

# CS5412: TORRENTS AND TIT-FOR-TAT

Lecture VII

Ken Birman

# BitTorrent

2

- Widely used download technology
- Implementations specialized for setting
  - ▣ Some focus on P2P downloads, e.g. patches
  - ▣ Others focus on use cases internal to corporate clouds

# BitTorrent

3

- The technology really has three aspects
  - ▣ A standard that BitTorrent client systems follow
  - ▣ Some existing clients, e.g. the free Torrent client, PPLive
  - ▣ A clever idea: using “tit-for-tat” mechanisms to reward good behavior and to punish bad behavior (reminder of the discussion we had about RON...)
  
- This third aspect is especially intriguing!

# The basic BitTorrent Scenario

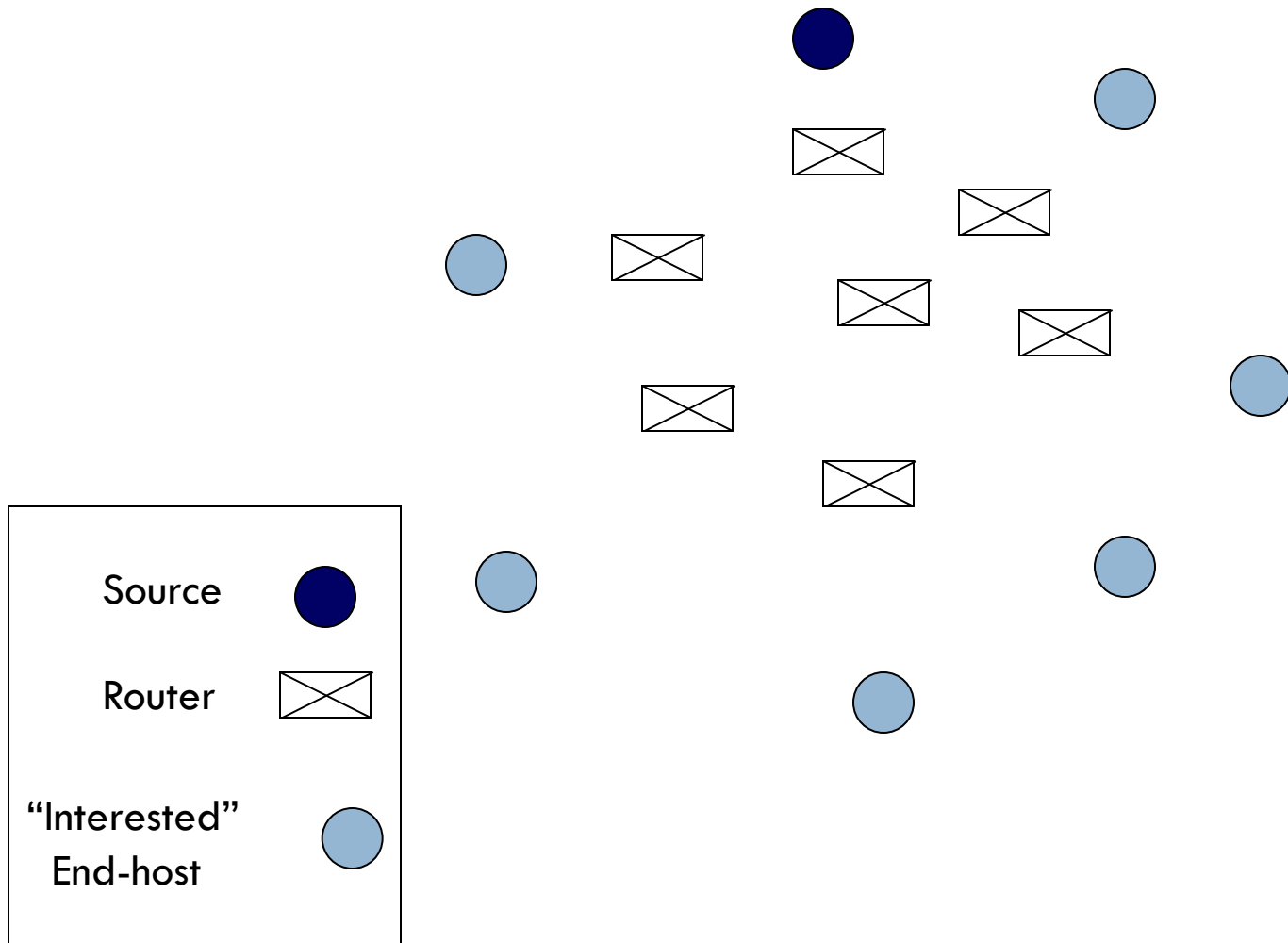
4

- Millions want to download the same popular huge files (for free)
  - ISO's
  - Media (the real example!)
- Client-server model fails
  - Single server fails
  - Can't afford to deploy enough servers

