



**HAL**  
open science

# Peer-to-Peer Applications : From BitTorrent to Privacy

Arnaud Legout

► **To cite this version:**

Arnaud Legout. Peer-to-Peer Applications : From BitTorrent to Privacy. 3rd cycle. 2010. cel-00544132v1

**HAL Id: cel-00544132**

**<https://cel.hal.science/cel-00544132v1>**

Submitted on 7 Dec 2010 (v1), last revised 6 Jan 2012 (v2)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Peer-to-Peer Applications

**Arnaud Legout**

**INRIA, Sophia Antipolis, France  
Projet Planète**

Email: [arnaud.legout@inria.fr](mailto:arnaud.legout@inria.fr)

# Outline

## □ Overview

- What is a P2P application?
- Popularity of P2P applications?

## □ Content Replication

## □ BitTorrent

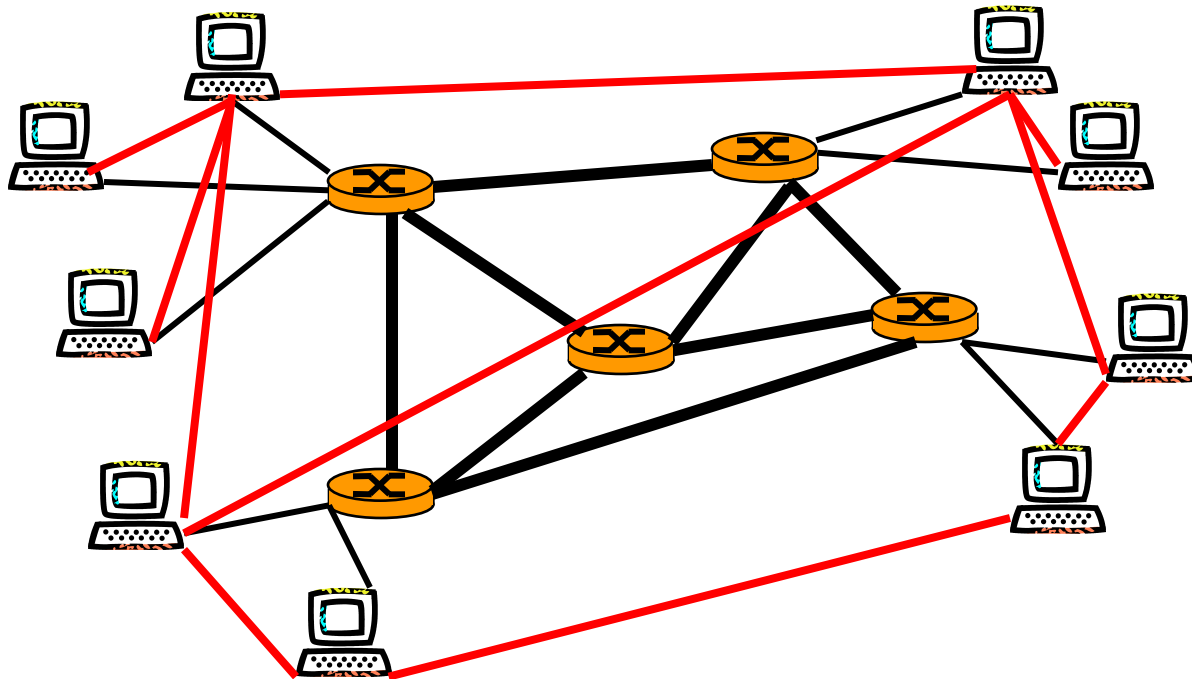
## □ Security

## □ Localization

# Definition: Overlay

## □ Overlay Network

- Network at the application layer (layer 7)



# Definition: Overlay

- ❑ Formed by **communicating** among themselves
  - Dedicated machines
  - End-users
- ❑ Types of overlay
  - General purpose overlay (application-layer multicast)
  - Application specific overlay (CDN)
- ❑ Overlay construction
  - Network topology
  - Network metrics (delay, bandwidth, etc.)

# Definition: Overlay

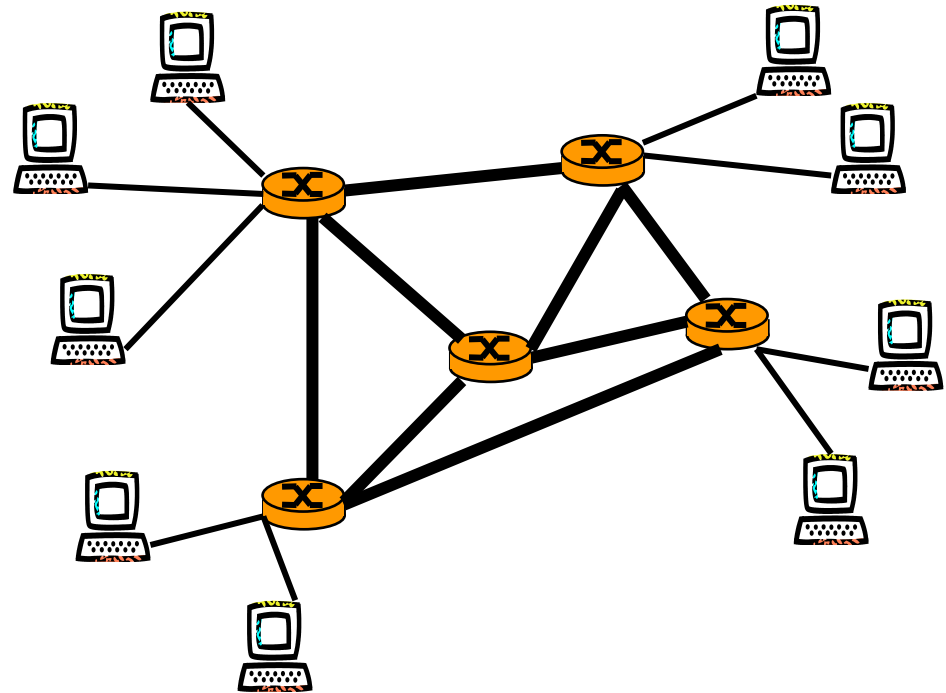
## □ Why do we need overlays?

- Create a service that is not (or that cannot be) provided by the network (layer 3)
  - Create an application layer service
- Example of services
  - Application layer multicast
  - Content Delivery Network (CDN)
  - DNS (IP only provides IP addresses and don't know how to route on names)

# Definition: Peer

## □ Peer

- A computer, an end-user, an application, etc.
  - Depends on the context
  - Always an **end system**, but an end system is not always a peer
  - An end system can be a dedicated video server that is part of a CDN, or a BitTorrent client that is part of a P2P network



# Definition: Peer

## □ Leecher

- A peer that is client and server
- In the context of content delivery
  - Has a partial copy of the content

## □ Seed

- A peer that is only server
- In the context of content delivery
  - Has a full copy of the content



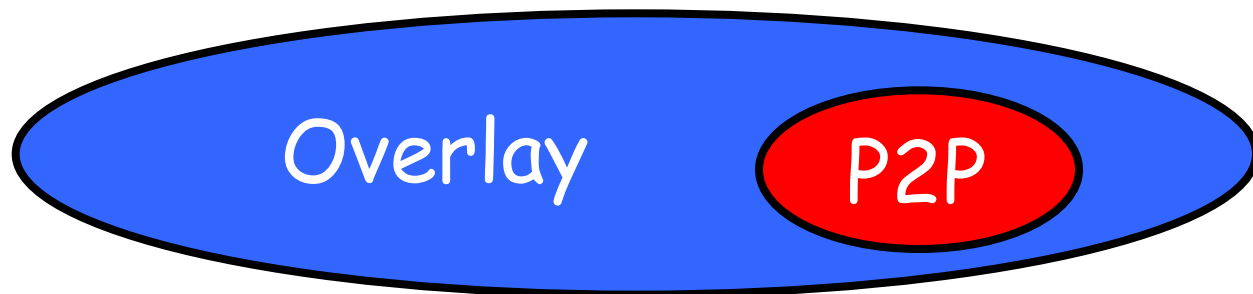
# Definition: P2P

## □ Peer-to-peer applications

- No general definition
- Application specific
  - If not, overlay is a more appropriate definition
- At least two peers
- Every peer is a client and a server
  - For a same application
  - Possibility of hierarchy
- Peers form an overlay network

# Definition: P2P

- Overlay Network vs. P2P applications
  - A P2P application forms an overlay network
  - An overlay network is not always a P2P application
  - Trend to define a P2P application as overlay network formed by end-users
  - Depends on the definition of P2P



# Example: Web

## □ The case of the Web

- Service: HTML pages access
- Pages served only by dedicated machines (HTTP servers)
  - End-users cannot serve HTML pages
- No share of HTML pages among servers: servers are not communicating among themselves, but with clients
- This is not an overlay network!

# Example: Email Servers

- The case of Email servers
  - Service: Email delivery
  - POP/SMTP/IMAP servers are dedicated machine
  - Email servers communicate to deliver emails
  - This is an overlay network!
  - But, not a P2P application
  - Probably the oldest example of overlay

# The New P2P Paradigm

- ❑ Web, Email, etc. is an old technology
- ❑ Is overlay network an old techno?
- ❑ Yes, when applied to servers
- ❑ But, its applications to end-users is new
  - New applications
  - New problems
  - New techniques, algorithms, protocols
  - This is P2P!

