Cooperative End-to-end content distribution

Márk Jelasity

Content distribution

- So far we looked at search
- Content distribution is about allowing clients (peers) to actually get a file or other data after it has been located
- Different types of content require different techniques
  - Downloading huge files (dvd-s, linux distributions, etc)
  - Streaming media

Content distribution networks

- System organization
  - Centralized
    - **Server farms behind single domain name, load balancing**
  - Dedicated CDN
    - **CDN is an independent system for typically many providers, that clients only download from (use it as a service); typically http**
  - End-to-end (p2p)
    - **special client is needed and clients self-organize to form the system themselves (as usual in p2p)**

Outline

- Large file distribution
  - Dedicated CDN
    - **Akamai: privately owned CDN**
    - **CoralCDN: similar idea to Akamai, only cooperative p2p technology is used**
  - End-to-end p2p CDN
    - **Bittorrent**
    - **The network coding approach**
- Media streaming
  - SplitStream, bullet

Akamai

- Provider (eg CNN, BBC, etc) allows Akamai to handle a subset of its domains (authoritive DNS)
- Http requests for these domains are redirected to nearby proxies using DNS
  - Akamai DNS servers use extensive monitoring info to specify best proxy: adaptive to actual load, outages, etc
- Currently 20,000+ servers worldwide, claimed 10-20% of overall Internet traffic is Akamai
- Wide area of services based on this architecture
  - availability, load balancing, web based applications, etc

CoralCDN: motivation

- Commercial CDN-s are good but expensive: small sites with low bandwidth can't afford them
- Small sites are more vulnerable to flash crowds and any fluctuation of traffic in general
- P2P filesharing has shown willingness to provide bandwidth for popular content
- Let's build a P2P CDN to support (popular but) small, underprovisioned websites
- [motivation is shaky, but interesting technology anyway]

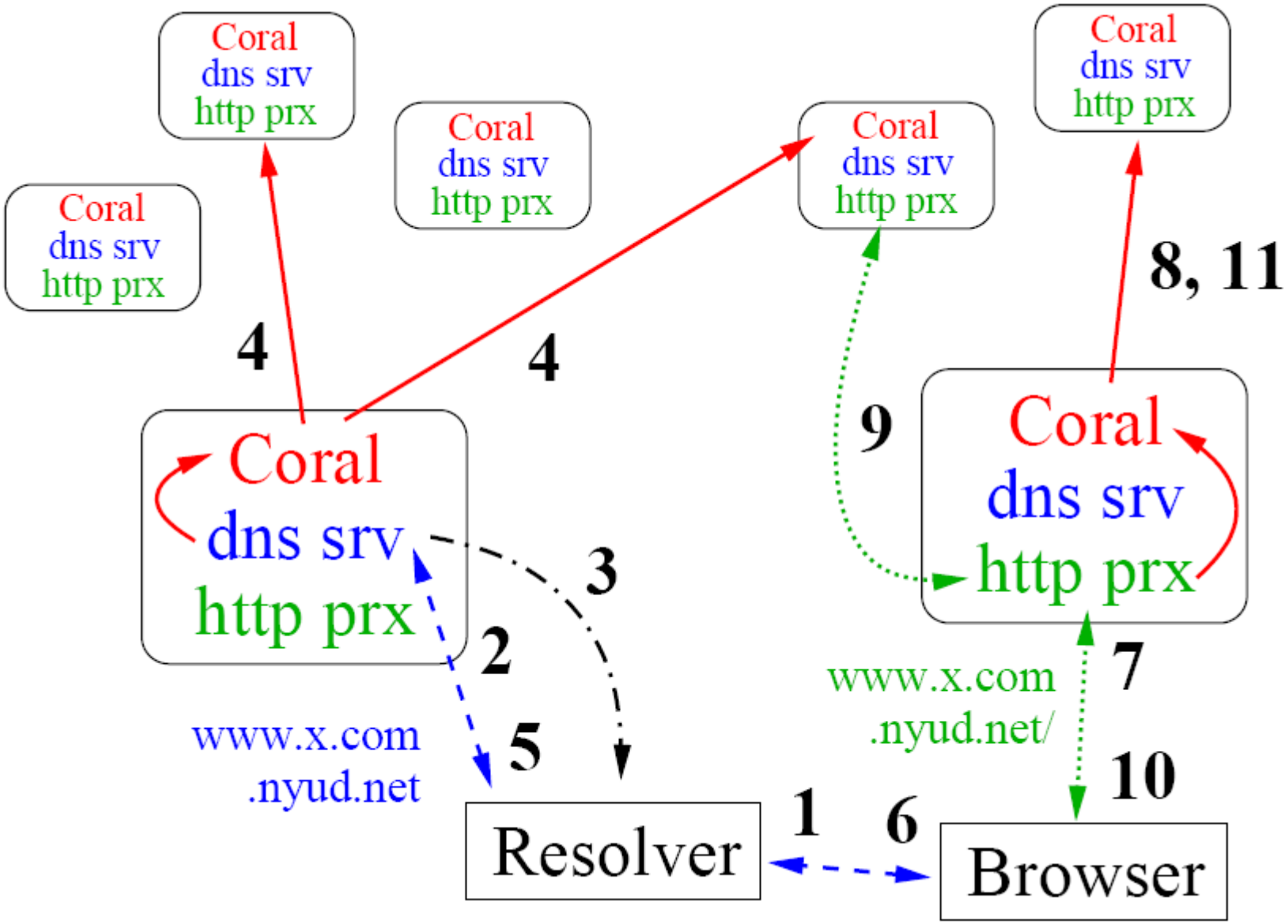
CoralCDN

- Participating peers form
  - an indexing infrastucture (DHT-like)
  - a network of HTTP proxies
  - a network of DNS servers
- How to use the system?
  - Publishers can “coralize” urls by appending the domain “.nyud.net:8090” to the name of the server, eg http://www.inf.u-szeged.hu.nyud.net:8090/
  - In email, usenet, etc messages any url-s can be coralized the same way (thereby preventing the “slashdotting” of the site in question)

How it works

- Clients tries to resolve www.inf.u-szeged.hu.nyud.net
- Coral DNS server probes the client for RTT and looks for coral DNS and HTTP servers nearby
- Coral DNS returns DNS and HTTP servers for www.inf.u-szeged.hu.nyud.net
- Clients send HTTP request to http://www.inf.u-szeged.hu.nyud.net:8090/
- If given coral serever has the page, sends it.
- Otherwise looks up the URL in Coral, and if it is available, caches it from within Coral, and sends it
- Otherwise it fetches the page from original location http://www.inf.u-szeged.hu/
- The coral server notifies the system that it now caches the URL