# Why not use IP Multicast?

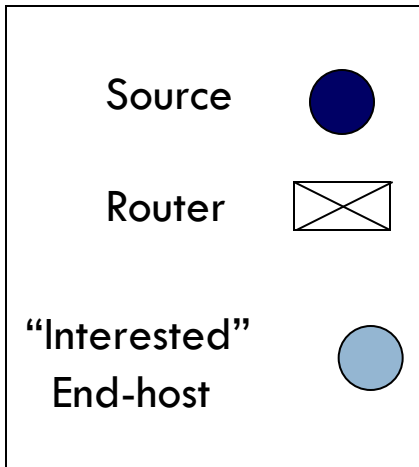
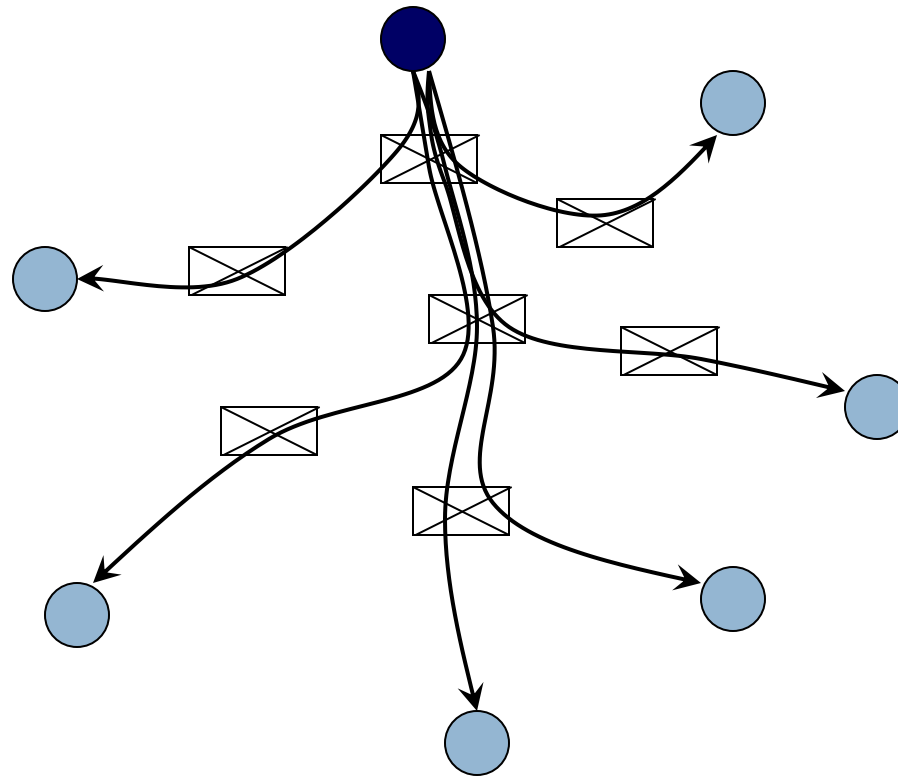
5

- IP Multicast not a real option in general WAN settings
  - ▣ Not supported by many ISPs
  - ▣ Most commonly seen in private data centers
- Alternatives
  - ▣ End-host based Multicast
  - ▣ BitTorrent
  - ▣ Other P2P file-sharing schemes (from prior lectures)



# Client-Server

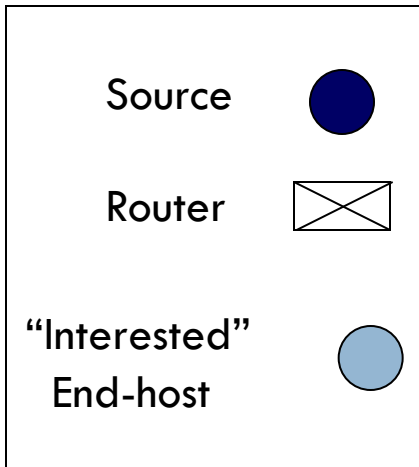
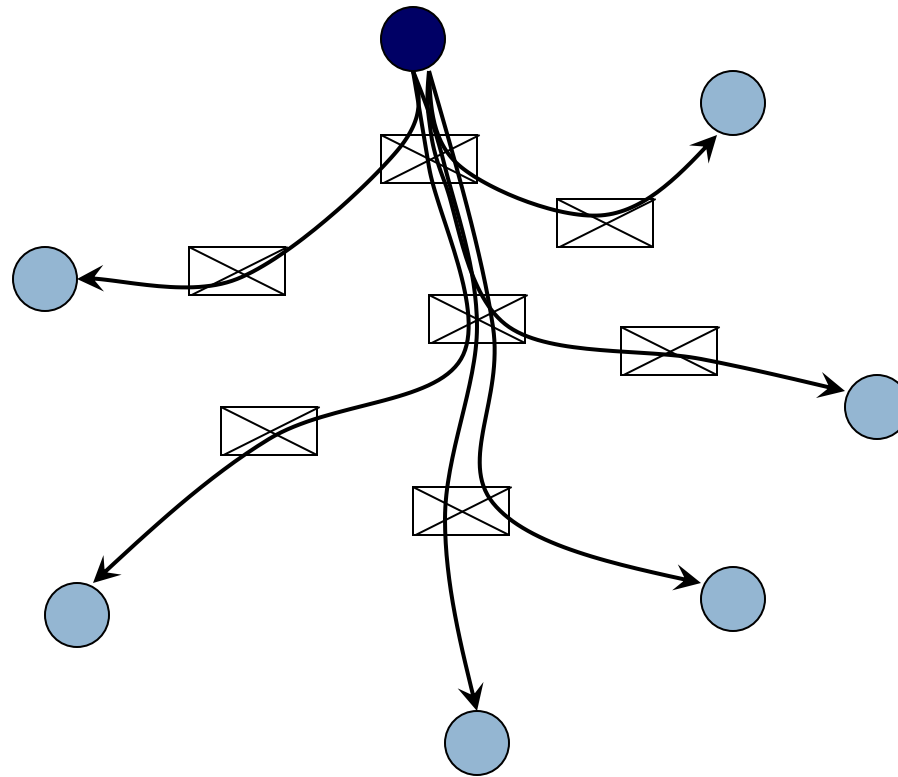
7