# The New P2P Paradigm

- Why P2P applications became popular recently
  - High speed Internet connections
  - Power shift from servers to end-users
    - End-to-end argument [7] (1984) undoubtedly visionary
      - Still alive (01/2006):  
<http://lwn.net/Articles/169961/>
- P2P applications are a true revolution
  - Aside TCP/IP and the Web

# New P2P applications

- P2P applications capitalize on any resource from anybody
  - P2P applications can share CPU, bandwidth and storage
    - seti@home (not P2P, but distributed)
    - BitTorrent, Emule, Gnutella
    - Skype, Google talk
    - Publius

# Focus of this Course

- ❑ P2P **file sharing** applications
- ❑ Main focus on file replication
  - This is the true revolution
- ❑ But also
  - Distributed security
  - Peer anonymity



# P2P file sharing taxonomy

- Current taxonomy for P2P file sharing [11]
  - Unstructured P2P: BitTorrent, Gnutella, KaZaA, etc.
  - Structured: DHT (e.g., Chord, CAN, Pastry, Kademlia, etc.)
- What is wrong?
  - Assume that P2P applications must be classified according to their content localization architecture

# P2P file sharing taxonomy

- Proposed taxonomy for P2P file sharing
  - Content localization
    - Unstructured
    - Structured
  - Content replication
    - Parallel download
    - File splitting
    - Piece selection
      - rarest first, random, sequential, etc.
    - Peer selection
      - choke algorithm, tit-for-tat, priority queue, etc.

# P2P file sharing taxonomy

## □ Is it better?

- No more a focus on content localization
- Decouple file localization and file replication

## □ See the discussion thread on the P2Prg

<http://www1.ietf.org/mail-archive/web/p2prg/current/msg00816.html>

- No agreement
- This is a complex problem (many applications of P2P from sensor networks to BitTorrent)
- Taxonomy depends on where is your expertise
- The one I proposed is still weak

# Outline

## □ Overview

- What is a P2P application?
- Popularity of P2P applications?

## □ Content Replication

## □ BitTorrent

## □ Security

## □ Localization

# Measurement of P2P traffic

- ❑ Hard to perform P2P traffic measurements
- ❑ Known techniques [3,4]
  - Port detection
    - Can be easily circumvented
  - Layer 7 inspection
    - Do not work with encrypted payload
    - Database of signatures hard to maintain
  - Heuristic
    - Hard to identify, similar to other applications

# Cache Logic Measurement Study

- ❑ Performed by Cache Logic [1]
- ❑ Business
  - Traffic management
  - Specialized in P2P traffic
- ❑ Customers
  - Internet Service Provider (ISP) and Telecommunications sectors.
- ❑ Technique
  - Deep packet inspection

# Critical analysis

- Customers
  - No enterprise traffic
  - But, it is a huge portion of the traffic
- Deep packet inspection
  - Port level and layer 7 inspection
  - But, how accurate the database of signature is?
- Other studies give also P2P as the dominant traffic of the Internet

# Major results

## □ End of 2004

- BitTorrent is dominating the Internet traffic
  - 30% of the internet traffic!
- Shift of demand from music to movies
- Major sources of torrent file discontinued due to legal actions (e.g., Suprnova.org)

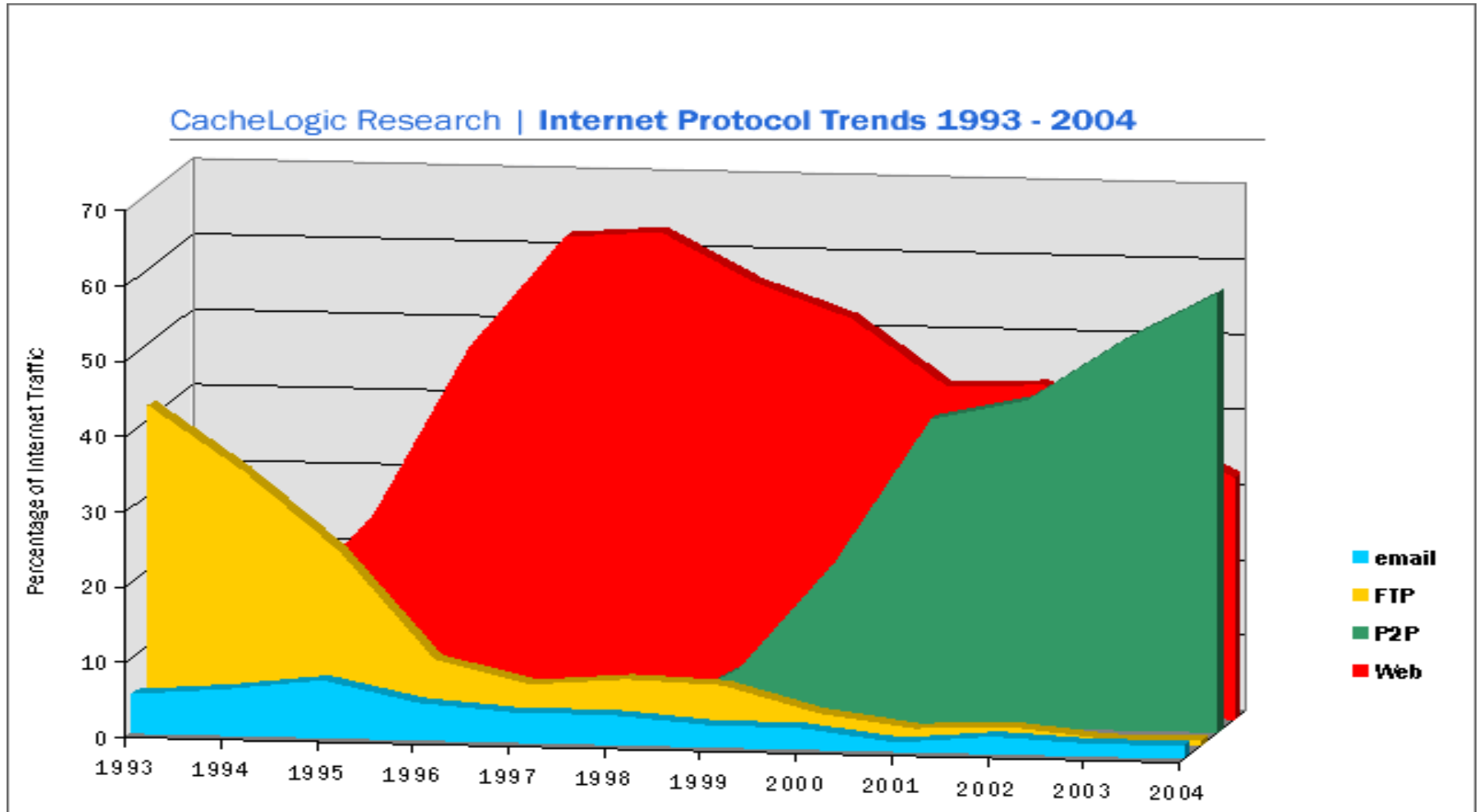


# Major results

## □ 2005

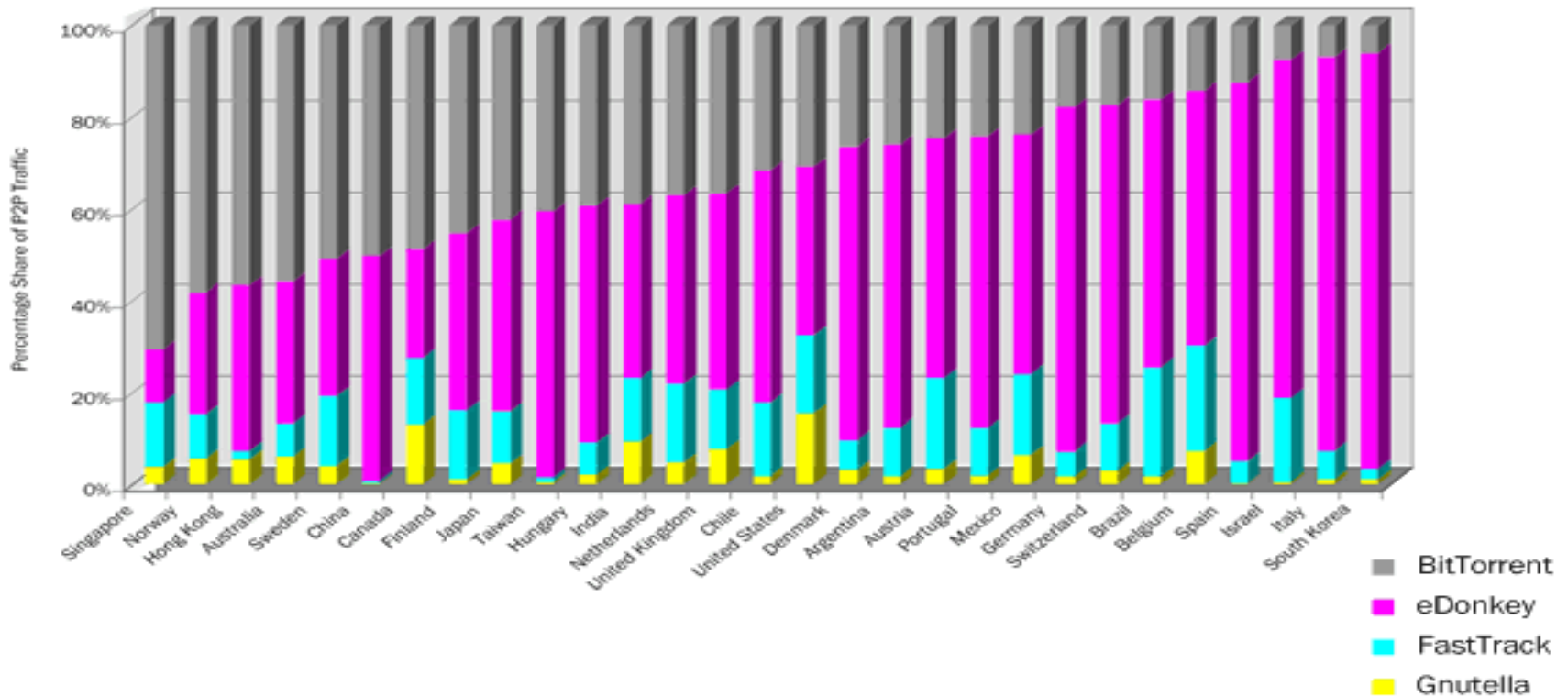
- Shift from BitTorrent to eDonkey
  - eDonkey is fully decentralized
  - Many clients and localization
- But, this is not the end of the story
  - BitTorrent heavily used for legal contents
  - Decentralized versions of BitTorrent
  - New large BitTorrent services (ThePirateBay, mininova, isohunt, etc.)

