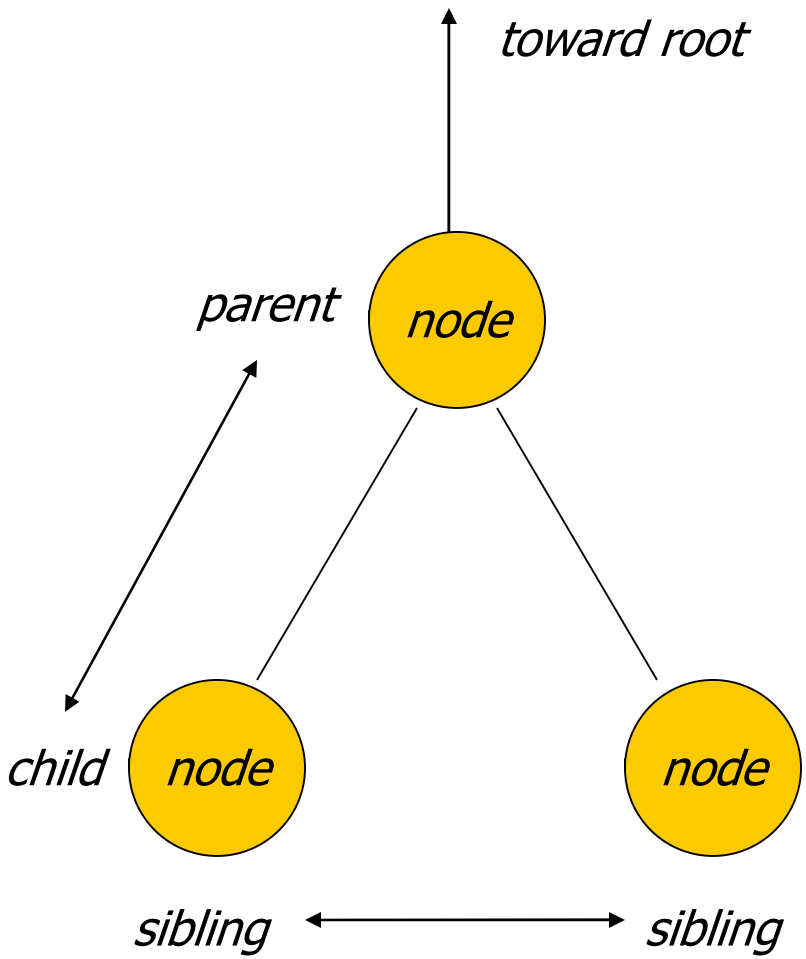
- University of Michigan
- David Helder and Sugih Jamin (RSVP, Zattoo…) in 2000

- Aim: ***tree-first creation*** of a tree-based overlay multicast

- Main approach:
  - Create a tree starting at a root
  - Join nodes at arbitrary node
  - Perform only local changes to adapt the tree
  - Next node on path to root is „parent“ (parents forward stream to children)
  - Children of same parent are „siblings“

*Source: www.eecs.umich.edu/techreports/cse/00/CSE-TR-429-00.pdf*
*http://nimrud.eet.bme.hu/ibmc/ppt/6ALM-ON-INTERNET/*

# The BTP Tree (Extract)



# Main Functionality

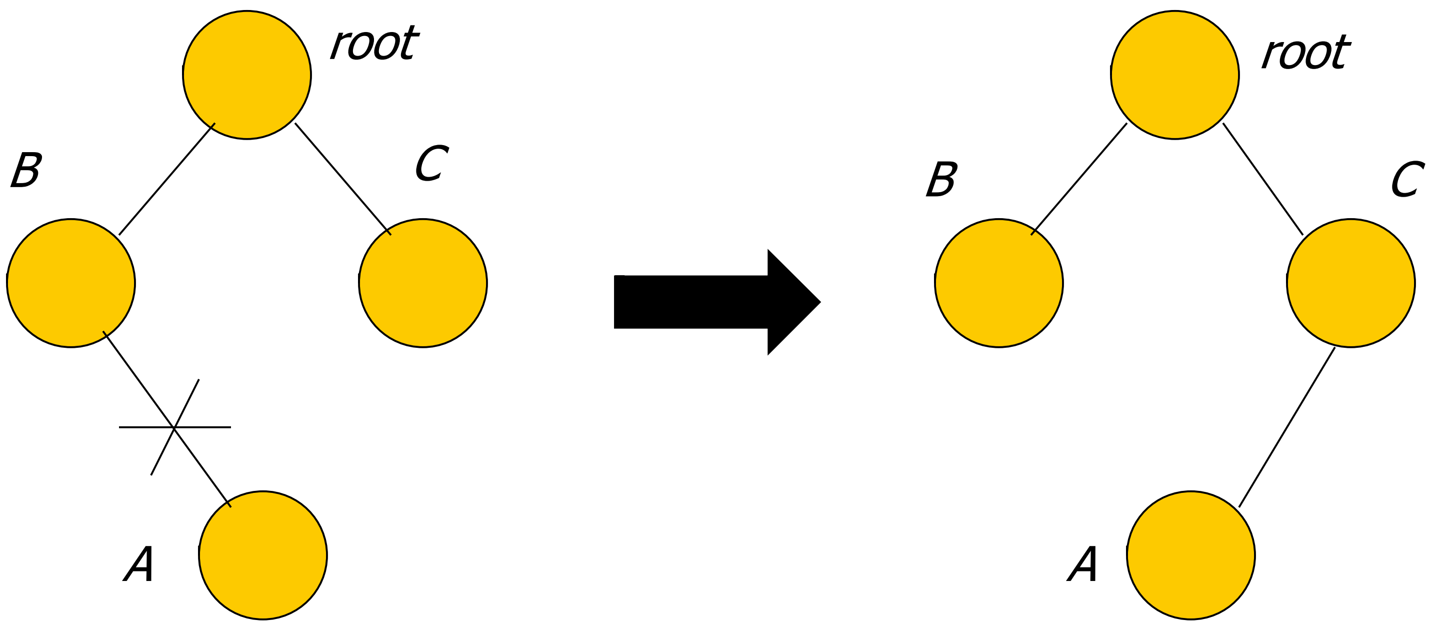
- Existence of a *bootstrap protocol* is assumed
- A host joins a group by becoming the child of a node currently in the tree (e.g., the root node)
- A node that joins an empty multicast group is the new root node