Overview of CoralCDN



Distributed sloppy hash table

- Sloppy: keys can be stored not only on nodes that are the closest, but also in nodes that are close enough: better load balancing
- Inserting a key
  - Approach the reposnsible node through routing as in DHT, but stop sooner, if nodes that are close are “full” and “loaded” (load balancing technique)
- Retrieval
  - Approach the responsible node for the key, and stop when finding the first node storing the key

Clustering

- Many DSHT-s in parallel: hierarchical clustering
  - 3 levels: according to RTT among cluster members
  - All nodes have same ID in all clusters, level 0 cluster covers full network
- Implementation of clustering
  - Storing hints in the DSHT: key: IP address of router and subnet prefix: value: node
    - **Joining nodes quickly find other nodes in the same subnets**
  - Collect RTT info in all contacts: if other cluster seems closer, change cluster

Exploiting clustering

- Retrieval is biased towards lower levels, so nearby HTTP and DNS servers can be located
  - Start routing protocol at level 2 (closest nodes)
  - If no key found, go to level 1 and simply continue the routing (the nodes level 2 cluster is subset of its level 1 cluster)
  - Go until reaching level 0
- Clusters do not increase lookup time (roughly the same as a simple routing at level 0)

Some notes

- CoralCDN is deployed on PlanetLab
  - 750 nodes
  - 1,500 Gbytes for 700,000 IP addresses a day
- It is only a proof of concept, but wide scale deployment is a question
  - If it is single administrative domain, why not more control, why the p2p approach?
  - If it is voluntary, multiple admin domain, who would want to join voluntarily without restricting content? What kind of content? Etc
  - DNS and other overhead makes it rather slow

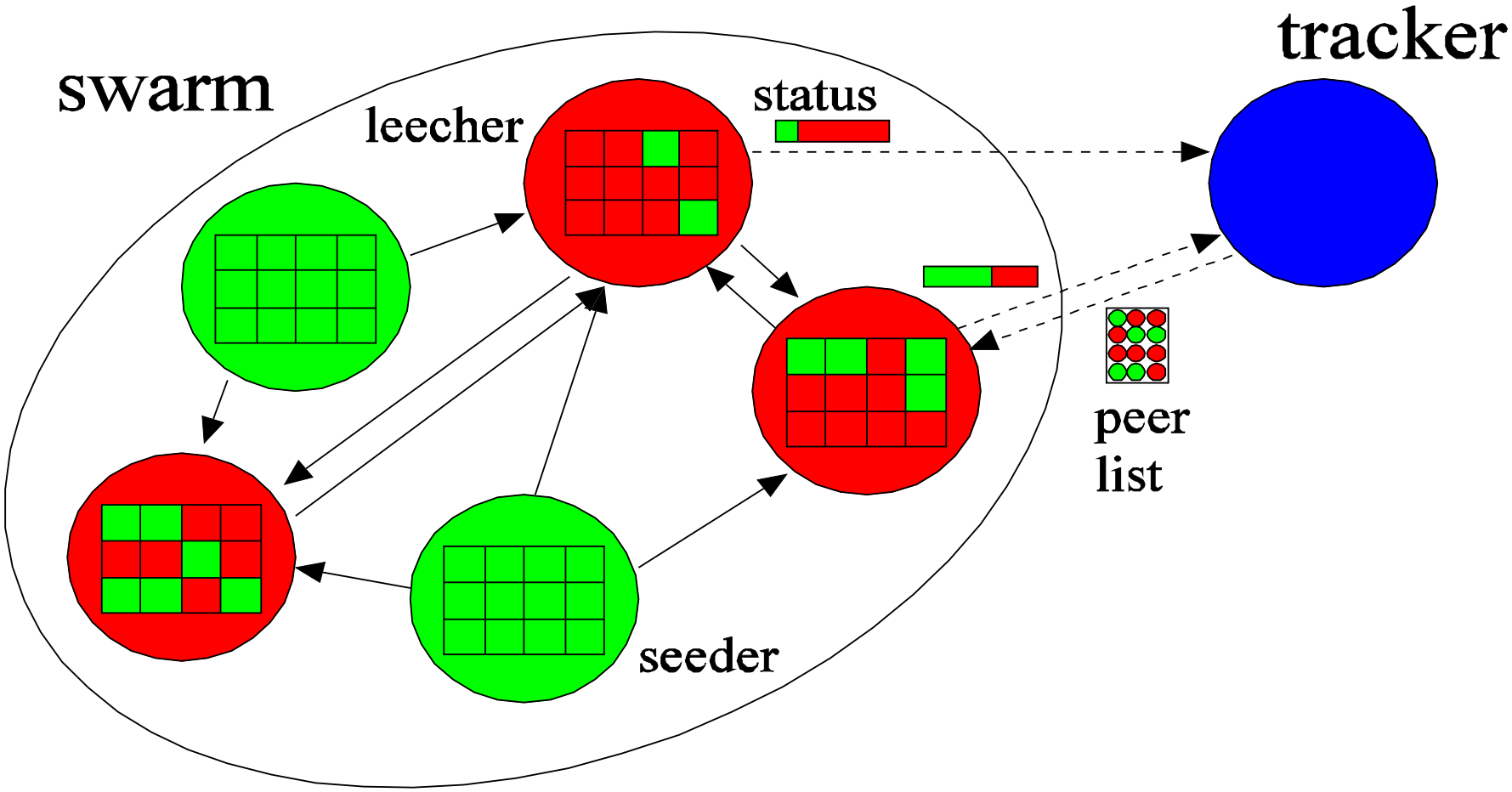
End-to-end P2P CDN: BitTorrent

- Invented by Bram Cohen
- Currently 20-50% of Internet traffic is BitTorrent
- Special client software is needed
- Basic idea
  - Clients that download a file at the same time help each other (ie, also upload chunks to each other)
  - BitTorrent clients form a swarm: a random overlay network

BitTorrent

- Publishing a file
  - Put a ".torrent" file on the web: it contains the address of the tracker, and information about the published file: eg chunk hashes (256M chunks)
  - Start a tracker, a server that
    - **Gives joining downloaders random peers to download from and to**
    - **Collects statistics about the swarm**
  - [Note that there are "trackerless" implementations already]
- Download a file
  - Install a bittorrent client and click on a ".torrent" file

BitTorrent overview



BitTorrent client

- Client first asks 50 random peers from tracker
  - Also learns about what chunks they have
- Picks a chunk and tries to download its pieces (16K) from the neighbors that have them
  - Download does not work if neighbor is disconnected or denies download (choking)
- Allows only 4 neighbors to download (unchoked neighbors)
  - Periodically (30s) does optimistic unchoke: allows download to random peer (important for bootstrapping and optimization (exploration))
  - Otherwise unchokes peer that allows the most download (each 10s)

tit-for-tat

- Tit-for-tat in iterated prisoners dilemma
  - Cooperate first, then do what the opponent did in the previous game
  - Very good strategy (Axelrod)
- BitTorrent is a kind of tit-for-tat
  - We unchoke peers (allow them to download) that allowed us to download from them
  - Optimistic unchoking is the initial cooperation step to bootstrap the thing
- How about hacked clients? Why don't they spread and kill BitTorrent?