# Client-Server

8

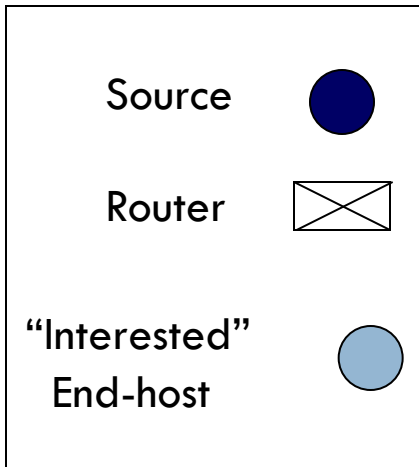
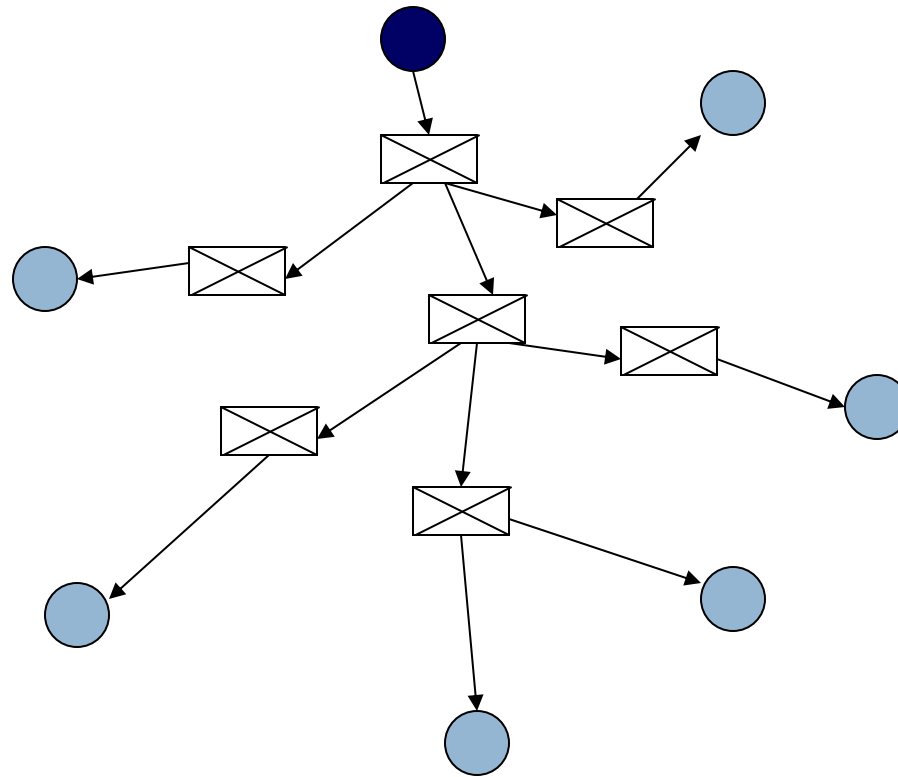
Overloaded!





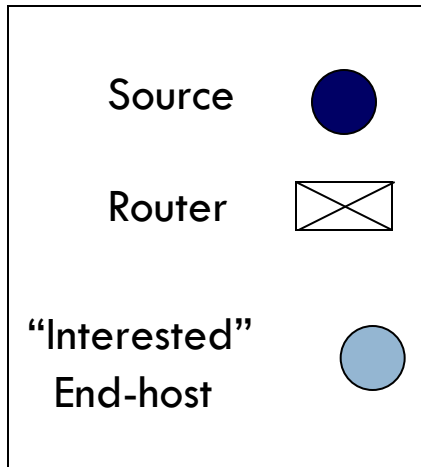
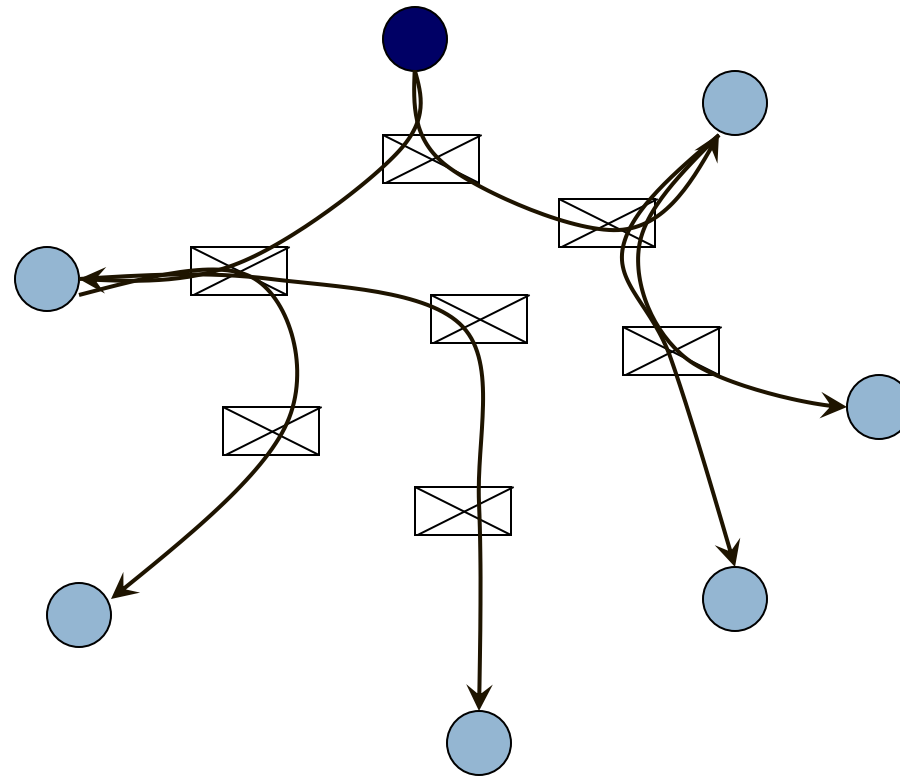
# IP multicast