- Any node can multicast:
  - To multicast, a node sends the packet(s) to its neighbors
  - On reception of a packet, each node forwards it to all other neighbors

- In case of a departing parent the tree partitions, children reconnect to root
- Nothing is done on failure of child(ren), since successors will automatically reconnect to root

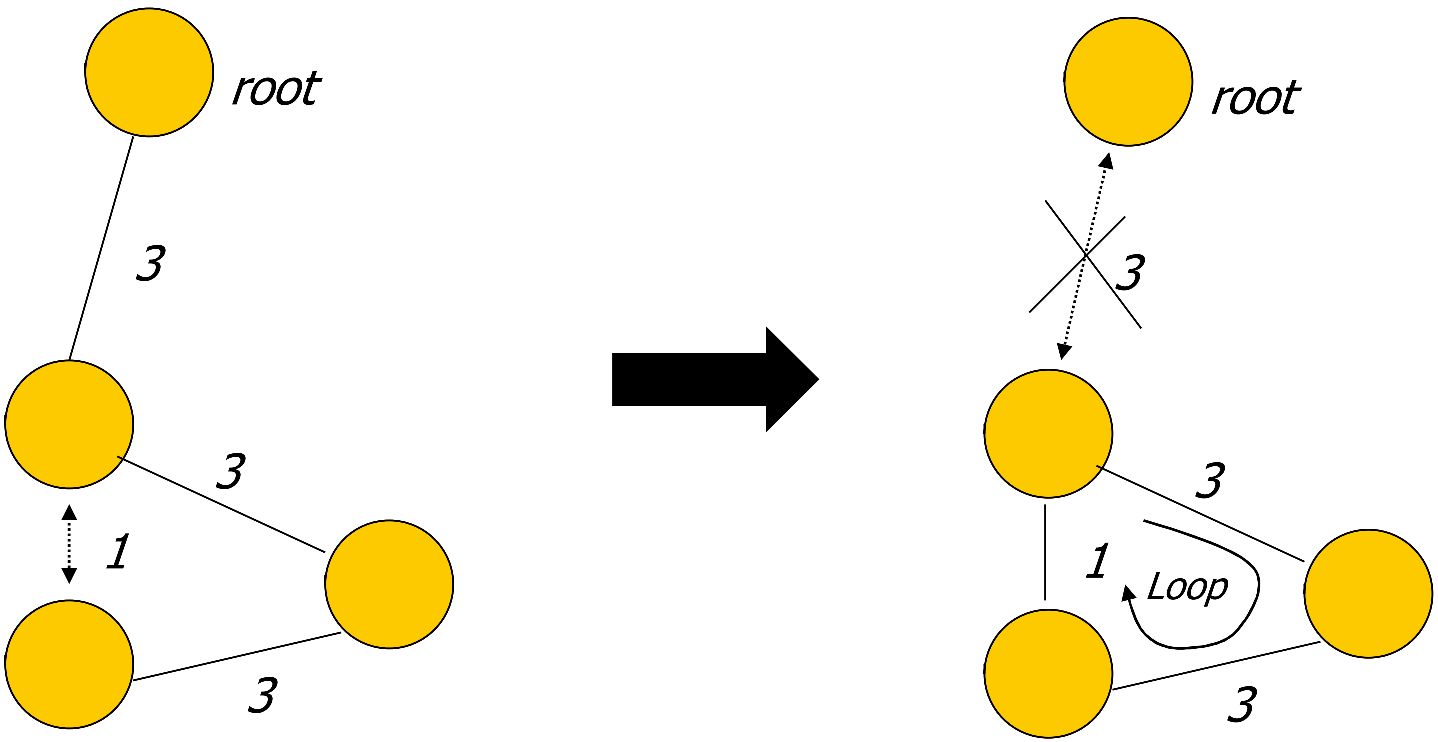
# Changing Parents in the Group

- Changing the parent nodes is called “switching”
- Example: node A switches to node C



# Switching Parents

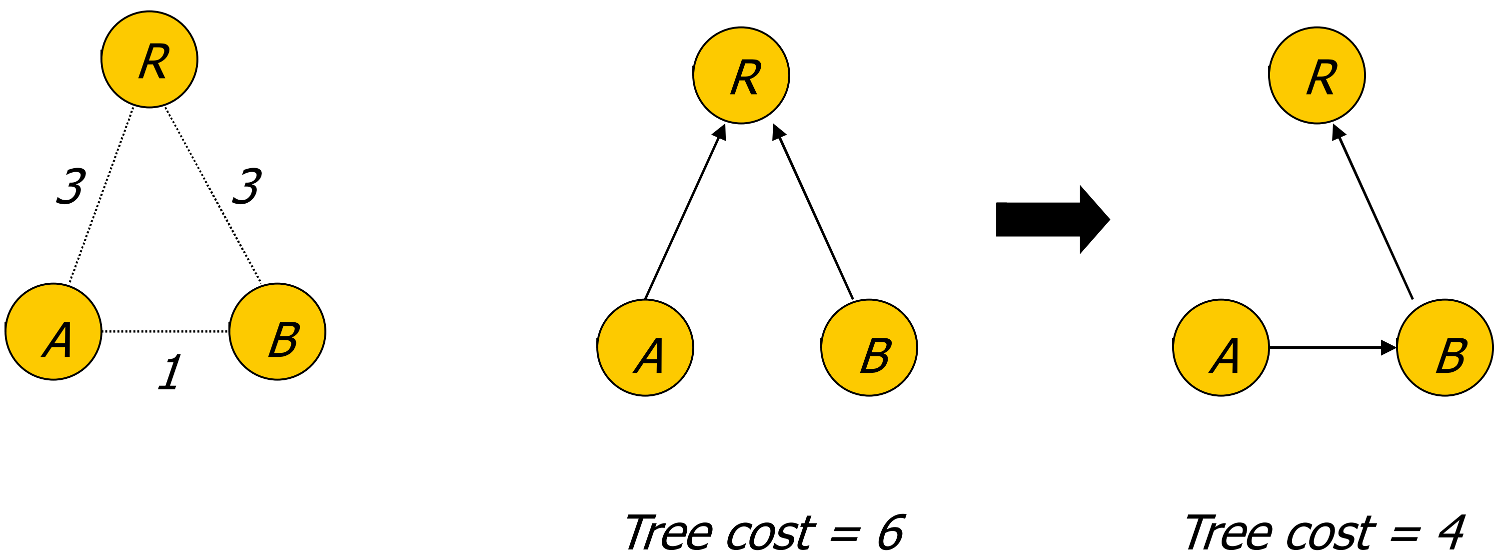
- Nodes can switch parents to optimize the tree (or on bandwidth depletion of parent)
- Switching to arbitrary nodes could lead to loops…




→ Nodes can only switch to the ***root*** or a ***sibling***, since they can not be successors and hence loops cannot occur

# Optimizing the tree

- *Why* would a node switch parents?
- So far the tree evolves purely by lack of bandwidth
  - Connect to node (root)
  - If bandwidth depleted, child switches down
- Additional switches of parents to optimize the tree for low cost.