# Dominant Traffic with Time



# P2P applications per Country

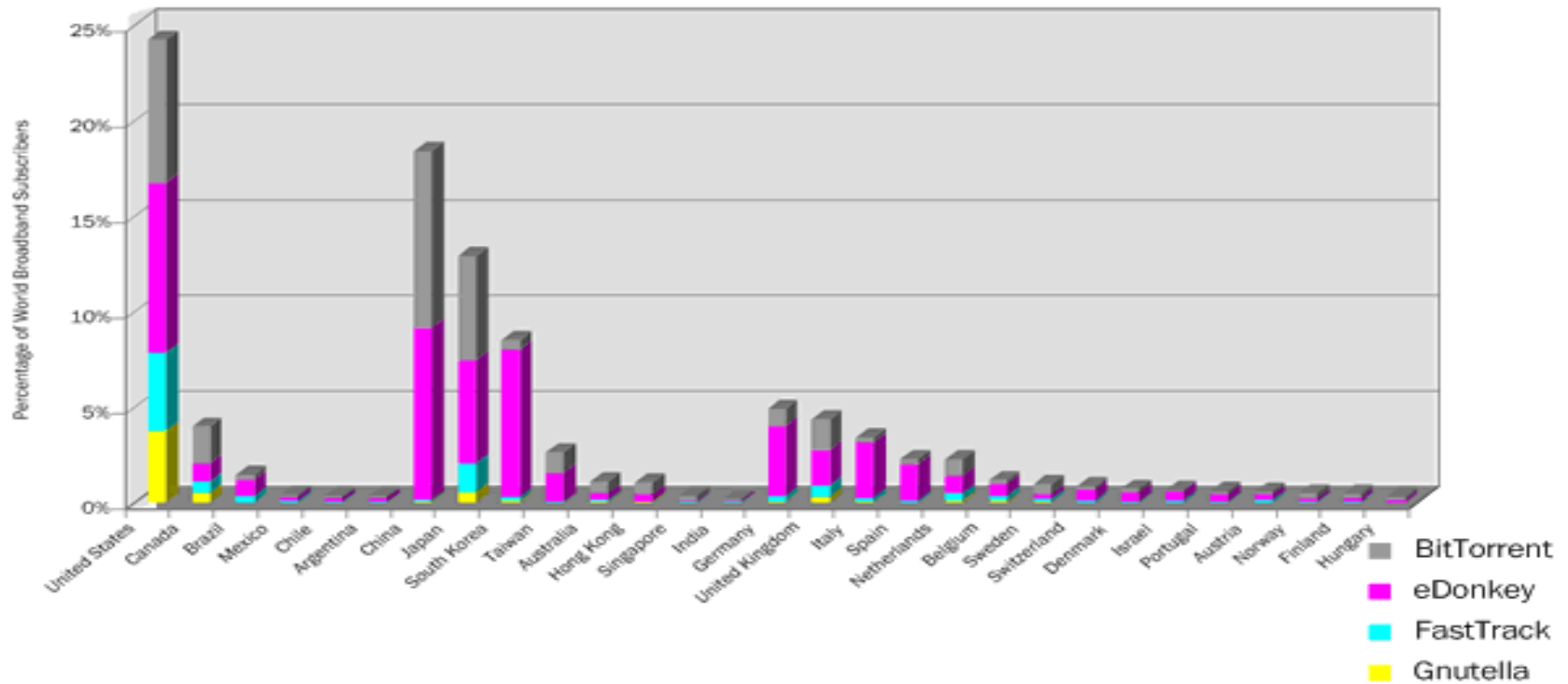
CacheLogic Research | P2P Market Share by Country



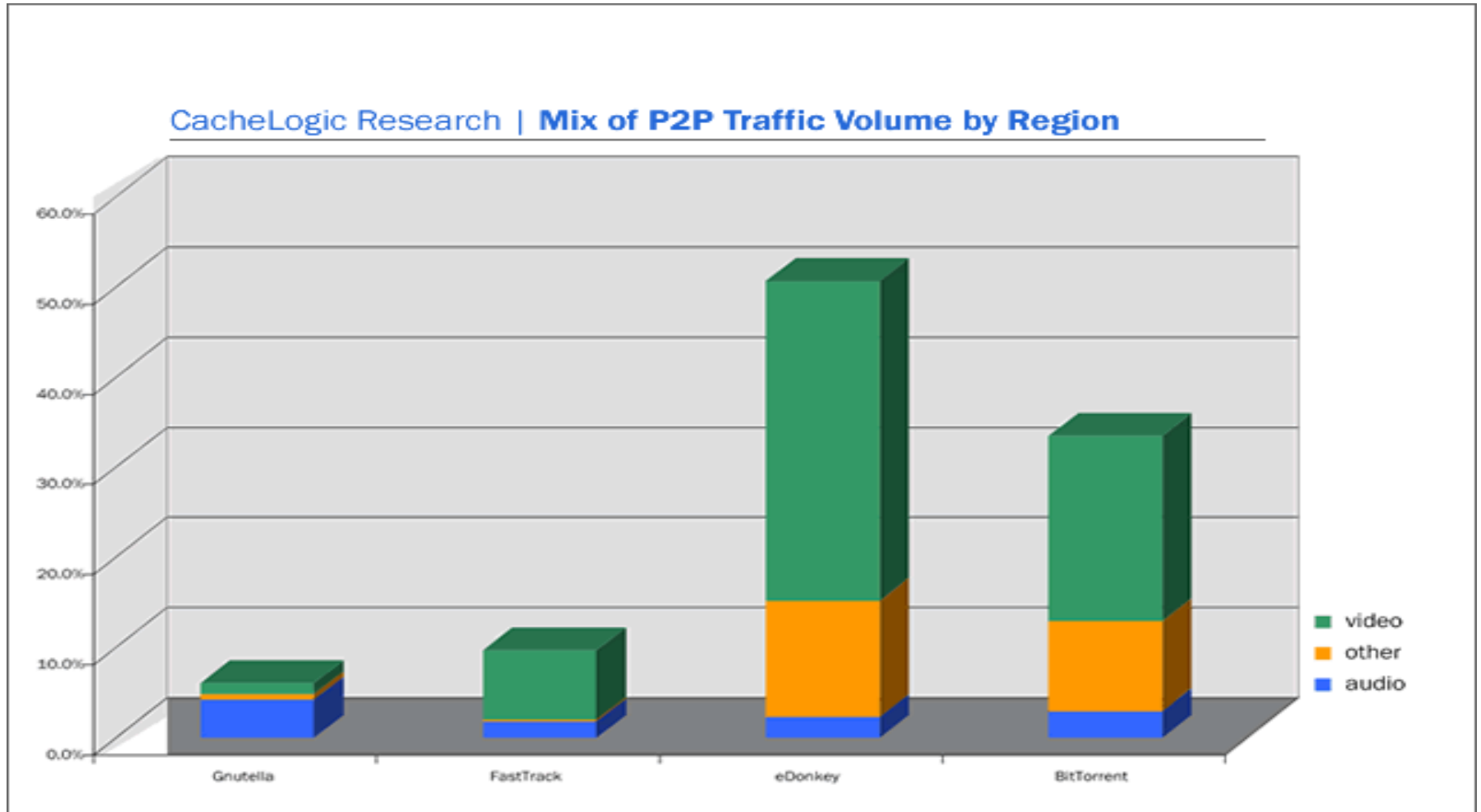
# P2P applications per Country

## CacheLogic Research | Mix of P2P Traffic Volume by Region

Weighted by number of Broadband Subscribers per region



# Type of Contents



# More Recent Results

- Unofficial numbers from FT (September 2006)
  - 80% to 90% of P2P traffic
    - Only 10% to 20% of P2P traffic due to BitTorrent
    - The rest is due to Emule
  - Traffic captured at a BAS
    - Aggregate several DSLAMs (thousands of peers)
    - Claimed layer 7 inspection, but no access to the data and methodology
- 78% of P2P traffic in Japan in 2008 [41]

# More Recent Results

## □ Sandvine (Fall 2010)

- BitTorrent is dominating P2P
  - Except in Latin America where it is Ares
  - But Ares implements BitTorrent, thus it is not clear how much Ares traffic is BitTorrent
- P2P is dominating upstream traffic
- Real-time entertainment (audio and video streaming) is dominating the downstream traffic
  - But, P2P share has significantly increased since 2009 (doubled in some regions)

# Lessons Learned From the Past

- ❑ Specific events might significantly impact popularity of P2P protocols
  - Disconnection of popular services
    - Supernova, Mininova, ThePirateBay
  - Specific laws
    - 3-strikes, lawsuits
- ❑ This has always been a transient impact



# Why to Study P2P (Old Version)

P2P represents most of the Internet traffic

Yes, but it is for illegal contents. P2P applications are evil

Don't you think there is a need for such a service

Yes, but people should pay for the service and we need to keep control on it

And, in this case which techno will you use to reach millions of users without huge distribution costs