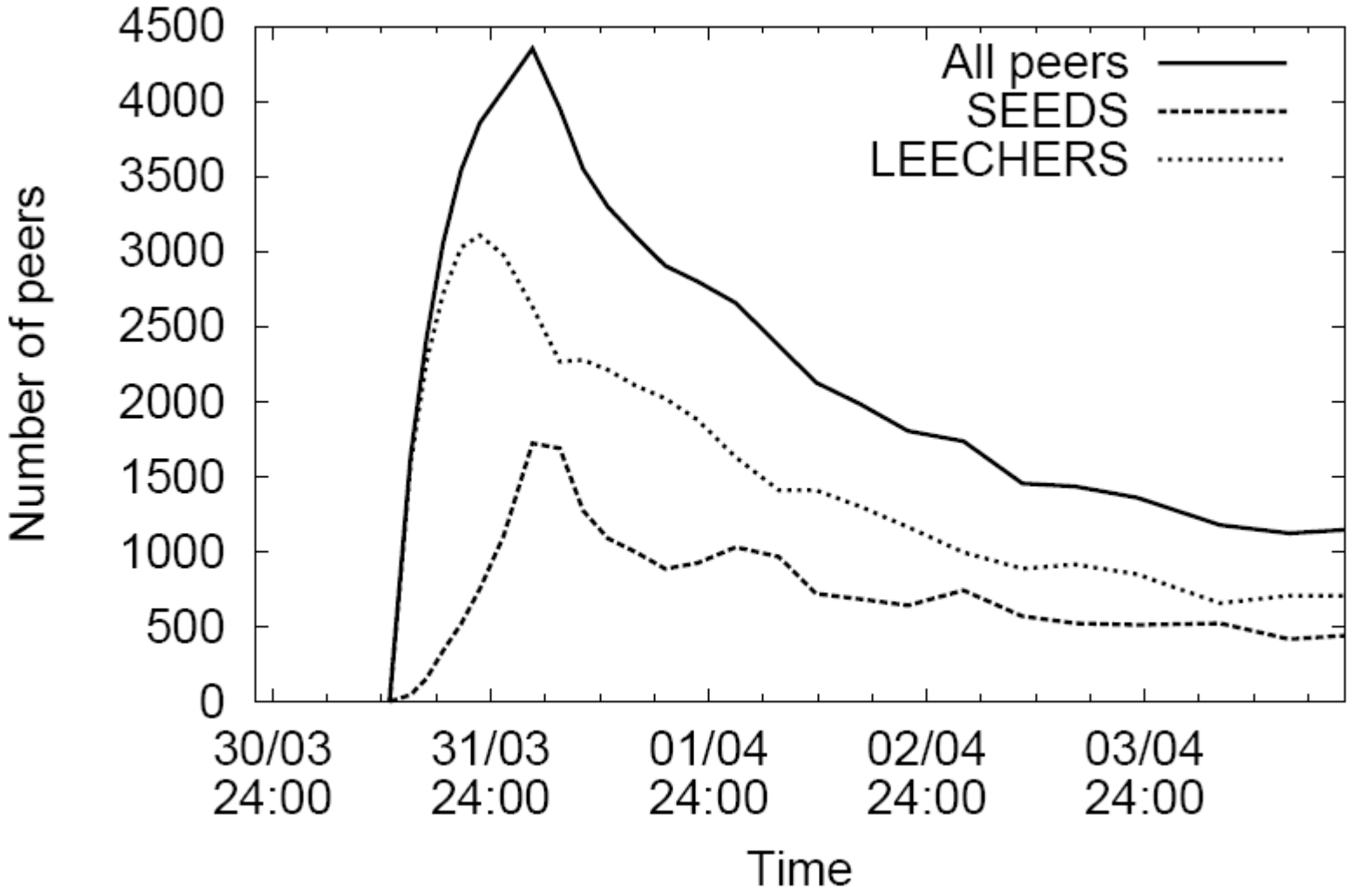
Chunk selection

- Another very important question is what chunk to select to download?
- Clients select the chunk that is rarest among the neighbors (local decision)
  - Keeps all chunks equally represented
  - This is good because no chunks get lost, and it is likely that peers find chunks they don't have
- Exception is first chunk
  - Select a random one (to make it fast: many neighbors must have it)