# Some Rules to Avoid Chaos

- Parents send information about siblings regularly
- Siblings ping each other to determine distance

- To switch, a swichting request is sent

- Alternative parent only accepts if it‘s not switching itself at the same time (always reject while switching yourself…)
- Switching request includes current parent information

# BTP – A Summary

- BTP is first ***tree-first*** ALM approach
- Optimization by local topology adaptation
- Able to achieve (rather) low cost trees
- Rather efficient (not a lot of signalling overhead, pings are costly)

- Some issues
  - ***Which topology change is possible?***
  - ***How does this effect the paths?***
  - ***How is the expected performance in high-churn scenarios?***
  - ***What happens if peers are malicious?***

# DONet - Introduction

- DONet is main core (PPLive, Coolstreaming, etc.)
- “Data-driven“ streaming
  - Aim: Use availability of data rather than explicit topology to guide data flow

→ *Peer-assisted streaming* (main traffic provided by servers)

→ *Pull-based*

  - Bootstrap into overlay using central point (“tracker”, monitors system)
  - Periodically exchange data availability with random partners and retrieve new data
  - Load Balancing is main problem due to real time constraints

***„CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming”***

# DONet - Components

- **Membership Manager**
  - Maintain information
    - A list of members in the group
  - Update information
    - Periodically generate membership message
    - Distribute it using Scalable Gossip Membership Protocol (SCAMP)

- **Partnership Manager**
  - Partners are members that have desired data segments
  - Exchange Buffer Map (BM) with partners
  - Buffer Map contains information of availability of segment

- **Load Balancing** („*Scheduling*“ in DONet terms)
  - Determine which segment should be obtained from which „partner“
  - Get and provide segments from/to partners

# DONet - Implementation

- Each node has unique ID and a membership cache ("*mcache*")

*1* New nodes contact server, get randomly selected *deputy nodes*

*2* Get partner candidates from deputy node's mCache

- Membership messages are gossiped among nodes (fall back info)

- Videos are divided into segments of uniform size
- Available segments represented in the Buffer Map (BM)
  - BM usually contains 120 bits for 120 segments
- Local exchanges of BM to disseminate availability information

# Load Balancing in DONet

- *Aim:* Adaptation to network dynamics

- Each segment has playback deadline
  - Minimize number of missing segments at deadlines
  - Consider heterogeneous bandwidth of peers

- Variation of the Parallel Machine Scheduling problem (pms)
  - NP-hard problem
  - Increasing dynamics worsen the problem…
  - DONet introduces simple heuristic

- Number/properties of potential predecessors for each segment is estimated
  - Window-based buffer maps are exchanged…

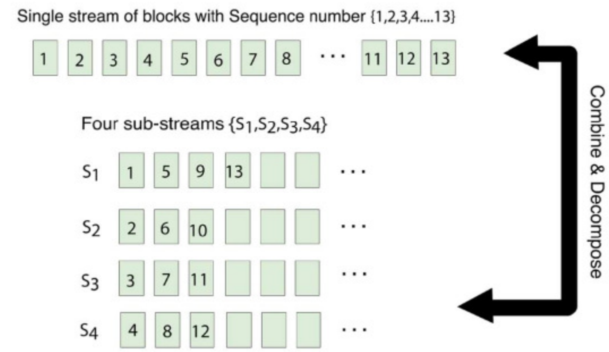
- DONet heuristic:
  - Chose suppliers of rare segments first
  - Multiple suppliers: highest bandwidth within deadline first
  - ***(rarest first / earliest deadline-best provider second… it's almost like BitTorrent ;-) )***

# DONet - Failure Recovery

- Graceful departure
  - Issue a departure message when departing
- Node failure
  - Partner detecting failure will issue substitute departure message
- Departure messages again disseminated using gossiping
- Nodes periodically establish connection to other nodes in mCache
  - Connect to nodes with high segment send/receive throughput

# Coolstreaming

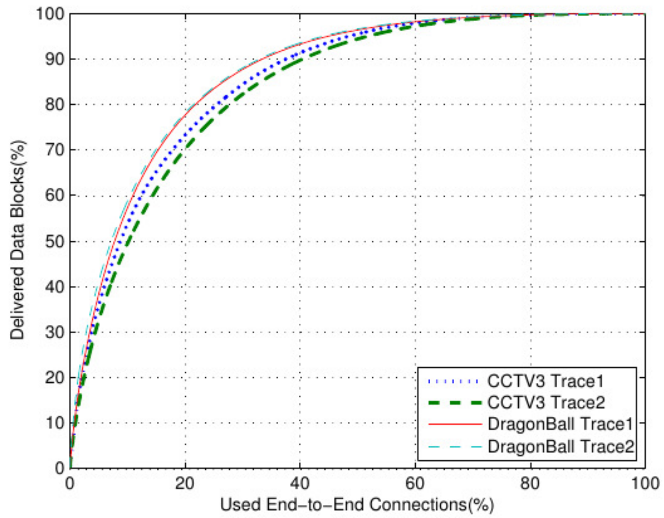
- Basic components
  - Similar to DONet:
    - Membership management
    - Partnership management
    - Stream manager: Data-driven (with buffer map exchange)
  - Different: Substreams
- Content delivery
  - Pushing substreams after receiving a subscription request
  - Parent nodes do not dropt child nodes
  - Child nodes have to monitor in-coming connections for parent reselection
- Peer adaptation
  - By monitoring in-coming throughput
  - By comparing buffer status of parent and other partners