P2P

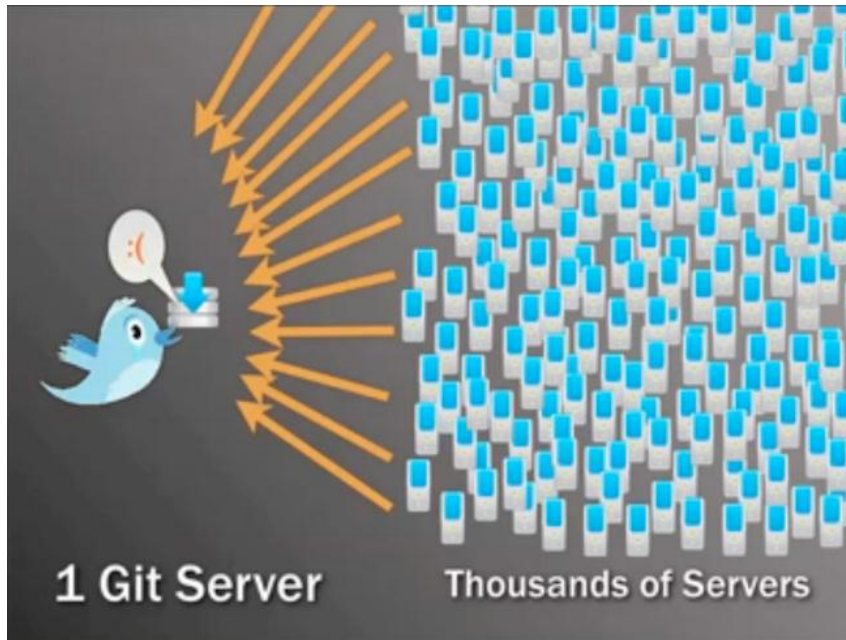


# Why to Study P2P (New Version)

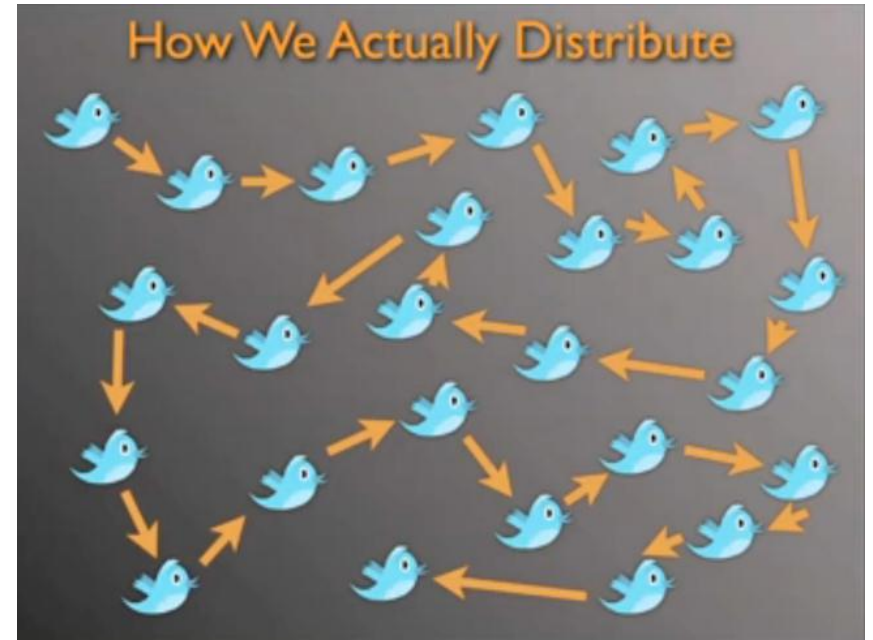
- ❑ BiTorrent is super fast to distribute contents
  - Start to be used by several big companies
- ❑ Twitter is using Murder to update Twitter servers (July 2010)
  - 75x faster
  - <http://engineering.twitter.com/2010/07/murder-fast-datacenter-code-deploys.html>

# Murder

□ Without Murder

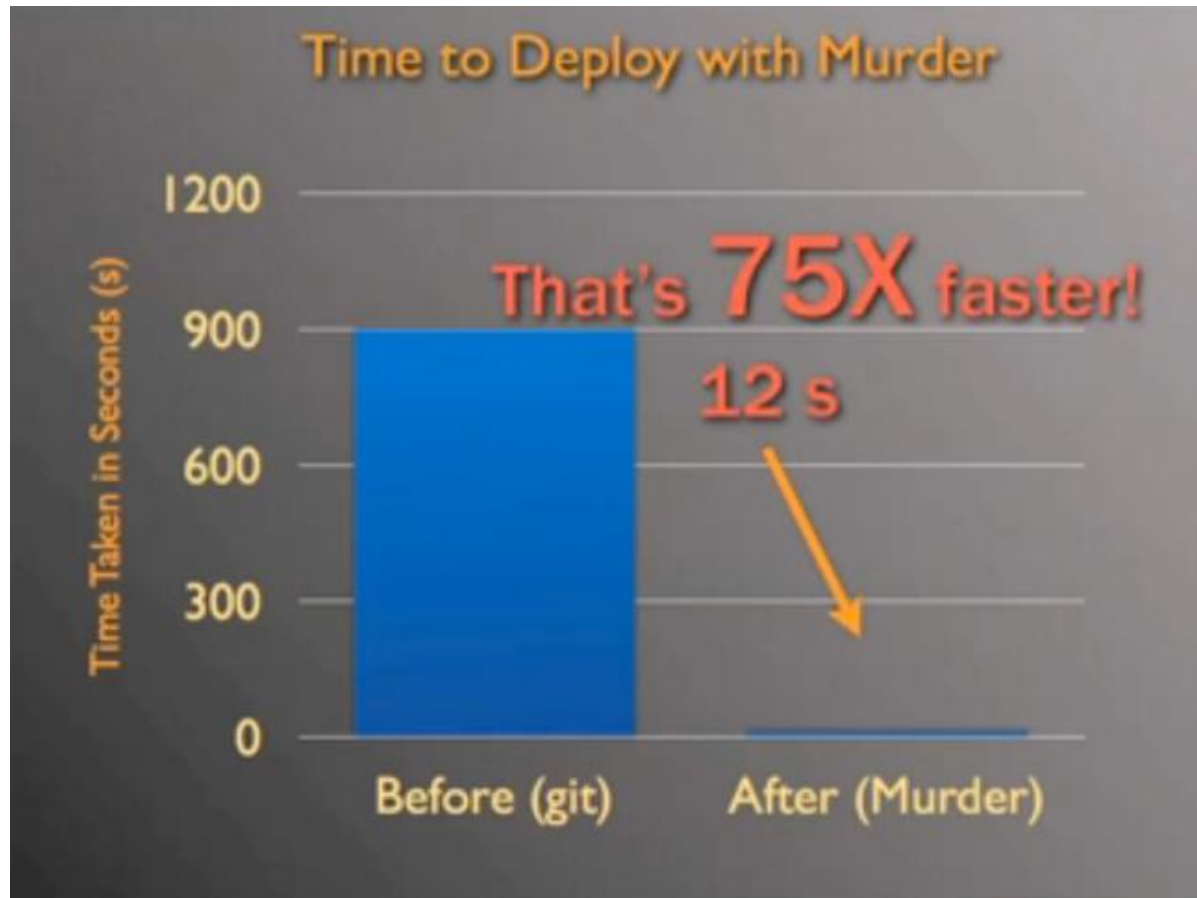


□ With Murder



Credit: Larry Gadea

# Murder Performance



Credit: Larry Gadea

# P2P in the Research Community

- P2P is no more a hot topic
  - What is hot in the community is not a well balanced choice, but close to a fashion decision
  - It is very hard to publish **classical** P2P in major conferences
- However
  - Privacy issues in P2P is still a hot topic

# Outline

- Overview
- Content Replication
  - P2P performance
  - Parallel Download
  - Piece and Peer selection
- BitTorrent
- Security
- Localization

# Definitions

- Service capacity
  - Number of peers that can serve a content
  - It is 1 for client-server, constant with time
- Flash crowd of  $n$ 
  - Simultaneous request of  $n$  peers (e.g., soccer match, availability of a patch, etc.)
- Piece (a.k.a. chunk, block)
  - A content is split in pieces
  - Each piece can be independently downloaded

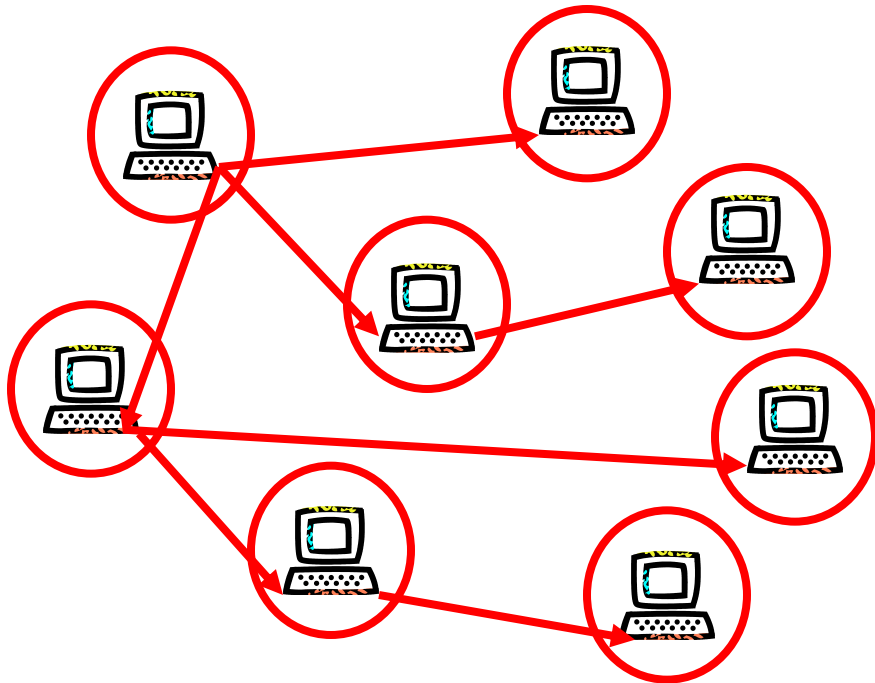
# Why P2P is so efficient?