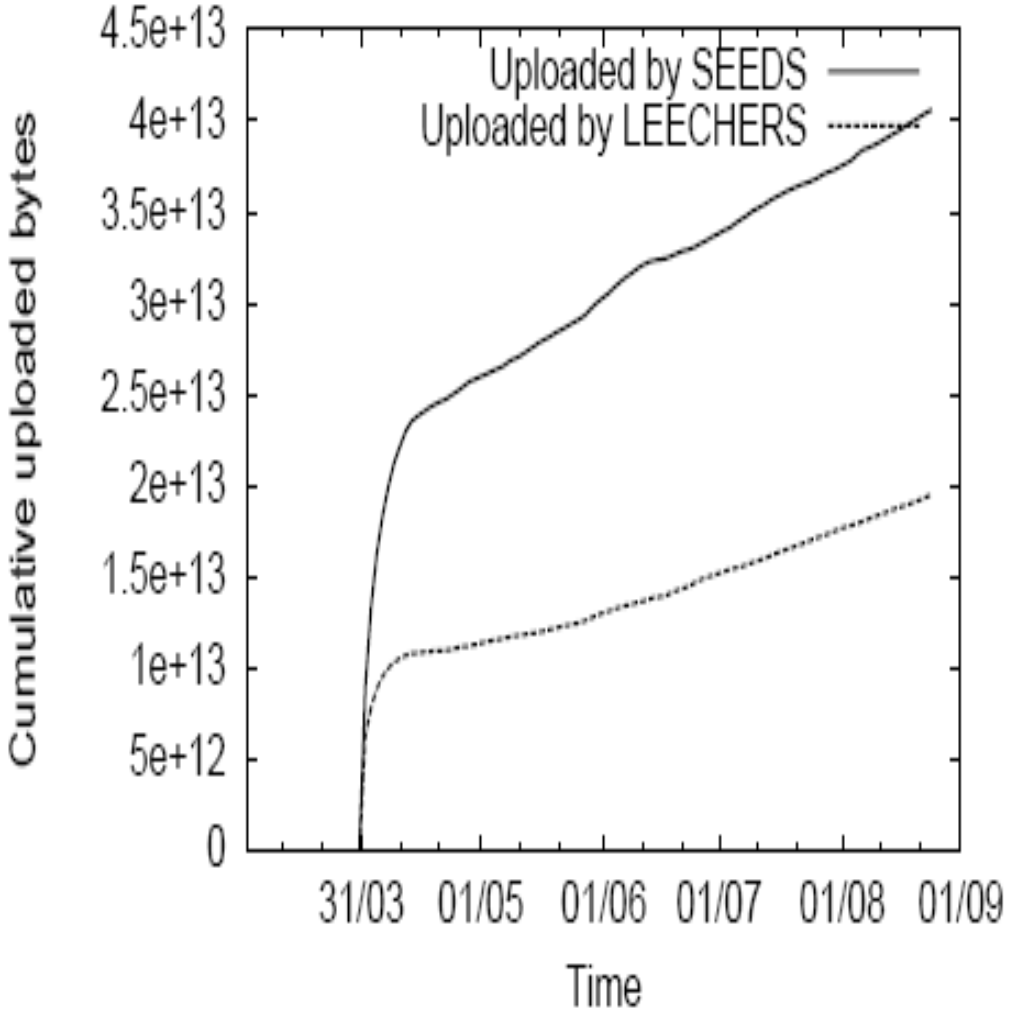
Measurements

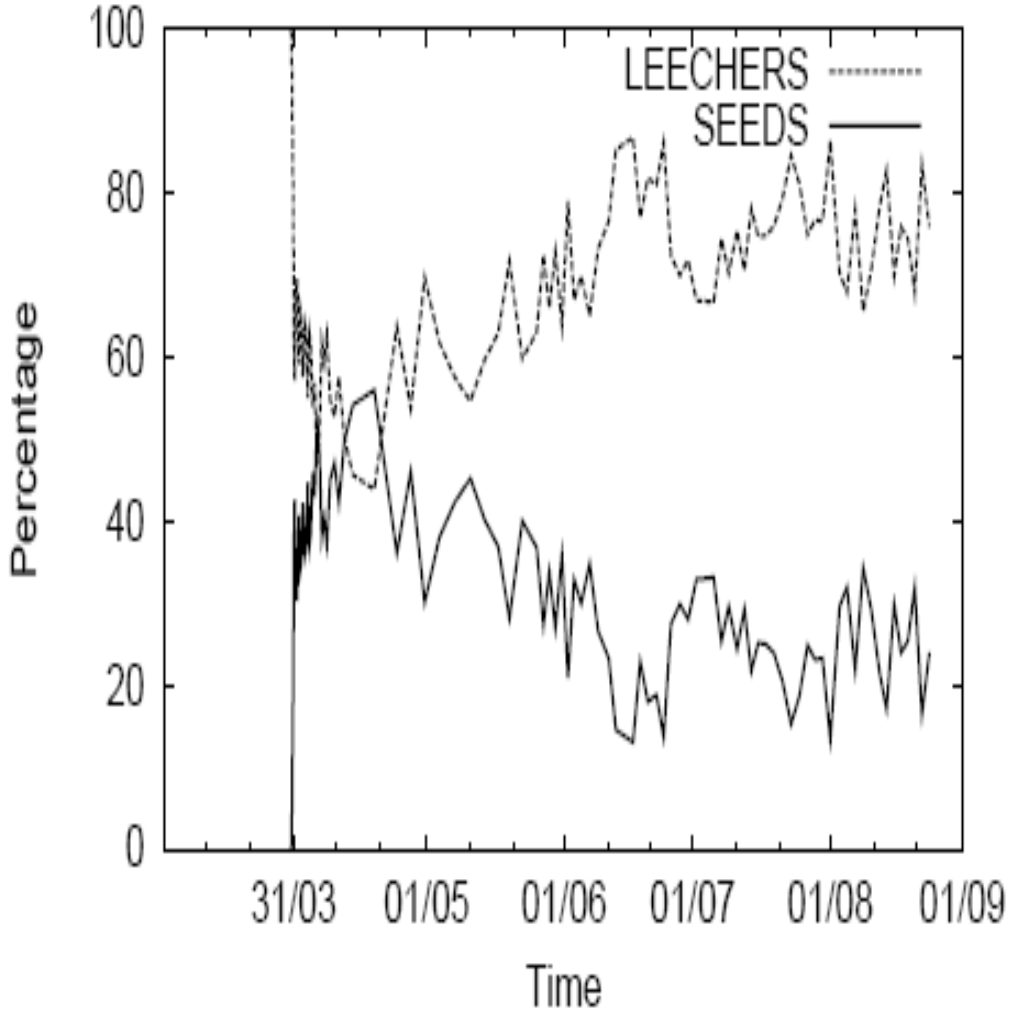
- 5 month trace of the 1.77GB RedHat ISO image
- Two sources of data
  - Tracker statistics
  - Modified client participating in the swarm
- 180,000 clients total
- 50,000 clients in the first five days
  - Flash crowd

Initial flash crowd



Seeds and leechers: altruism





Some statistics

- Average download rate is 500kb/s, during flash crowd, active clients averged at 800kb/s
- 5% of sessions is “seed session”
  - Joining peer already has to complete file, joins only to share it
- About 50% of sessions (peer joins) belong to peers that spend little time in the network and down/upload little data
  - Maybe disappointed users behind slow links

Summary

- BitTorrent: simple (by design and also to use), almost optimal and works $\rightarrow$ it is popular
- The devil is in the details too (good efficient client)
- Only slight problem: endgame
  - Last chunks in endgame mode: aggressive parallel downloads to maximize speed
  - Does not result in very significant overhead

Network coding

- In bittorrent: chunk selection and peer selection are important to make sure that
  - All chunks are represented equally
  - We have a random network
- We can get rid of these using coding theory
  - Works even if overlay has bottlenecks
  - No need to worry about chunk selection