*„Coolstreaming: Design, theory, and practice"*

# mTreebone

- Purpose:
  - to leverage advantages of tree and mesh approaches



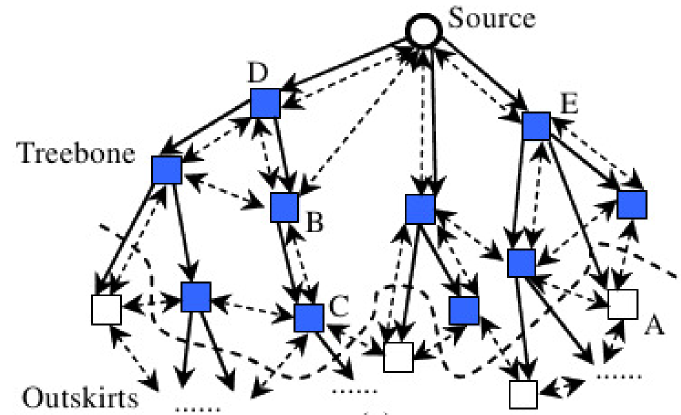

- Novelty:
  - Identifying and using stable nodes
  - Explicit tree on top of mesh overlay
- Motivation: PPLive trace study
  - 80% of data was delivered by 20% of connections
  - Nodes of those connections are “stable”
  - The delivery path of single segment: tree
  - There is a small set of representative trees

***http://www.cs.sfu.ca/~jcliu/Papers/mTreebone-icdcs2007.pdf***

# mTreebone - Architecture

- Stable tree-based backbone
  - Treebone consists of only stable nodes
  - Non-stable nodes: attached the tree as out-skirt
  - Most of streaming data: pushed through the treebone

- An adaptive auxiliary mesh overlay
  - Consists of all nodes
  - Each node keeps a partial list of active nodes
    - Mesh neighbors
    - At least one dedicated treebone parent
  - Exchange Buffer Maps but not request



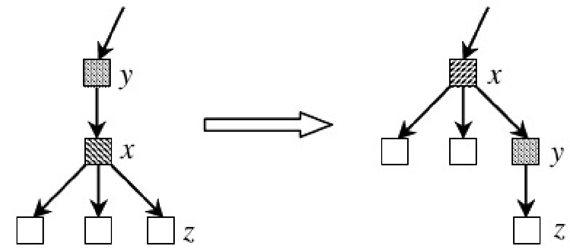
(a)

# mTreebone – Construction

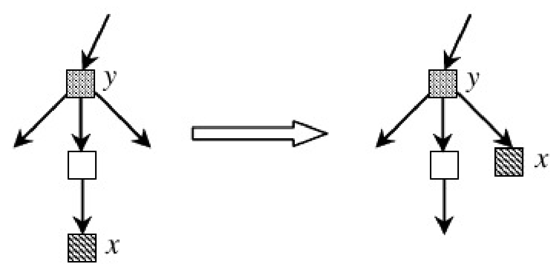
- Optimal stable node promotion
  - Stability of nodes is proportional to duration in the overlay
  - Definition of node's age in the session: Time elapsed since arrival
  - Age threshold is critical: 30% residual session time
- Bootstrap
  - New node obtains from source:
    - session length
    - arrival time
    - a partial list (at least one treebone node)
  - New node: attaches to treebone + connects to mesh neighbors
  - Periodically checks node's own age
  - If age exceeds the threshold → promote itself as a treebone node
  - Randomize promotion (why?)

# mTreebone - Optimization

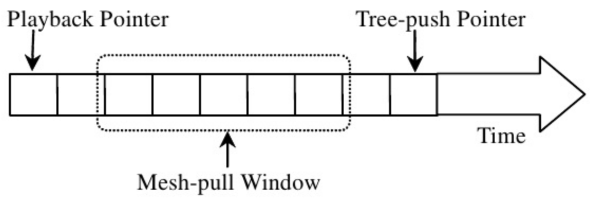
- High degree preemption
  - Each treebone node (X) periodically checks if there is a node Y whose:
    - Has less the number of child nodes
    - Is closer to the source
  - X will preempt Y
  - Y will rejoin the treebone
  - Where to look for Y? (parent of X, or a node in X's partial list)




- Low delay jump
  - Each node (X) periodically checks the delay of other treebone nodes
    - If a treebone node (Y) has
      - Less delay than the parent node of X
      - Enough bandwidth
    - Jump: X will
      - Leave current parent node
      - Attach itself to node Y



# mTreebone – Collaborative Push/Pull

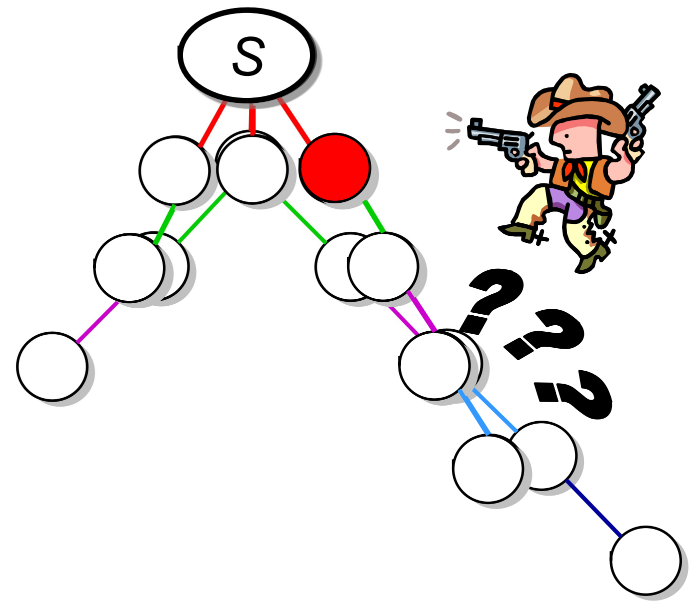



- Push/Pull switching
  - Idea:
    - Data mainly pushed (with tree-push pointer)
    - Data pulling when segments are missing (mesh-pull window)
- Dealing with node churn
  - Graceful leave
    - Informs mesh neighbors and treebone child nodes
  - Failure is detected by
    - Missing control messages (e.g., buffer maps)
    - Persistent loss

# The Importance of Resilience

- *Locality of end systems not known*
- *End systems are less reliable and more easily attacked and destroyed than dedicated servers*






*Resulting Questions:*

- *How can neighbors be selected in order to create a **resilient** ALM?*
- *Which regions of the ALM overlay are sensitive? How can they be protected?*
- *How can the privacy of users be protected (against whom?)*
- *Plus: how do we create a cost efficient overlay with low latency, anyways?*