- The service capacity grows exponentially with time
- With a flash crowd of  $n$  peers, the mean download time is in  $\log(n)$ 
  - It is in  $n$  for a client server model
- The mean download time decreases in  $1/(\# \text{ of pieces})$  when the  $\#$  of pieces increases
  - Do not take into account the overhead

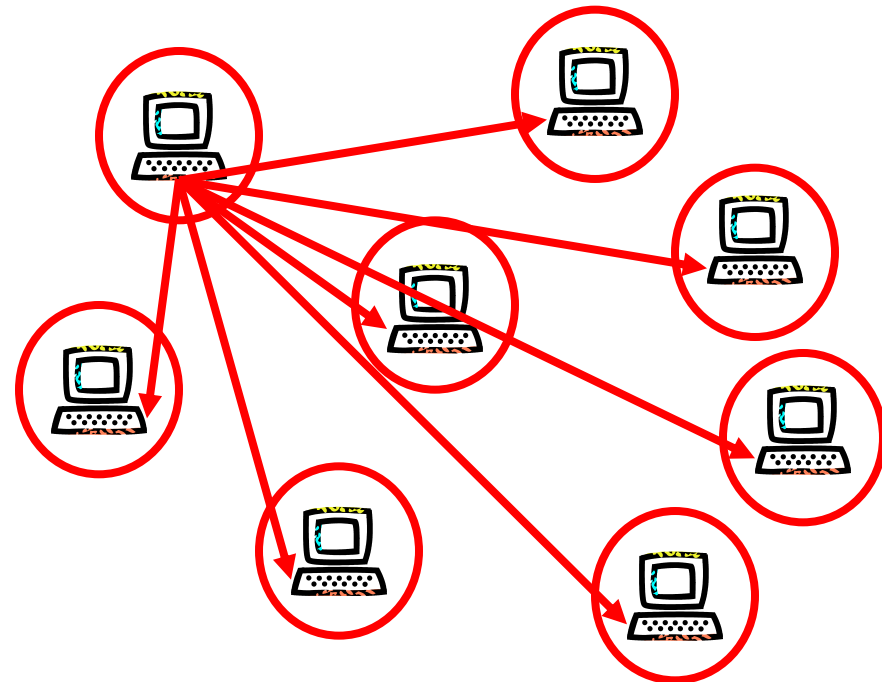


# Intuition

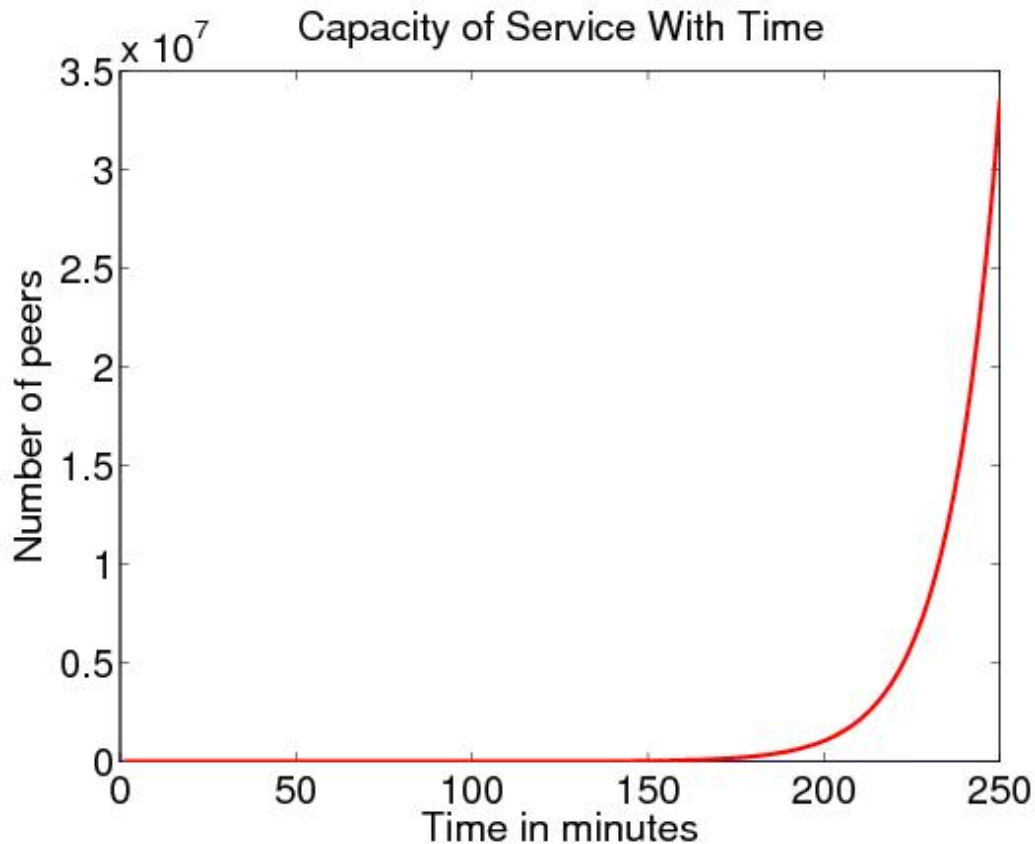
□ P2P



□ Client-server



# P2P vs. Client-Server



## □ P2P

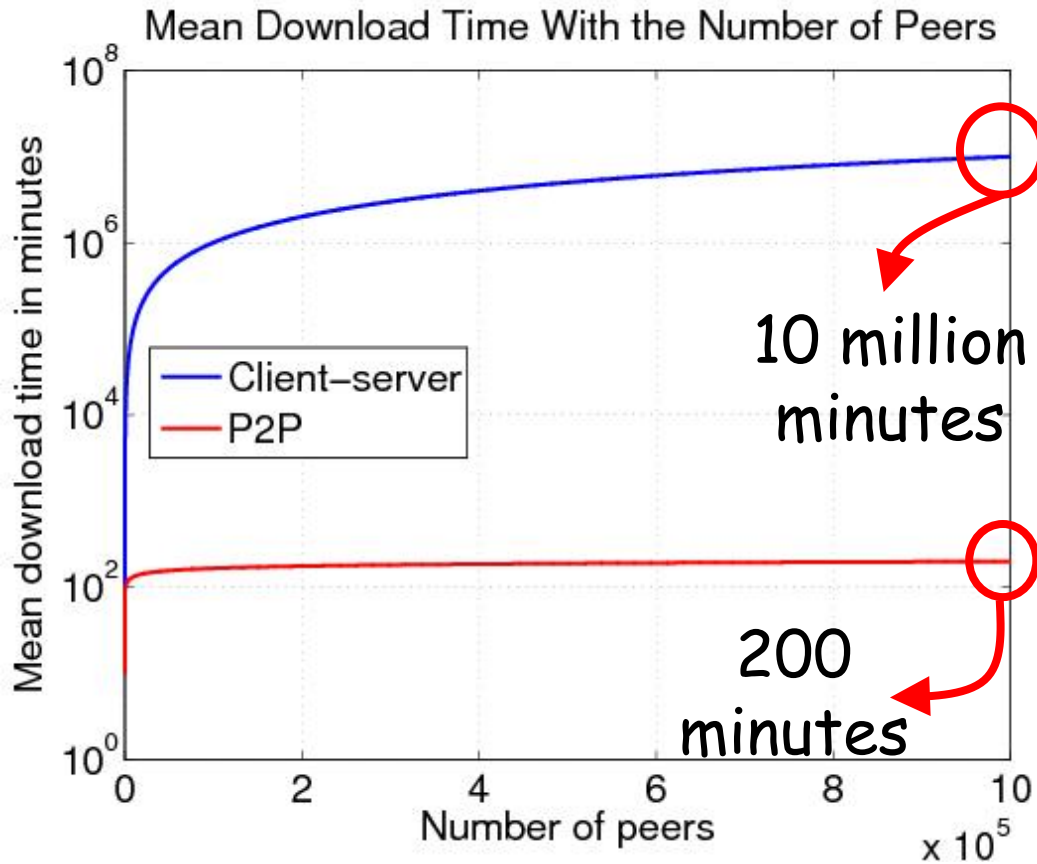
- Capacity of service  $C(t) = O(e^t)$ , where  $t$  is time

## □ Client-server

- Capacity of service  $C(t) = 1$ , where  $t$  is time

## □ Time to serve a content: 10 minutes

# P2P vs. Client-Server



## □ P2P

- Download completion time  $D(n) = O(\log(n))$ , when  $n$  is the number of peers