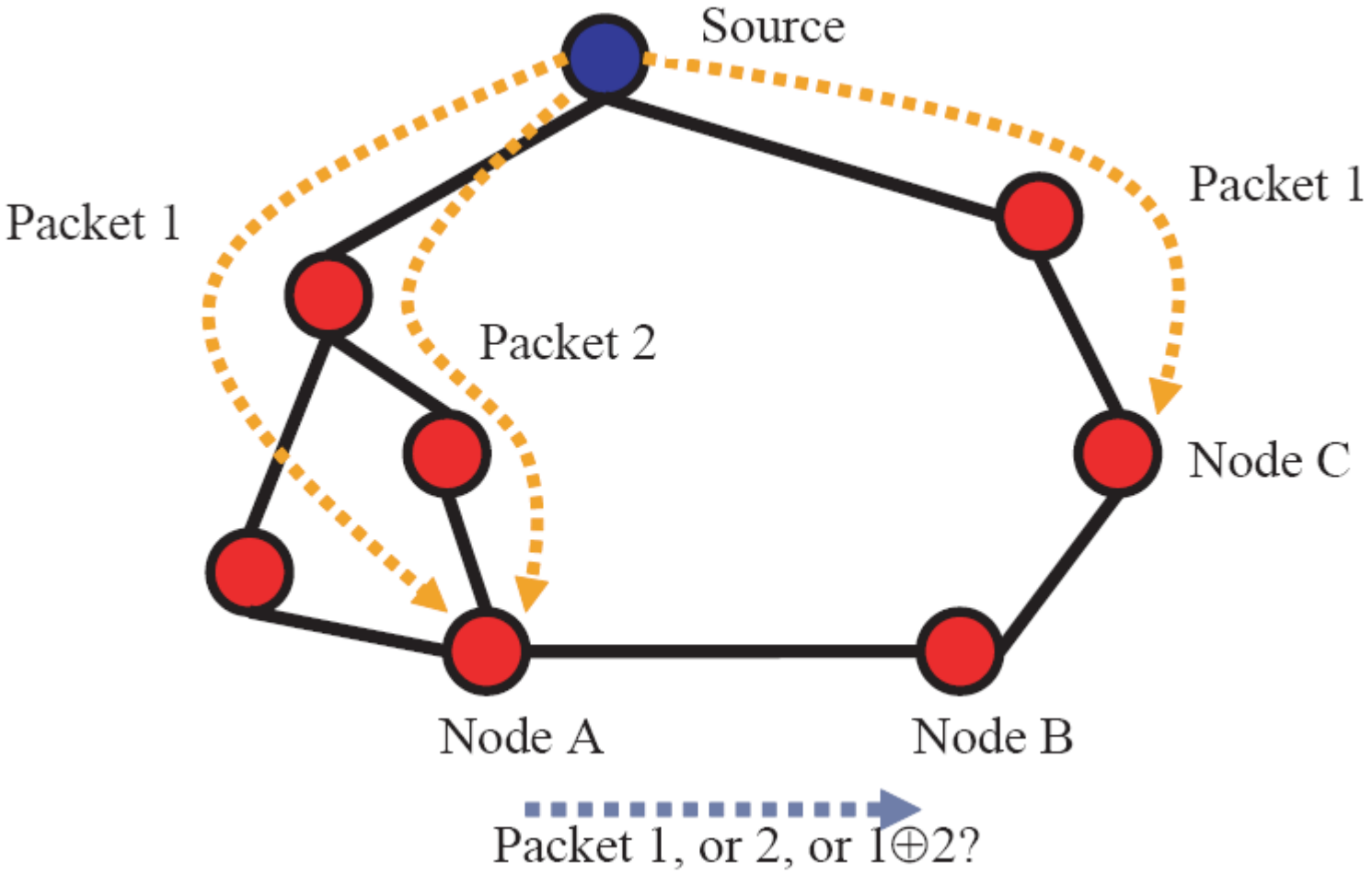
Coding theory for CDNs

- Erasure codes
  - Data is divided into k packets
  - Transformed into n>k packets such that any k packets can reconstruct the original data (erasure codes)
  - Reed-Solomon or Tornado codes
- Implementing a digital fountain
  - “fountain” keeps transmitting these n packets
  - Downloaders can join at any time, can catch any k of the packets (perhaps from neighbors) and leave

Network coding

- Not only server does encoding but also the clients
- A huge, practically unlimited number of different packets are floating around, generated by clients concurrently
- Any k of these packets is enough for decoding
- Possible coding approach: linear combination over finite fields
  - All codes are linear combinations of the original packets
  - Clients create new linear combinations when they offer content
  - Decoding is solving a linear system of equations

Advantage of network coding



Only the number of packets counts, no worries about which packet to fetch

A possible protocol

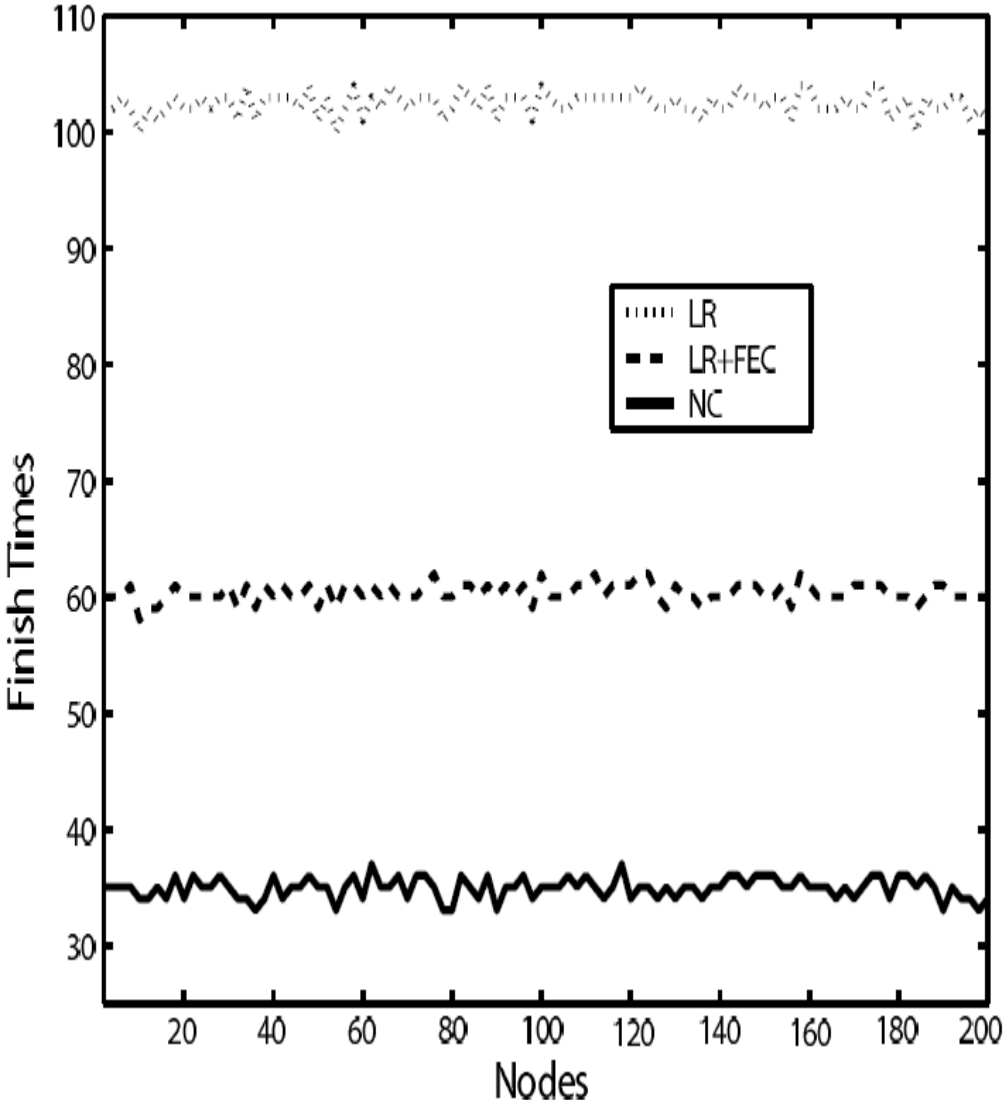
- Same as BitTorrent, only
  - Clients offer new random linear combinations for download and transfer the coefficients as well (low overhead)
  - There is no chunk selection problem
    - **No rare chunks can occur**
    - **No endgame problem**
    - **No topology bottleneck problem**
    - **No data loss problem due to catastrophic failure**
- Same incentive mechanisms too (tit-for-tat), but with explicit accounting (no more upload than download)