*Th. Strufe: Ein Peer-to-Peer-basierter Ansatz zur Live-Übertragung multimedialer Datenströme, Dissertation*

# Resilience

- Errors of two different types:
  - Transmission errors (erroneous / incomplete packets)
  - Structural errors (link or node breakdowns)

- Tackling transmission errors is easy! ;-)
  - ARQ
  - FEC

- Automatic Repeat Request / Selective Repeat Request, etc (*given*)
- Forward Error Correction (increase redundancy)
  - Block Codes / Convolutional Codes
  - LDPC vs Reed Solomon vs Turbo Codes (etc., etc.)
  - Interesting: Scalable Video Coding and Multiple Description Coding

*Strufe et al.: Methods for Improving Resilience in Communication Networks and P2P Overlays. Praxis der Informationsverarbeitung und Kommunikation (PIK), 2009*

# Structural Errors

- Two ways to tackle:
  - Restoration (you will always need this part ;-)
  - Protection

- In order to restore the structure you need to
  - Detect the failure
  - Locate/isolate the failure
  - Notify concerned nodes
  - Restore the structure (potentially find new sources, re-route)

  - But: ***how do you detect errors!? this may take time!***
  - ***Repairs are costly (additional messaging, restructuring)***
  - ***Permanent restoration (high churn, low reliability) too expensive!***

# Protection for Structural Errors

- Might be better to protect structure (topology) in the first place?

- Idea: slightly increase redundancy
  - Of state information
  - Of hardware…

- In infrastructure networks:
  - Alternative global routing (know a second e2e path!)
  - Alternative local routing (know a detour around pot holes)
  - Keep bypass topologies (maintain a fall back structure)

- ***Do you know examples? :-)***

# Protection in P2P Streaming

- What are the characteristics in P2P streaming?
  - Node depends on ***all other nodes*** on path to source
  - Nodes are highly unreliable
  - Link characteristics vary drastically

- So what can we do?
  - Detect error early (detection and repair is costly, takes time)
  - Missed one packet? It‘s video, not such a big deal… Try to get most of them…
  - Spread dependence on multiple paths (plus Multiple Description/Layered C.)!
  - ***AND:*** decrease dependency (number of pre-/successors)…

# Resolving the Unrealibility: SplitStream

- Microsoft Research / Rice University
- Castro, Druschel, Kermarrec, Rowstron, Singh (2003)
- …same as Pastry, PAST, Scribe…

- Aims
  - Tree-based multicast
  - Fair load balancing
  - (later: robustness against failing nodes (with multiple description encoding))

- Main approach
  - Create multiple multicast trees
  - Keep sets of inner (forwarding) nodes of all trees disjoint
  - ***Simple!*** Create different Scribe groups, only nodes with matching prefix relay

    ***(why are they disjoint?)***

# SplitStream – Basic Concepts

- Stream is split into multiple partial streams (a.k.a. „Stripes“)
- Aim at balancing the bandwidth of each stripe
- Aim at balancing information of all stripes

- Consequences:
  - No hierarchy between stripes
  - Some information can be presented, even if one (some) stripe(s) goes missing

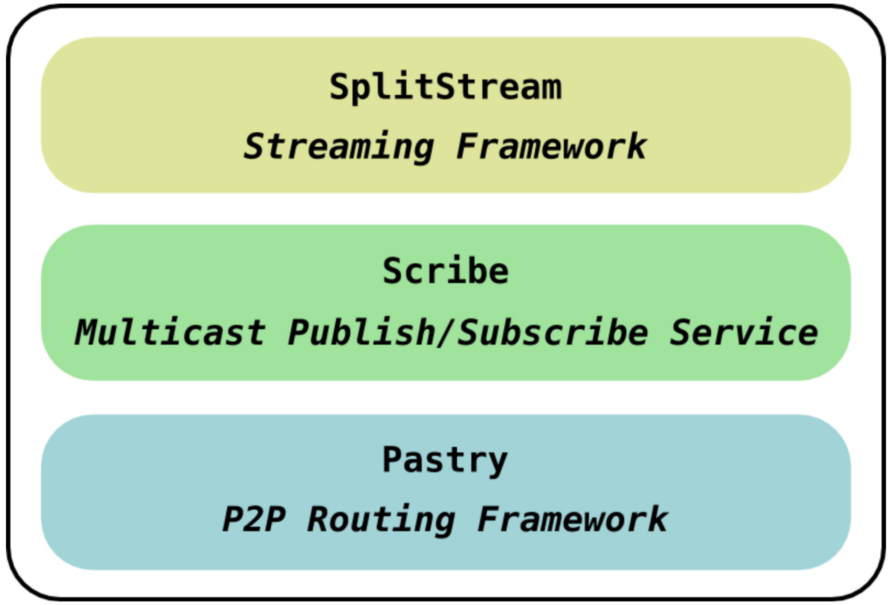
- Create a single multicast tree for each stripe
- Results in a multicast „forest of trees“ for each stripe (it‘s a tree, actually)

- Allow nodes to forward in only one stripe (exceptions occur due to bandwidth limitations in the real world)