9



# End-host based multicast

10



# End-host based multicast

11

- “Single-uploader” → “Multiple-uploaders”
  - Lots of nodes want to download
  - Make use of their *uploading* abilities as well
  - Node that has downloaded (part of) file will then upload it to other nodes.
  - Uploading costs amortized across all nodes

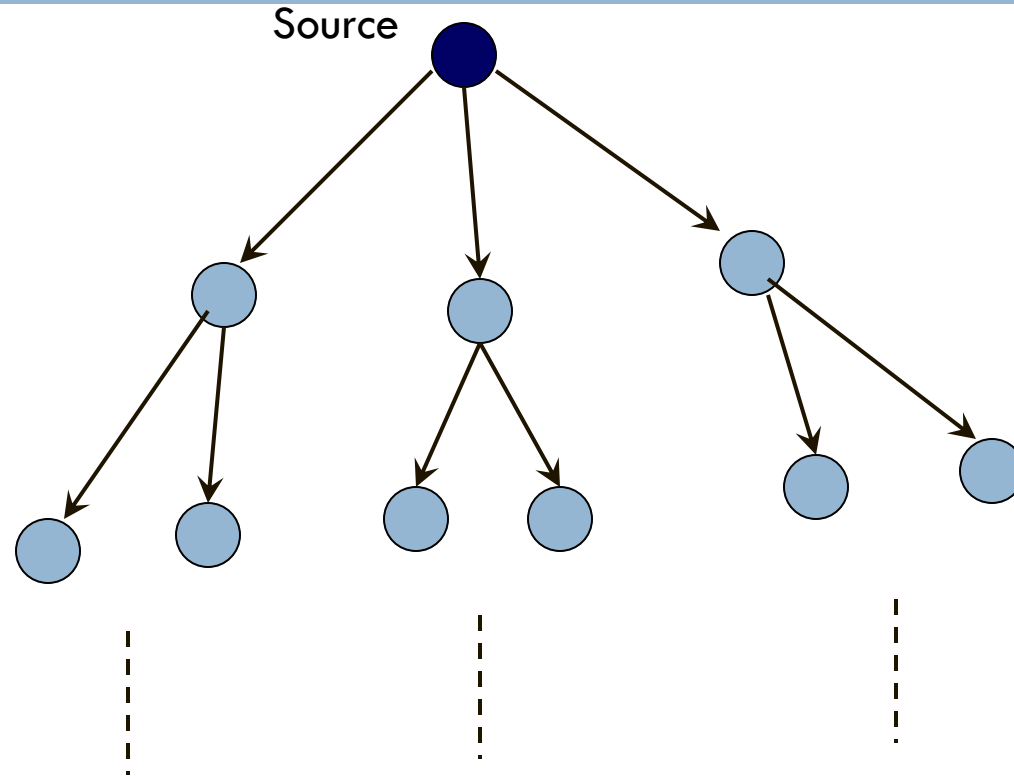
# End-host based multicast

12

- Also called “Application-level Multicast”
- Many protocols proposed early this decade
  - ▣ Yoid (2000), Narada (2000), Overcast (2000), ALMI (2001)
    - All use single trees
    - Problem with single trees?