Experimental results

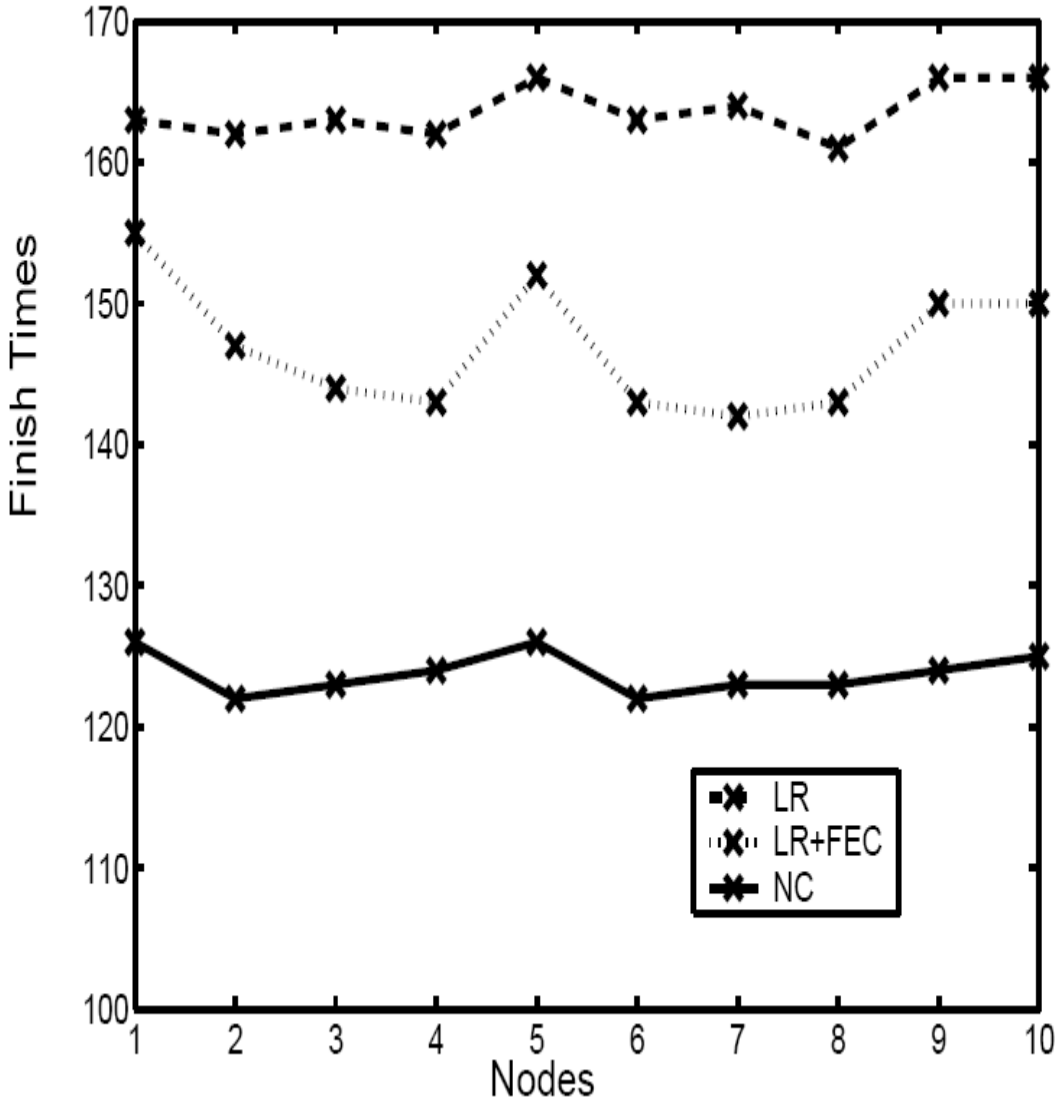
- Three algorithms
  - Local rarest chunk selection (LR) (similar to BitTorrent)
  - Local rarest combined with server encoding
  - Network encoding
- Network size is 200 (small!)
- Neighbors is max 6 (small!)
- Different scenarios
  - Clustered topology, heterogeneity, dynamism (If random network and homogeneous static peers, then the strategies are very similar)