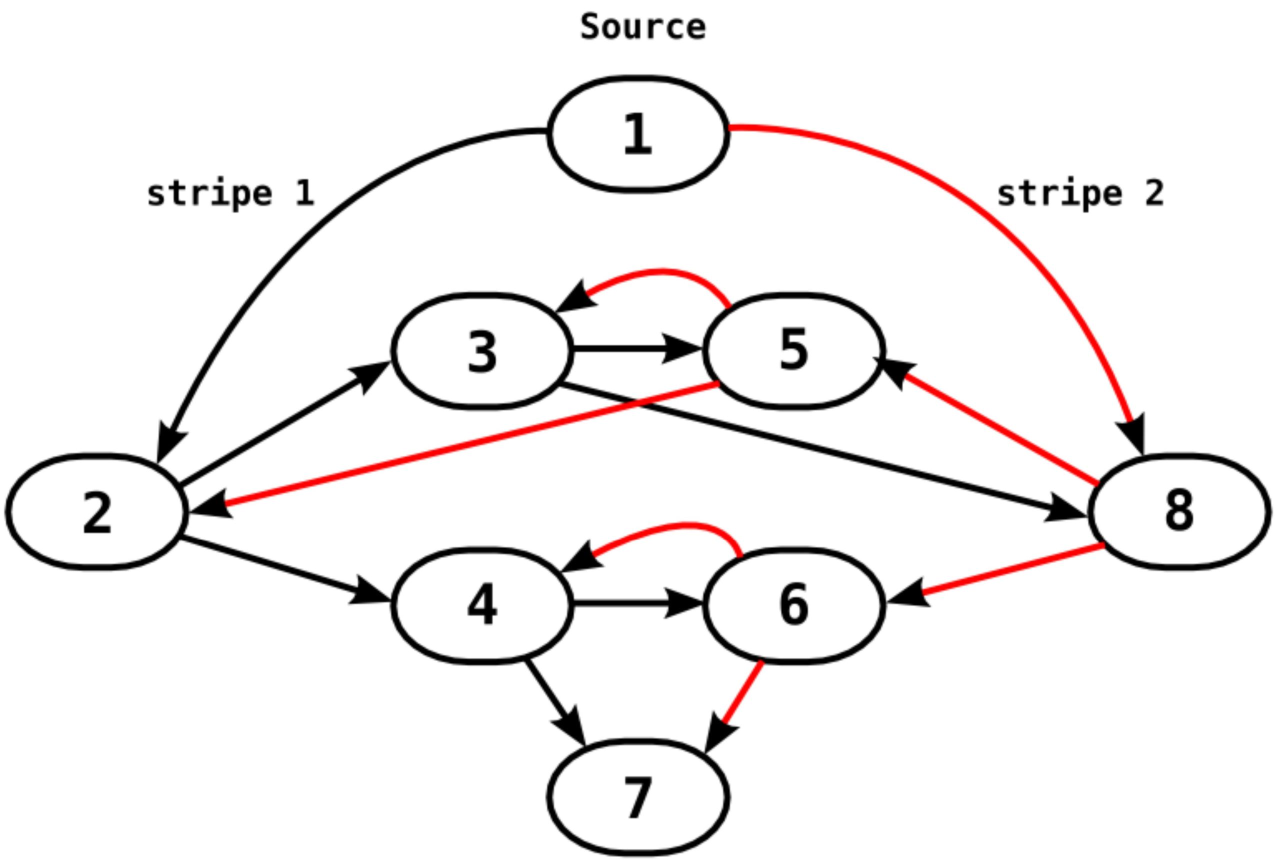
# SplitStream – Architecture

- SplitStream uses Scribe uses Pastry:

- SplitStream
  - stripeId := groupId
- Scribe
  - groupId := messageKey
- Pastry
  - Routing by <nodeId, key> pairs



# SplitStream – Multicast Tree



# SplitStream – Summary

- SplitStream achieves fair load balancing
- Robust to departure of nodes
- Using location aware Pastry the stretch is fairly low (Stretch of 2 in the best case)

- Issues of SplitStream
  - Heterogeneity of nodes is a problem: „spare capacity group“ needed in case nodes with matching prefix run out of bandwidth
  - SCG leads to *very* unbalanced and fragile topologies with realistic churn / user models
  - Trees potentially ***very high*** hence…
    - Each tree is not resilient to failures
    - Delay penalty (jitter!) high
  - Nodes can gather any knowledge on the topology (***what about attacks?***)

# Achieving DoS Resilience

- All systems so far aim at ***efficiency***, ***scalability***, and ***robustness***

- Is it possible to avoid censorship?
- What happens in commercial scenarios?

- ***…what about resilience?***

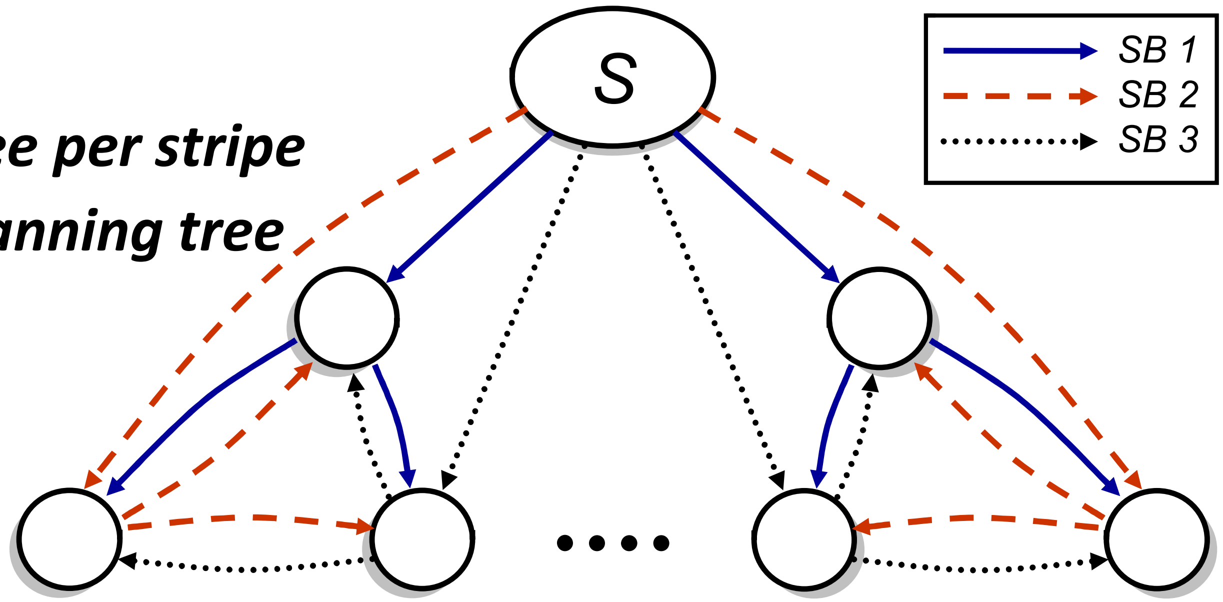
- An ALM is resilient (towards DoS) if:
  - It is not easy for an attacker to identify important nodes
  - An attack (any attack) does not lead to significant damage

# Optimal Stable Topology

- Minimize damage of each single node failure
- Minimize dependency (predecessors of nodes)
- Balance the relevance of all nodes

*Construction:*

- ***Stripe the stream***
- ***Use one spanning tree per stripe***
- ***Max. 2 layers per spanning tree***
- ***Balance successors***




- Don't disclose *any* information that's not absolutely necessary
- → The evolving topology is optimally resilient towards DoS