## □ Client-server

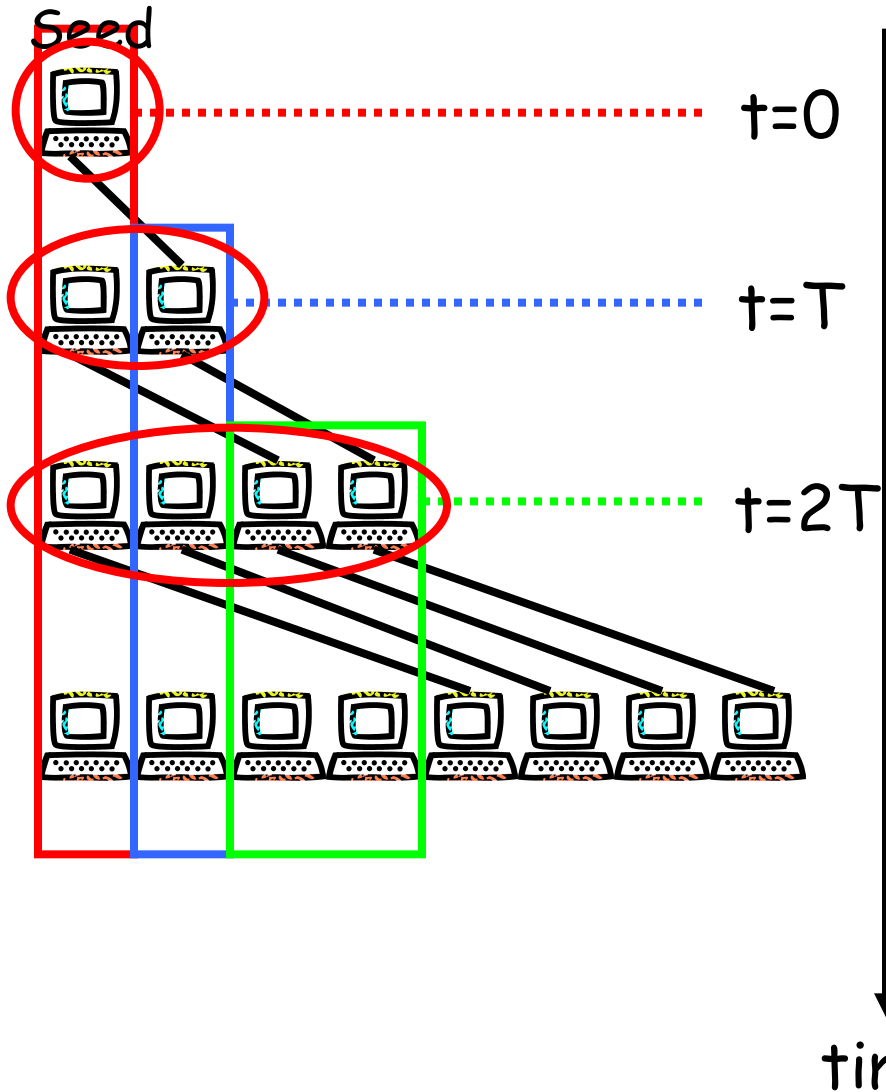
- Download completion time  $D(n) = n$ , where  $n$  is the number of client

- Time to serve a content: 10 minutes

# Content Transfer Model

- Simple deterministic model [5] (to read)
  - Each peer serves only one peer at a time
  - The unit of transfer is the content
  - $n-1$  peers want the content
  - We assume  $n=2^k$
  - $T$  is the time to complete an upload
    - $T=s/b$ ,  $s$  content size,  $b$  upload capacity
  - Peer selection strategy
    - Easy with global knowledge: Binary tree

# Proof: Capacity

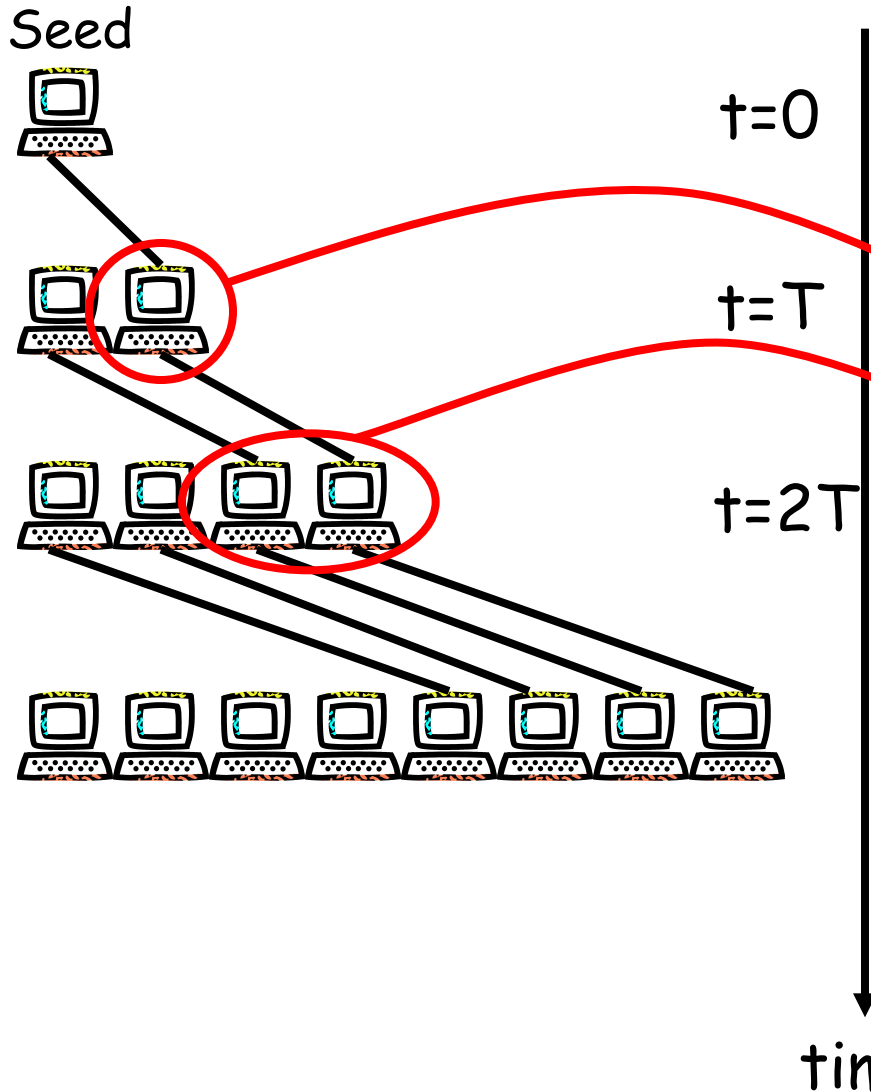


□ Capacity of service  $C$

- $t=0, C=2^0$  peers
- $t=T, C=2^1$  peers
- $t=2T, C=2^2$  peers
- ...
- $t=iT, C=2^i$  peers

▪  $C=2^{t/T}$  peers

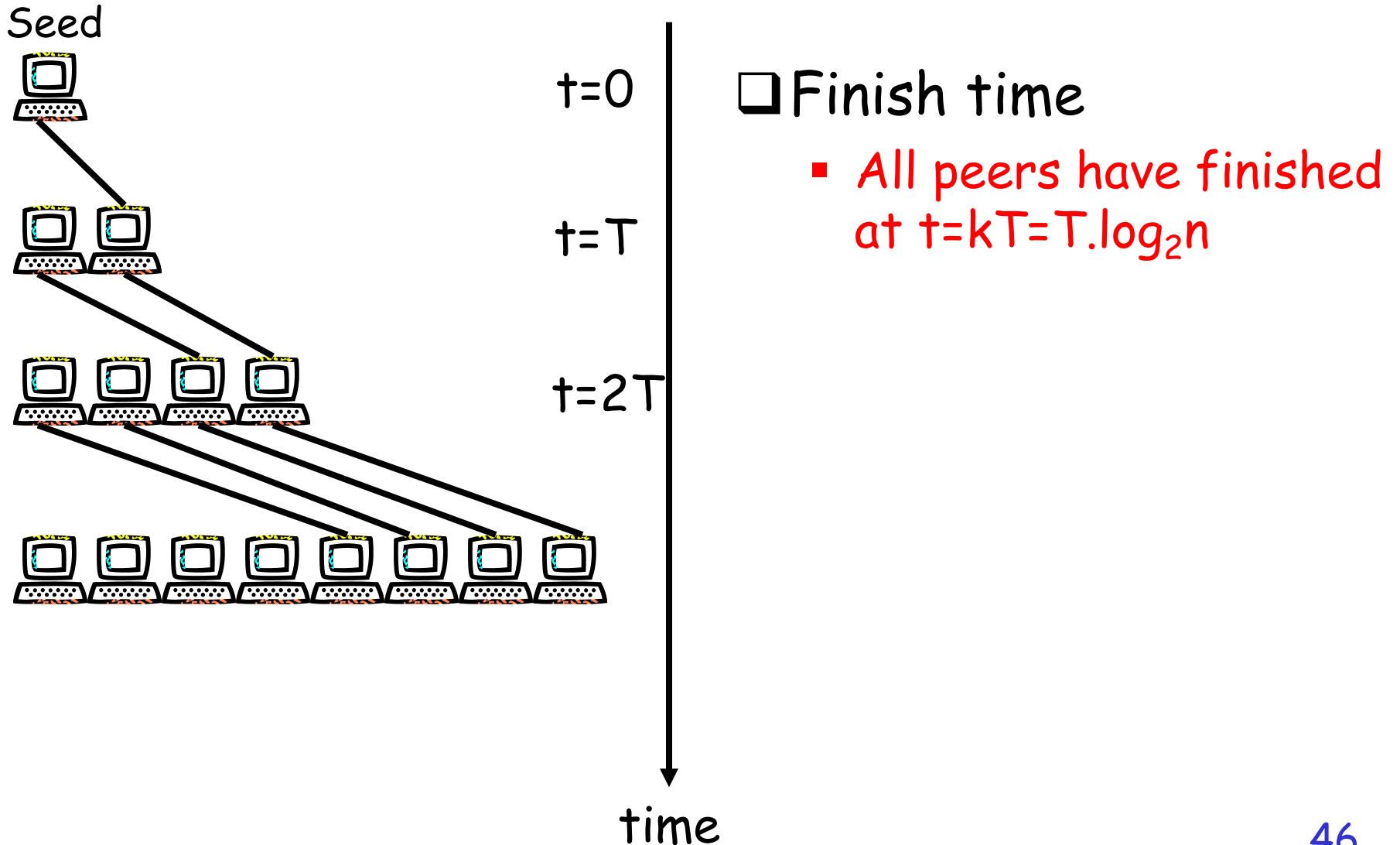
# Proof: Finish Time



## □ Finish time

- Seed has the content at  $t=0$
- $2^0$  peers finish at  $t=T$
- $2^1$  peers finish at  $t=2T$
- ...
- $2^{k-1}$  peers finish at  $t=kT$
- We covered the  $n$  peers
  - $1 + 2^0 + 2^1 + 2^2 + \dots + 2^{k-1} = 2^k$   
 $= n$