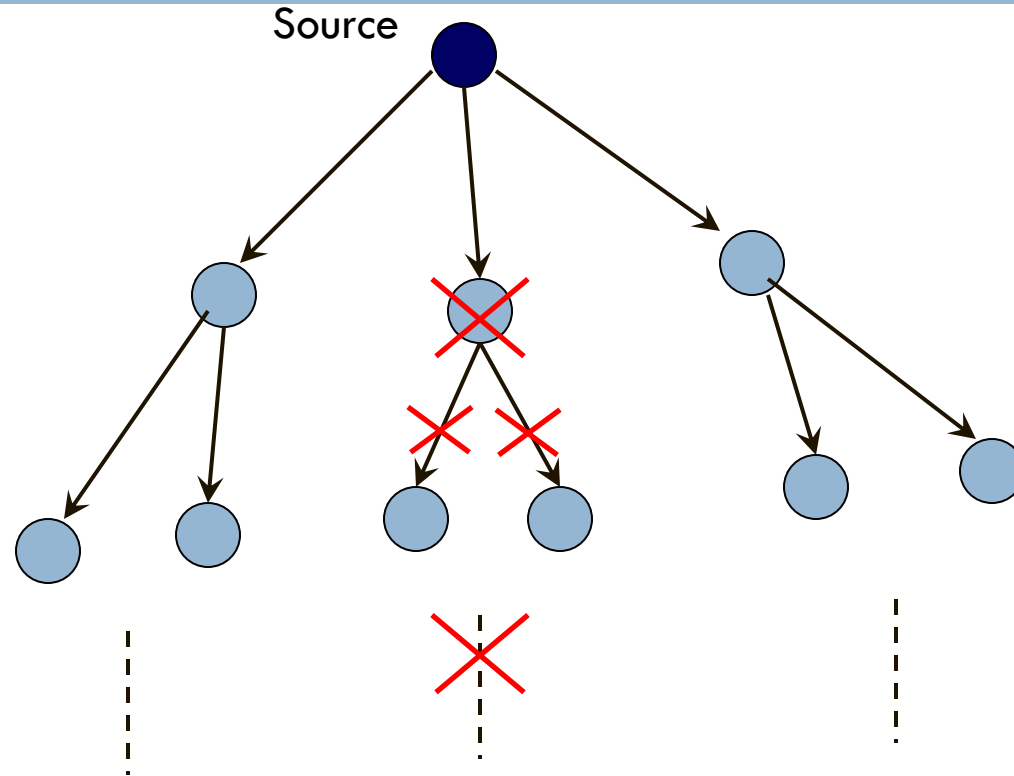
# End-host multicast using single tree

13



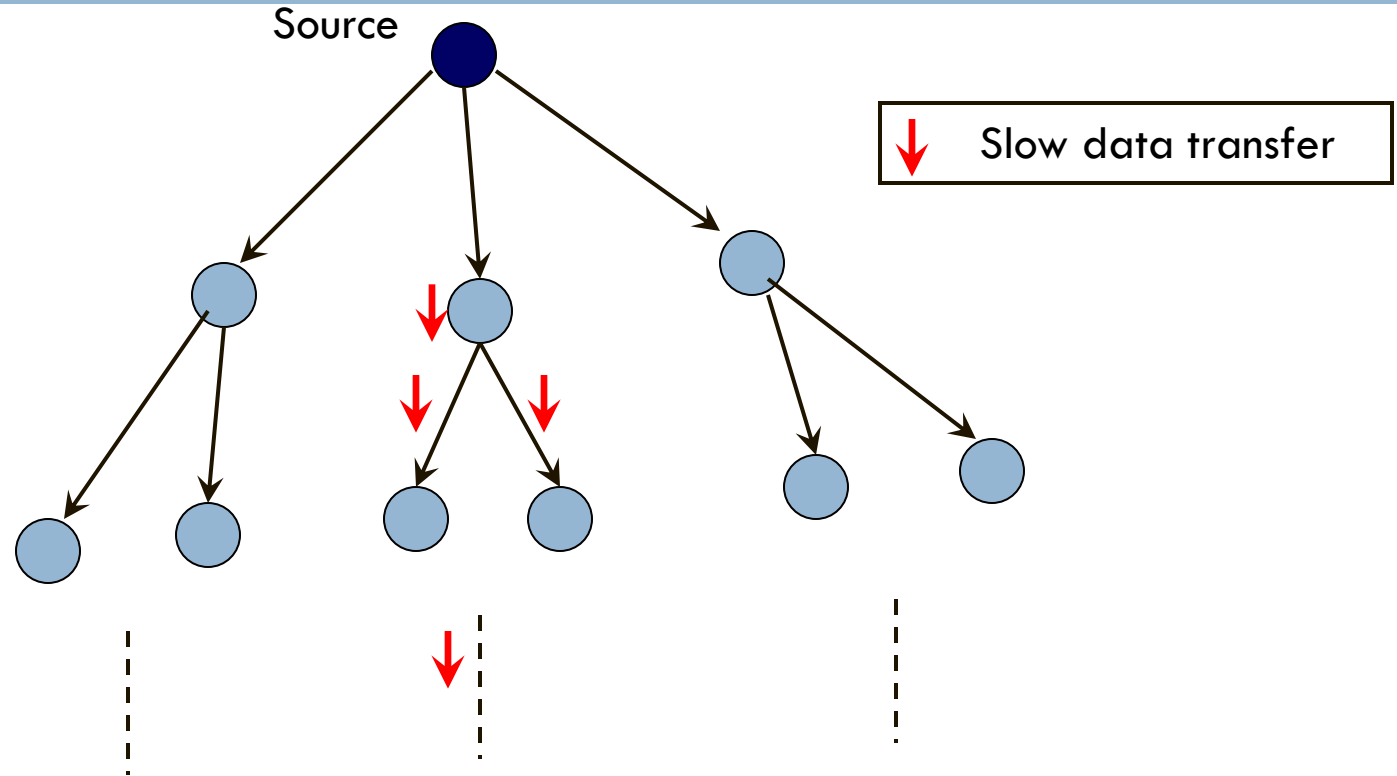
# End-host multicast using single tree

14



# End-host multicast using single tree

15



# End-host multicast using single tree

16

- Tree is “push-based” – node receives data, pushes data to children
- Failure of “interior”-node affects downloads in entire subtree rooted at node
- Slow interior node similarly affects entire subtree
- Also, leaf-nodes don't do any sending!
- Though later multi-tree / multi-path protocols (Chunkyspread (2006), Chainsaw (2005), Bullet (2003)) mitigate some of these issues