Clustering and heterogeneity

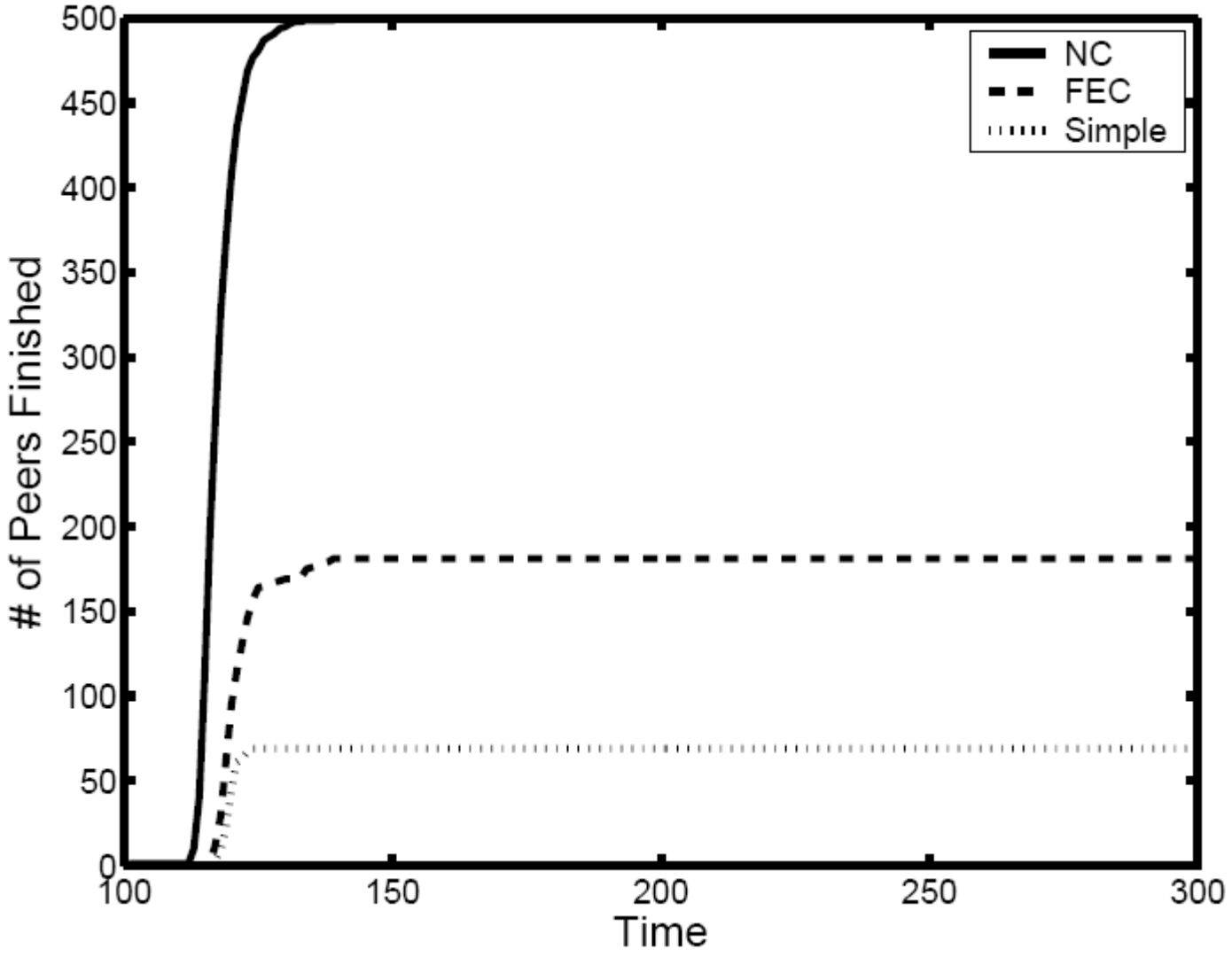
Two clusters: 100 nodes each



10 fast nodes (4x faster) 190 slow nodes



Server availability



Server leaves after serving all chunks plus 5% extra chunks, nodes immediately leave when finished

server coding needs 10-15%, no coding 20-30% extra chunks to achieve full coverage

Final note

- BitTorrent is in fact quite good in practice
  - No network bottlenecks occur because a random network is maintained
  - Rarest chunk policy is very good (combined with initial random chunk and end-game strategies)
  - Heterogeneity might be an issue (in practice low capacity nodes simply go away, as we saw...)
- A convincing study is still to be written with larger scale systems and a more complete BT implementation

Media streaming

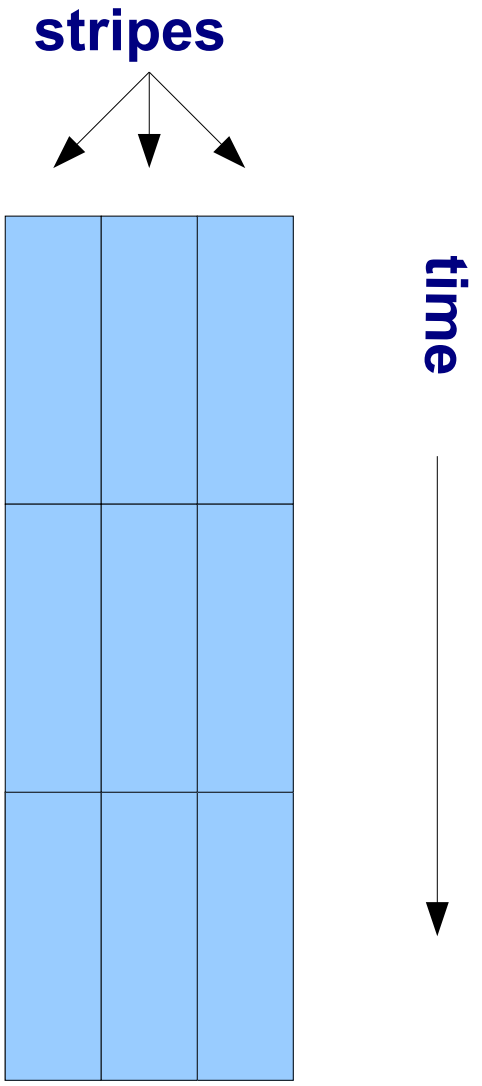
- Similar to distribution of large files but time is important
  - Packets must have low delay
  - If we do not get a packet for some time, we forget about it
- Classification as before
  - Dedicated router infrastructure (Cisco, etc)
  - Dedicated application layer overlay (Akamai, etc)
  - P2P cooperative approaches
- We look at SplitStream and Bullet (both P2P)

Multiple description coding

- We have seen erasure codes for large file distribution
  - Here any k packets were enough for decoding, but k-1 packets is of not much help
- Multiple description code (MDC) is similar
  - k packets are enough for decoding
  - Less than k packets can be used to approximate content
    - **Similar to progressive encoding, only order of packets is insignificant**
    - **Useful for multimedia (video, audio) but not for other data**

Multiple description coding

- Media streaming applications often use MDC in some form because
  - Loosing a packet results in no interruption, only quality degradation
  - Lower bandwidth nodes simply ask for < k packets
- Streams can be sliced into parallel “stripes” that are MDC encoded



Trees: not optimal

- The most natural way of cooperative media streaming is through broadcast trees
- Trees have problems though, esp in end-to-end approaches
  - Vulnerable to failure (no cycles)
  - Bandwidth strictly decreases towards leaves
  - Difficult to create optimal tree (and it is important to do so)
  - Leaves do not contribute in a cooperative setting