*Brinkmeier, Strufe, Schäfer: Optimally DoS resistant P2P Topologies for Live Multimedia Streaming. Transactions on Parallel and Distributed Computing*

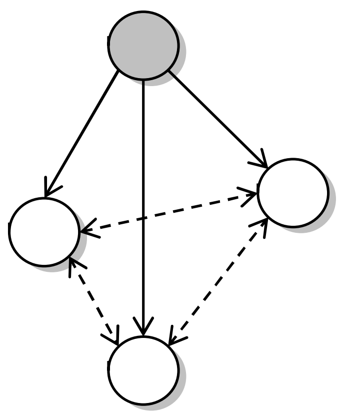
# Design Decisions

- Locality aware DHT for ***stream*** location with short links
- Redundant registration of streams to achieve short location paths

- Tree-first tree-based overlay live streaming
- Decrease dependency: ***stripe the stream***
- Use ***local*** information only (don't even provide means to gather any information about anything but your neighbors)
- Cost-based local optimization of the topology:
  - Minimize link distances
  - Minimize number of successors of each node
  - Balance the successors over all child nodes
  - Maximize vertex connectivity in the overlay

# Cost-based Local Topology-Optimization

- Only parents optimize local situation
- Two possible operations:
  - Move down (forward child to other child as alternative parent)
  - Move up (request successors)



- *Optimize in three steps:*
  1. *Select edge with highest cost*
  2. *Select best alternative parent*
  3. *Calculate gain and execute if gain > threshold*
- *Bandwidth available? Request successors!*

- Combined optimization: $K = s * K_{stability} + (1 - s) * K_{efficiency}$
- Adjust the optimization using weight (***s***)

# Resiliency Cost Metrics

- ***Requirement***: Low dependencies!
  - Low topologies
  - Balanced topologies (no single failure leads to high damage)
  - Disjoint spanning trees for different stripes

    → ***Each node only relays data in the spanning tree of one stripe***

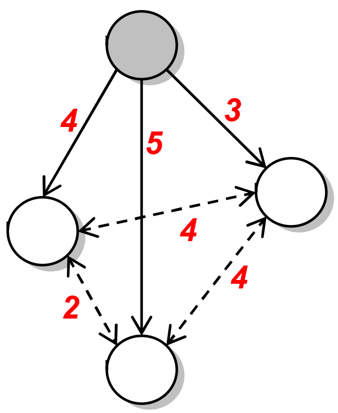
- ***Cost functions*** (Node v, Child w, Stripe i):
  - $K_{forw}(w,i)$: Place nodes which relay packets in the spanning tree close to the source
  - $K_{bal}(v,w,i)$: Balance underlying branches
  - $K_{dep}(v,w)$: Direct dependency of the child node
  - $K_{sel}(i)$: Selected edge is in ***unwanted*** spanning tree

- $K_{stability}(v,w,i) = K_{forw}(w,i) + K_{bal}(v,w,i) + K_{dep}(v,w) + K_{sel}(i)$

# Efficiency Cost Metric

- ***Efficiency requirement***:
  - Low sum of distances (hence the topology is network efficient)

→ Minimize global distances by minimizing local distances




- *Calculate the maximum possible local decrease in distances in the spanning tree*
- *Cost function:*

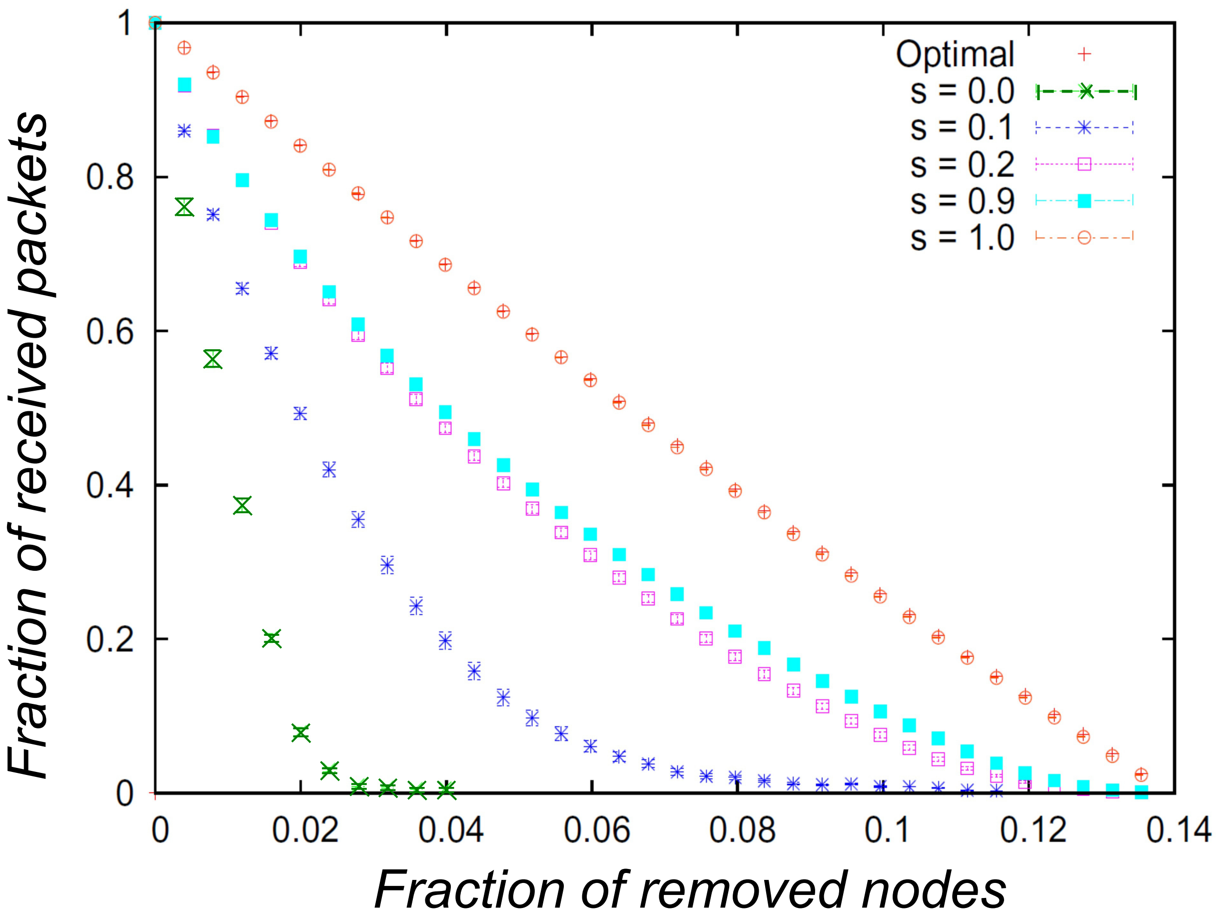
$$K_{efficiency}(v,w,C_i(v)): \quad \max\left\{ 1 - \frac{d(v,w)+d(w,u)}{d(v,w)+d(v,u)}, \; u \in C_i(v) \setminus \{w\} \right\}$$