# BitTorrent

17

- Written by Bram Cohen (in Python) in 2001
- “Pull-based” “swarming” approach
  - ▣ Each file split into smaller **pieces**
  - ▣ Nodes request desired pieces from neighbors
    - As opposed to parents pushing data that they receive
  - ▣ Pieces not downloaded in sequential order
  - ▣ Previous multicast schemes aimed to support “streaming”; BitTorrent does not
- Encourages contribution by all nodes

# BitTorrent Swarm

18

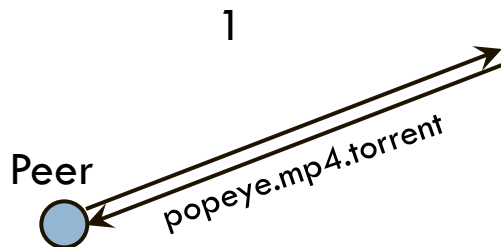
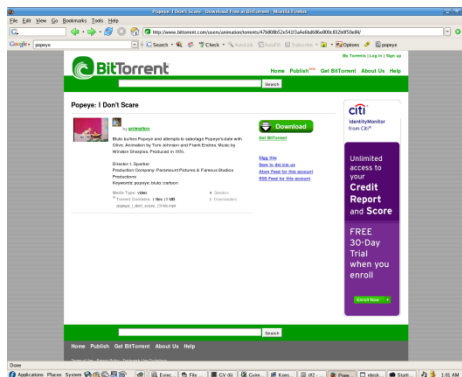
- **Swarm**
  - ▣ Set of peers all downloading the same file
  - ▣ Organized as a random mesh
- Each node knows list of pieces downloaded by neighbors
- Node requests pieces it does not own from neighbors
  - ▣ Exact method explained later

# How a node enters a swarm for file “popeye.mp4”

- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of **tracker** for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

# How a node enters a swarm for file “popeye.mp4”

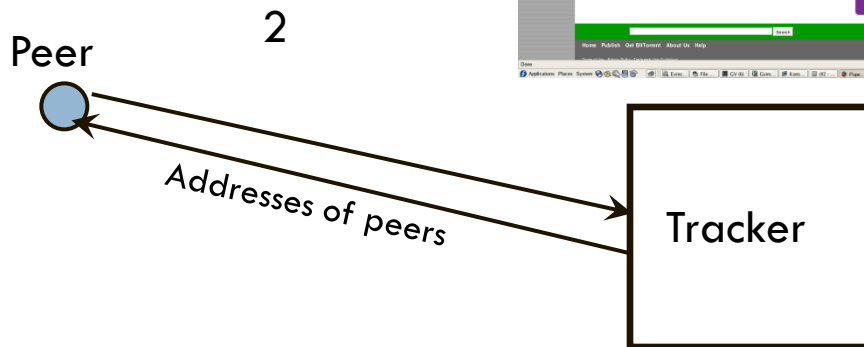
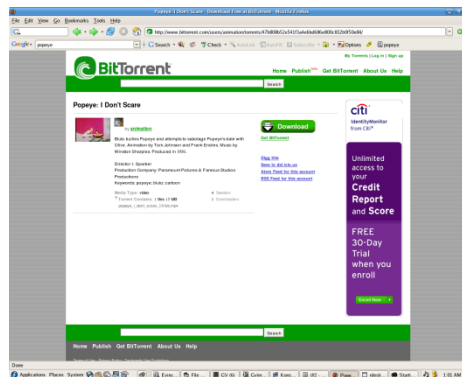
www.bittorrent.com



- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of **tracker** for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

# How a node enters a swarm for file “popeye.mp4”

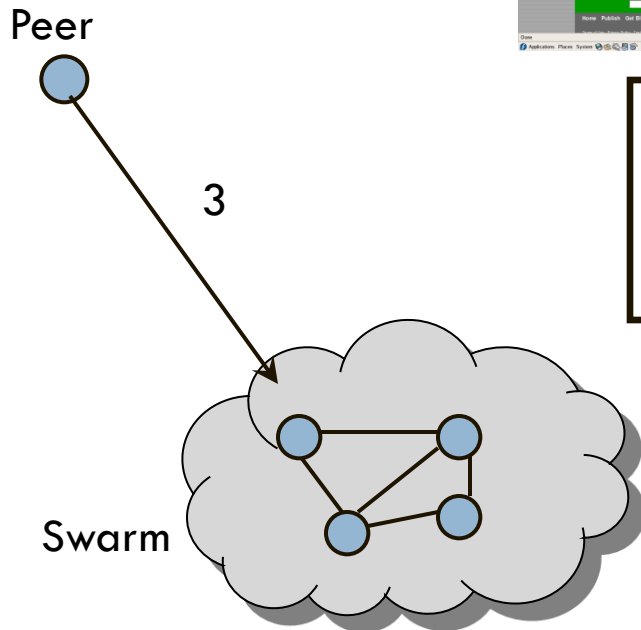
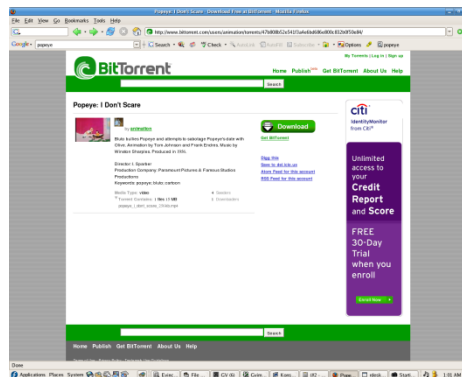
www.bittorrent.com



- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of **tracker** for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

# How a node enters a swarm for file “popeye.mp4”

www.bittorrent.com



- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of **tracker** for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

# Contents of .torrent file

23

- URL of tracker
- Piece length – Usually 256 KB
- SHA-1 hashes of each piece in file
  - ▣ For reliability
- “files” – allows download of multiple files

# Terminology

24

- **Seed:** peer with the entire file
  - ▣ Original Seed: The first seed
- **Leech:** peer that's downloading the file
  - ▣ Fairer term might have been “downloader”
- **Sub-piece:** Further subdivision of a piece
  - ▣ The “unit for requests” is a subpiece
  - ▣ But a peer uploads only after assembling complete piece



# Peer-peer transactions:

## Choosing pieces to request

25

- **Rarest-first:** Look at all pieces at all peers, and request piece that's owned by fewest peers
  - Increases diversity in the pieces downloaded
    - avoids case where a node and each of its peers have exactly the same pieces; increases throughput
  - Increases likelihood all pieces still available even if original seed leaves before any one node has downloaded entire file

# Choosing pieces to request

26

## □ Random First Piece:

- When peer starts to download, request random piece.
  - So as to assemble first complete piece quickly
  - Then participate in uploads
- When first complete piece assembled, switch to rarest-first

# Choosing pieces to request

27

- **End-game mode:**
  - ▣ When requests sent for all sub-pieces, (re)send requests to all peers.
  - ▣ To speed up completion of download
  - ▣ Cancel request for downloaded sub-pieces

# Tit-for-tat as incentive to upload

28

- Want to encourage all peers to contribute
- Peer  $A$  said to **choke** peer  $B$  if it ( $A$ ) decides not to upload to  $B$
- Each peer (say  $A$ ) unchokes at most 4 *interested* peers at any time
  - ▣ The three with the largest upload rates to  $A$ 
    - Where the tit-for-tat comes in
  - ▣ Another randomly chosen (**Optimistic Unchoke**)
    - To periodically look for better choices

# Anti-snubbing

29

- A peer is said to be snubbed if each of its peers chokes it
- To handle this, snubbed peer stops uploading to its peers
- Optimistic unchoking done more often
  - ▣ Hope is that will discover a new peer that will upload to us

# Why BitTorrent took off

30

- Better performance through “pull-based” transfer
  - ▣ Slow nodes don’t bog down other nodes
- Allows uploading from hosts that have downloaded parts of a file
  - ▣ In common with other end-host based multicast schemes

# Why BitTorrent took off

31

- Practical Reasons (perhaps more important!)
  - ▣ Working implementation (Bram Cohen) with simple well-defined interfaces for plugging in new content
  - ▣ Many recent competitors got sued / shut down
    - Napster, Kazaa
  - ▣ Doesn't do "search" per se. Users use well-known, trusted sources to locate content
    - Avoids the pollution problem, where garbage is passed off as authentic content

# Pros and cons of BitTorrent

32

- Pros
  - ▣ Proficient in utilizing partially downloaded files
  - ▣ Discourages “freeloading”
    - By rewarding fastest uploaders
  - ▣ Encourages diversity through “rarest-first”
    - Extends lifetime of swarm
- Works well for “hot content”



# Pros and cons of BitTorrent

33

## □ Cons

- Assumes all interested peers active at same time; performance deteriorates if swarm “cools off”
- Even worse: no trackers for obscure content

# Pros and cons of BitTorrent

34

- Dependence on centralized tracker: pro/con?
  - ☹️ Single point of failure: New nodes can't enter swarm if tracker goes down
  - Lack of a search feature
    - 😊 Prevents pollution attacks
    - ☹️ Users need to resort to out-of-band search: well known torrent-hosting sites / plain old web-search