# Proof: Finish Time



# Proof: Finish Time

## □ Mean download time

$$\bar{d} = \frac{1}{n} (1 \cdot (0) + 2^0 \cdot T + 2^1 \cdot 2T + 2^2 \cdot 3T + \dots + 2^{k-1} \cdot kT)$$

$$\bar{d} = \frac{1}{n} \sum_{i=0}^{k-1} 2^i (i+1) T = \frac{T}{n} \sum_{i=0}^{k-1} 2^i (i+1) = \frac{T}{n} S_{k-1} \quad \text{See refresher}$$

$$\bar{d} = \frac{T}{n} [(k-1) \cdot 2^k + 1] = T \left[ k-1 + \frac{1}{n} \right]$$

$$\bar{d} = T \left( \log_2 n + \frac{1-n}{n} \right)$$



# Refresher

$$S_{k-1} = \sum_{i=0}^{k-1} 2^i (i+1) = 1.2^0 + 2.2^1 + 3.2^2 + \dots + k.2^{k-1}$$

$$S_{k-1} - 2.S_{k-1} = 2^0 + 2^1 + 2^2 + \dots + 2^{k-1} - k.2^k$$

$$-S_{k-1} = \frac{2^k - 1}{2 - 1} - k.2^k = n - 1 - k.n$$

$$S_{k-1} = n(k-1) + 1$$

# Model Discussion

- ❑ Each peer has the same upload capacity
- ❑ No network bottleneck
- ❑ Idealized peer selection strategy
  - Each peer always knows to which peer  $P$  to send the content at a given time
    - This peer  $P$  does not have the content yet
    - This peer  $P$  is not chosen by any other peer
  - Conflict resolution solved with global knowledge
  - No peer dynamics, i.e., arrival and departure
- ❑ No piece selection strategy
- ❑ No advanced hypothesis: reciprocation, parallel download, etc.

# Piece Transfer Model

- Piece based deterministic model [5] (2004)
  - Each peer serves only one peer per time slot
  - The unit of transfer is a piece
  - $n-1$  peers want the content
  - We assume  $n=2^k$
  - There are  $m$  pieces of the same size
  - We assume  $m \gg k$
  - $S=s/(bm)=T/m$  is a time slot,  $s$  content size,  $b$  upload capacity

# Piece Transfer Model

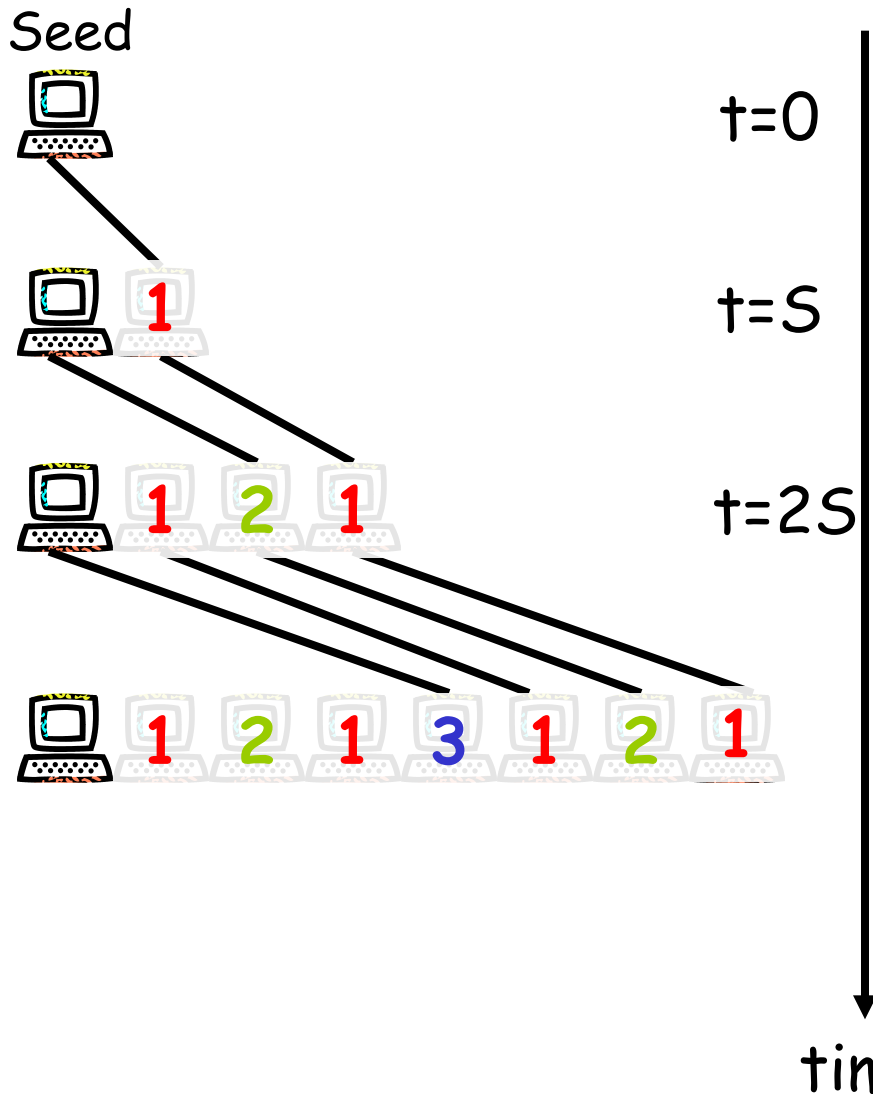
- Peer selection strategy (used in the model)
  - We define  $A_i$  as the set of peers that have piece  $i$ . We do not count in this set the seed
  - Two strategies
  - First strategy, when at least one peer has no piece
    - Peers send pieces to peers that has not yet obtained any piece.

# Piece Transfer Model

## □ Peer selection strategy

- Second strategy, when all peers have at least one piece
  - The set of peers  $A_i$  with  $n/2$  copies of  $i$  replicate  $i$  on the  $n/2-1$  other peers. The  $n/2-1$  other peers and the seed replicate pieces not present on the peers of  $A_i$ .
  - For instance, at  $k+1$ ,  $A_1$  replicate 1 on all the  $A_i$ ,  $i=2, \dots, k$  and the  $A_i$ ,  $i=2, \dots, k$  plus the seed replicate a piece on  $A_1$
  - At each round one peer of  $A_i$  is idle