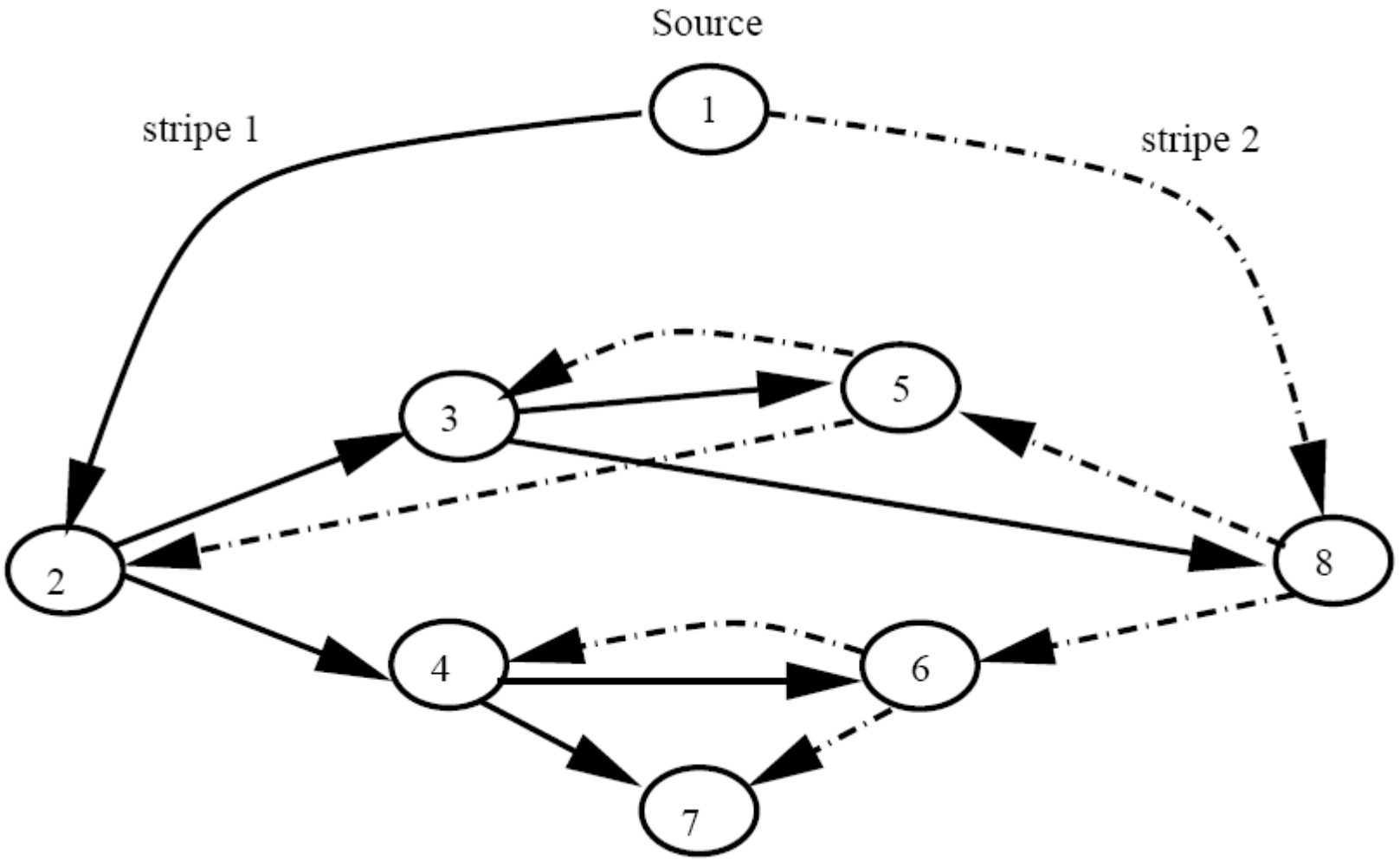
Solving the problems with trees

- Use multiple trees
- Use a tree but also use a mesh for cooperation
- Axe them (Chainsaw, IPTPS 2005)
  - We do not discuss this here, although remarkable
- In the following we look at
  - SplitStream that uses multiple trees
  - Bullet that uses the union of a mesh and a tree

SplitStream

- Basic idea
  - Split the stream into k stripes (perhaps with MDC encoding)
  - For each stripe create a multicast tree such that the forrest
    - **Contains interior-node-disjoint trees**
    - **Respects nodes' individual bandwidth constraints**
- Approach
  - Use Scribe (and some hacks) to create the forrest
  - Scribe is on top of Pastry

Illustration of SplitStream



The forrest construction problem

- A constraint satisfaction problem
  - All nodes have incoming capacity requirements (number of stripes they need) and outgoing capacity limits
  - There is one or more source for each stripe
  - We have to construct a weighted directed acyclic distribution graph (forrest) that respects these constraints
- An observation: such a forrest exists if
  - Sum of incoming capacity is less then or equal to the sum of outgoing capacity over the nodes and
  - All nodes that have large outgoing than incoming capacities must posess (receive or originate) all stripes

Constructing the forrest: scribe

- Scribe works over Pastry
  - Mutlicast groups are identified by an ID
  - Tree is definied by the route towards the ID in the Pastry network
  - Join: route towards the ID, connect to first member as child
- Basic idea
  - All k stripes are assigned a group ID, and Scribe is used to create mutlicast trees
  - This does not necessarily satisfy constraints

Constructing the forrest

- Additional tricks for constraint satisfaction
  - Group IDs start with a different letter: interior-node-disjoint forrest
  - If a node has too many children
    - **“Push-down” approach: joining node looks for a parent further down the tree, or if not found, in the “spare capacity group”**
  - Spare capacity group
    - **Scribe group that contains nodes that can take more children**
- Algorithm always succeeds if all nodes want to receive all stripes or suceeds with a high probability as a function of spare capacity and minimal incoming capacity

Bullet

- Basic idea
  - Use a multicast tree as a basis
  - In addition each node continuously looks for peers to download from
  - In effect, the overlay is a tree combined with a random network
- Approach
  - A service (ranSub) that provides random peers
  - A mechanism to select “good” peers
  - Low level transfer protocol (to replace TCP)

Bullet: RanSub

- Two phases
  - Collect phase: using the tree, membership info is propagated upwards (random sample and subtree size)
  - Distribution phase: moving down the tree, all nodes are provided with a random sample from the entire tree, or from the non-descendant part of the tree, etc.
- Nodes in the network receive random peers this way end select those that seem to be most useful

Bullet

- When selecting a peer, first a similarity measure is calculated
  - Based on “summary-sketches”
- Before exchange missing packets need to be identified
  - Bloom filter of available packets is exchanged (usual false positive issue)
  - Old packets are removed from the filter (to keep the size of the set constant)
- Periodically re-evaluate senders (how useful they are)
  - If needed, senders are dropped and new ones are requested

Some comments

- Tree is needed
  - Because of RanSub: but other sampling services can be used that do not rely on trees
  - To maximize diversity of packets in the network: but rarest first chunk selection in BT does the same, besides, with encoding techniques, it is irrelevant
- So is the tree needed?
- Isn't the protocol unnecessarily complex trying to explicitly control things that are “for free” in simpler approaches?
  - Eg in BT through the local-rarest-first strategy

Some refs

- Papers this presentation used material from
  - C Gkantsidis and P R Rodriguez. Network coding for large scale content distribution. In INFOCOM 2005, pp 2235–2245, 2005.
  - M Freedman, E Freudenthal, and D Mazières. Democratizing content publication with Coral. In NSDI '04, 2004.
  - M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garcés-Erice. Dissecting bittorrent: Five months in a torrent's lifetime. In Passive and Active Network Measurement, LNCS 3015, pages 1–11. Springer, 2004.
  - M Castro, P Druschel, A-M Kermarrec, A Nandi, A Rowstron, and A Singh. Splitstream: high-bandwidth multicast in cooperative environments. In SOSP'03, pages 298–313, New York, NY, USA, 2003
  - B Cohen. Incentives build robustness in bittorrent. In P2PECON, 2003.
  - D Kostic, A Rodriguez, J Albrecht, and A Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In SOSP'03, pages 282–297
- The course website
  - http://www.inf.u-szeged.hu/~jelasity/p2p/