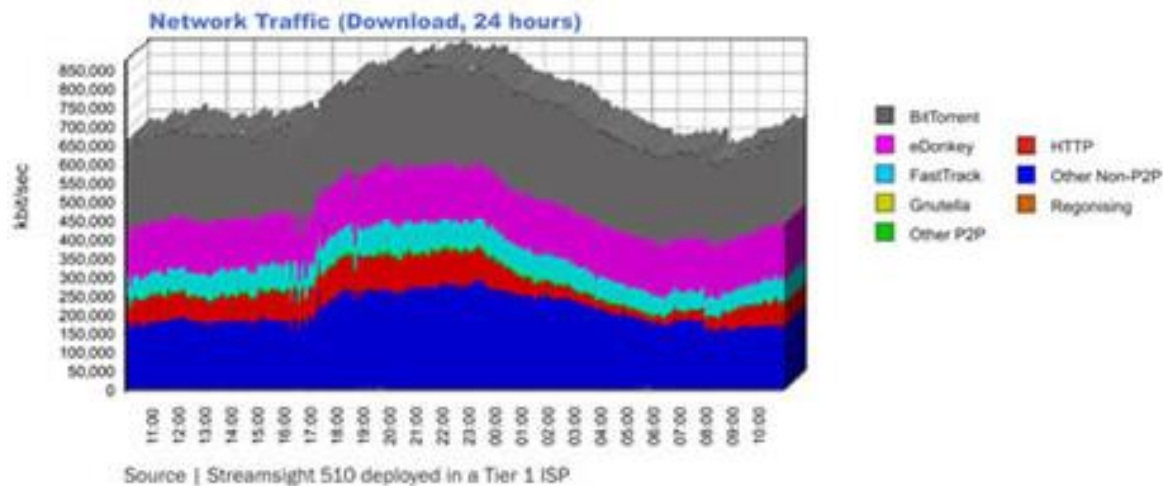
# “Trackerless” BitTorrent

35

- To be more precise, “BitTorrent without a centralized-tracker”
- E.g.: Azureus
- Uses a Distributed Hash Table (Kademlia DHT)
- Tracker run by a normal end-host (not a web-server anymore)
  - ▣ The original seeder could itself be the tracker
  - ▣ Or have a node in the DHT randomly picked to act as the tracker

# Prior to Netflix “explosion”, BitTorrent dominated the INternet!

36



(From CacheLogic, 2004)

# Why is (studying) BitTorrent important?

37

- BitTorrent consumes significant amount of internet traffic today
  - In 2004, BitTorrent accounted for 30% of all internet traffic (Total P2P was 60%), according to CacheLogic
  - Slightly lower share in 2005 (possibly because of legal action), but still significant
  - BT always used for legal software (linux iso) distribution too
  - Recently: legal media downloads (Fox)

# Example finding from a recent study

38

- Gribble showed that most BitTorrent streams “fail”
  - ▣ He found that the number of concurrent users is often too small, and the transfer too short, for the incentive structure to do anything
  - ▣ No time to “learn”
- His suggestion: add a simple history mechanism
- Behavior from yesterday can be used today. But of course this ignores “dynamics” seen in the Internet...

# BAR Gossip

39

- Work done at UT Austin looking at *gossip* model
  - Same style of protocol seen in Kelips
- They ask what behaviors a node might exhibit
  - Byzantine: the node is malicious
  - Altruistic: The node answers every request
  - Rational: The node maximizes own benefit
- Under this model, is there an optimal behavior?

[BAR Gossip. Harry C. Li, Allen Clement, Edmund L. Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, Michael Dahlin. OSDI 2006]

# Basic strategy

40

- They assume cryptographic keys (PKI)
  - ▣ Used to create signatures: detect and discard junk
  - ▣ Also employed to prevent malfactor from pretending that it send messages but they were lost in network
  
- This is used to create a scheme that allows nodes to detect and punish non-compliance



# Key steps in BAR Gossip

41

1. *History exchange*: two parties learn about the updates the other party holds
2. *Update exchange*: each party copies a subset of these updates into a *briefcase* that is sent, encrypted, to the other party
  - Two cases: *balanced exchange* for normal operation
  - *Optimistic push* to help one party catch up
3. *Key exchange*, where the parties swap the keys needed to access the updates in the two briefcases.

# Obvious concern: Failed key exchange

42

- What if a rational node chooses not to send the key (or sends an invalid key)?
  - Can't "solve" this problem; they prove a theorem
  - But by tracking histories, BAR gossip allows altruistic and rational nodes to operate *fairly enough*
- Central idea is that the balanced exchange should reflect the quality of data exchanged in past
  - This can be determined from the history and penalizes a node that tries to cheat during exchange
  - Nash equilibrium strategy is to send the keys, so rational nodes will do so!

# Outcomes achieved

43

- BAR gossip protocol provides good convergence as long as:
  - ▣ No more than 20% of nodes are Byzantine
  - ▣ No more than 40% collude.
  
- Generally seen as the “ultimate story” for BitTorrent-like schemes

# Insights gained?

44

- Collaborative download schemes can improve download speeds very dramatically
  - ▣ They avoid sender overload
  - ▣ Are at risk when participants deviate from protocol
  - ▣ Game theory suggests possible remedies
- BitTorrent is a successful and very practical tool
  - ▣ Widely used inside data centers
  - ▣ Also popular for P2P downloads
  - ▣ In China, PPLive media streaming system very successful and very widely deployed

# References

45

- BitTorrent
  - ▣ “Incentives build robustness in BitTorrent”, Bram Cohen
  - ▣ BitTorrent Protocol Specification:  
<http://www.bittorrent.org/protocol.html>
- Poisoning/Pollution in DHT's:
  - ▣ “Index Poisoning Attack in P2P file sharing systems”
  - ▣ “Pollution in P2P File Sharing Systems”