- Distance estimation ***d(e)*** using the synthetic coordinate system

# Evaluating the Topology Control

- ***Questions***:
  - *Do we create resilient topologies with a high weight ‚**s**‘?* (Resilience)
  - *Are they efficient, if we use a low **s**?* (Efficiency)
  - *Are resilience and efficiency of the topologies adjustable?* (Trade-Off)
- ***Simulation setup:***
  - Backbone with 750 routers
  - 1 source (1 stream)
  - 5 Stripes
  - {50,250} End systems (user model from literature of Veloso et al.)
  - Weight ‚s‘ [0,1.0]
- ***Metrics***
  - Resilience
  - hop-penalty (compare tree to minimum spanning tree)

# Resilience of the Topologies



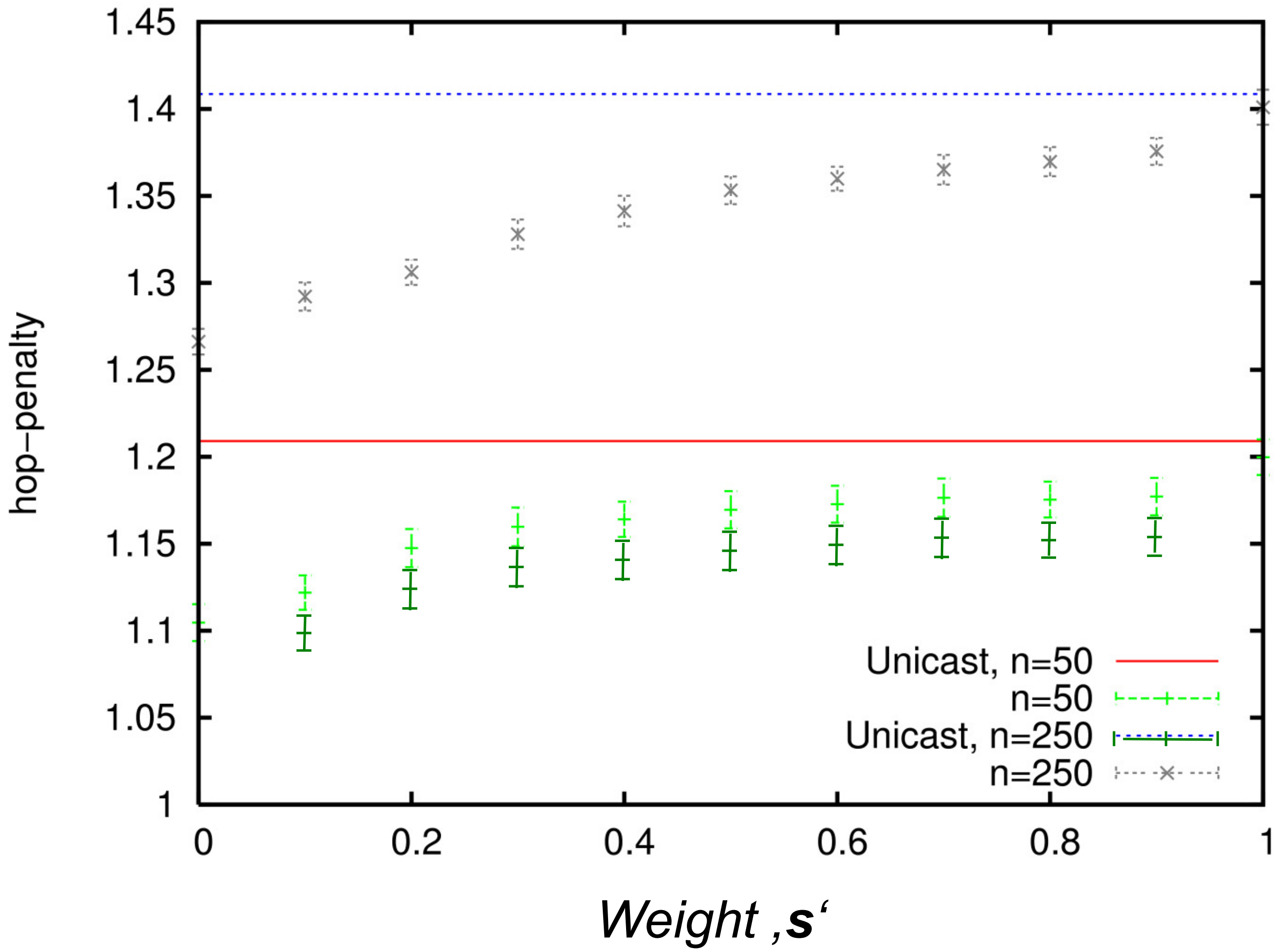

*Low resilience when using a low weight ‚s'*

*Resilience rises with increasing weight s*

*Resilience very similar with s Є [0.2,0.9]*

*With s=1.0 resiliency is almost optimal*

# Efficiency of the Topologies




*Very high efficiency when using a low weight s*

*With increasing weight s the efficiency drops*

*With any weight we transmit less packets than in a unicast scenario*

# Summary of ALM

- Video represents lion share of traffic today
- Video puts high burden on servers and networks
- Network multicast not available → P2P streaming *makes sense!*

- Terminology (*and its pitfalls… ;)* )
  - Multicast vs. „Broadcasting“ (1:n vs. n:m)
  - Pull vs. Push-based streaming
  - Tree-based vs. Mesh-based (Tree- vs. Mesh ***first***)

- Selected systems (Narada, BTP, …)
- Resilience as a special problem (SplitStream)

# Thanks for your attention! ☺

*(and remember: we‘re always looking for good students for BSc./MSc. theses and as student helpers („Hiwis“)… ;-) )*