# First Peer Selection Strategy



□ At  $t=0$

- Seed has all pieces

□ At  $t=S$

- $|A_1|=2^0$

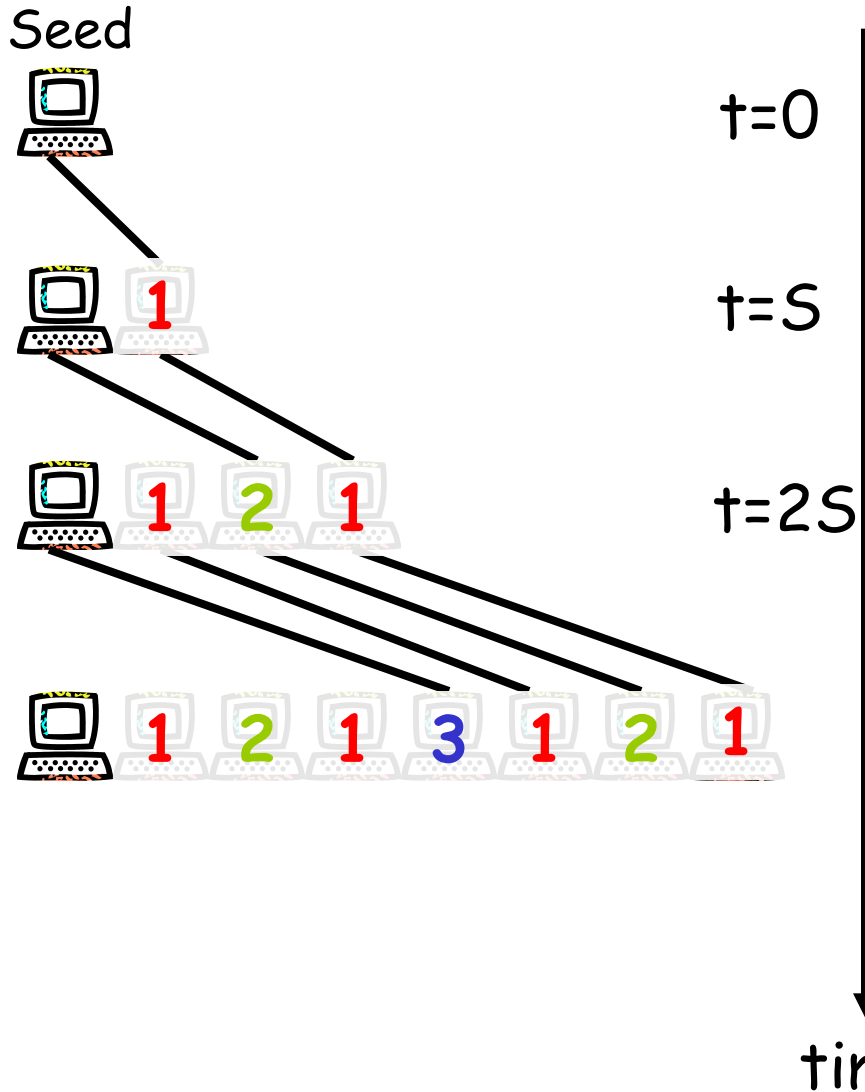
□ At  $t=2S$

- $|A_1|=2^1, |A_2|=2^0$

□ At  $t=3S$

- $|A_1|=2^2, |A_2|=2^1, |A_3|=2^0$

# First Peer Selection Strategy



□ At  $t=jS$

▪  $|A_i| = 2^{j-i}, i \leq j$

□ This strategy ends when  $j=k$

▪ All  $n-1=2^k-1$  leechers have a piece

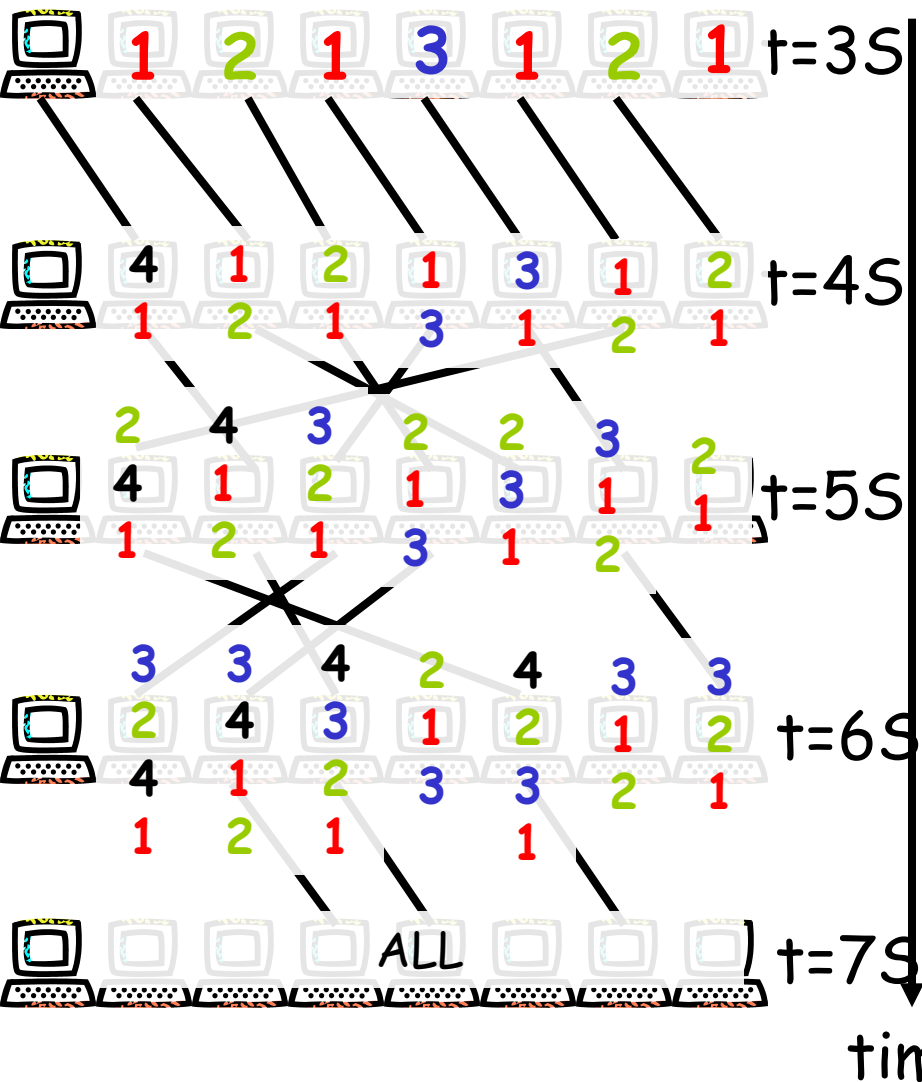
$$\begin{aligned} \sum_{i=1}^k |A_i| &= |A_1| + \dots + |A_{k-1}| + |A_k| \\ &= 2^{k-1} + \dots + 2^1 + 2^0 \\ &= n - 1 \end{aligned}$$

# Second Peer Selection Strategy

- We take as example  $m=4$  and  $k=3$
- In [5] they assume that the seed stops sending pieces when a copy of the content was served
  - Easier to model
  - Lower bound of the performance, because it uses less resources

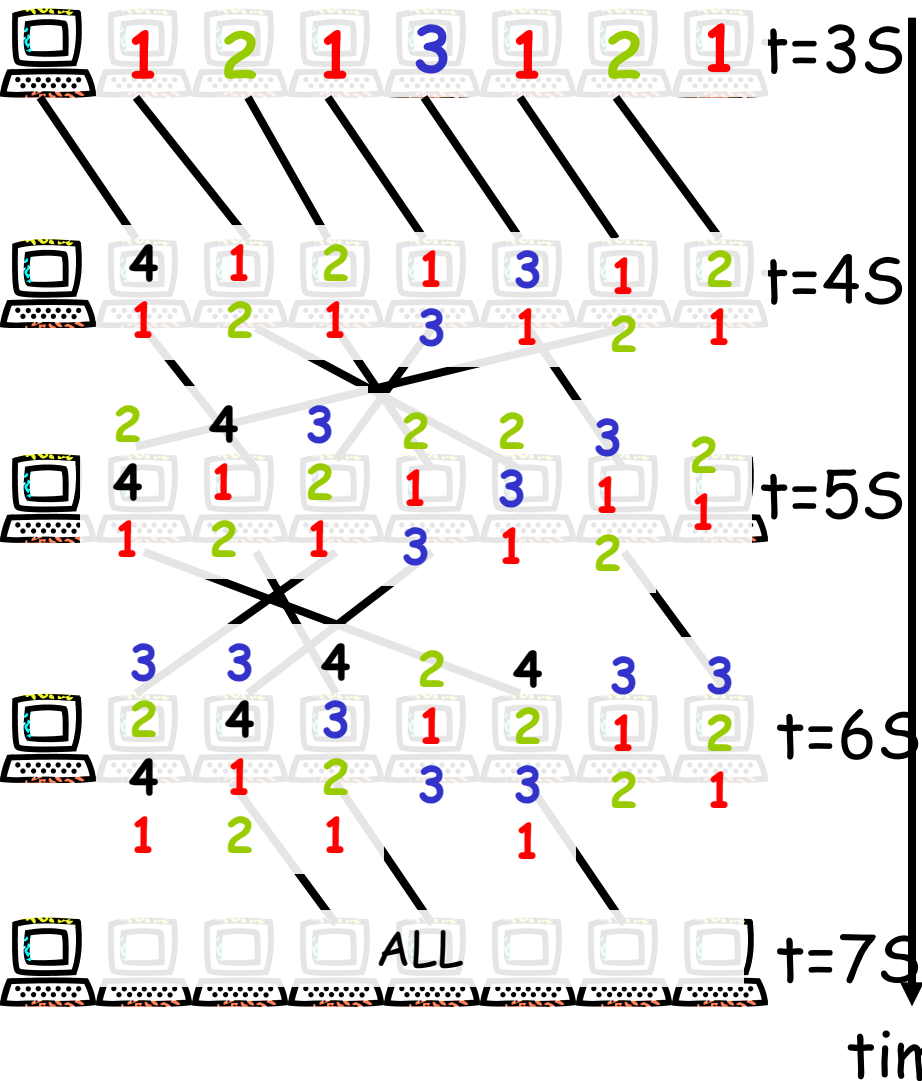


# Second Peer Selection Strategy



- We confirm that for  $k=3$  all peers have a piece
- $t=3S$ 
  - There are  $2^3/2$  piece 1
  - There are  $2^3/2^2$  piece 2
  - There are  $2^3/2^3$  piece 3
- $t=4S$ 
  - All have piece 1
  - There are  $2^3/2$  piece 2
  - There are  $2^3/2^2$  piece 3
  - There are  $2^3/2^3$  piece 4

# Second Peer Selection Strategy



□  $t=5S$

- All have piece 1 and 2
- There are  $2^3/2$  piece 3
- There are  $2^3/2^2$  piece 4

□  $t=6S$

- All have piece 1, 2, and 3
- There are  $2^3/2$  piece 4

□  $t=7S$

- All have piece 1, 2, 3, and 4

# Results

- At  $t=kS$  each peer has a single piece
  - $|A_i|=2^{k-i}$ ,  $i \leq k$
- At slot  $k+i$  for  $i \leq m$ 
  - Each peer has pieces  $1, \dots, i$
  - $|A_{i+1}|=n/2$  peers have piece  $i+1$  and replicate it on the  $n/2-1$  other peers
    - The seed already has piece  $i+1$
  - Each other peer replicates a piece on the peers in  $A_{i+1}$ 
    - At the  $m$  slot, the seed stops serving pieces
    - For all  $j > i+1$ ,  $|A_j| \leftarrow 2 * |A_j|$

# Results

## □ Finished time

- At each slot the number of copy of each piece is doubled
- When there are  $n=2^k$  peers, a piece needs  $k+1$  slots to be on all peers
  - We consider that the first slot for piece  $x$  is when  $x$  is sent by the seed to the first peer
- For  $m$  pieces,  $k+m$  slots are required to distribute all pieces on all peers

# Results

## □ Finished time

- All peers have finished at  $t=(k+m)S$
- $t=(k+m)S=T(k+m)/m=(T/m).\log_2 n + T$
- Decreases in  $1/m$  compared to the content based model
- Does not account for pieces overhead

# Results

## □ Mean download time

- With the proposed strategy, at  $kS$  each peer has only one piece
- As the number of pieces double at each slot, one needs  $k+m-1$  slots for half of the peers to have all the pieces
  - At  $k$ , 1 piece; at  $k+1$ , 2 pieces; at  $k+m-1$ ,  $m$  pieces
  - But at  $m$ , the seed stops serving pieces, thus at  $k+m-1$  only half of the peers have  $m$  pieces, the rest have  $m-1$  pieces
- The other half receives the last pieces at  $k+m$

# Results

□ Mean download time

$$\bar{d} = \frac{S}{2} [(k + m - 1) + (k + m)]$$

$$\bar{d} = \frac{T}{m} \left( \log_2 n + m - \frac{1}{2} \right)$$

$$\bar{d} = \frac{T}{m} \log_2 n + T \left( 1 - \frac{1}{2m} \right)$$

# Model Discussion

- ❑ Each peer has the same upload capacity
- ❑ No network bottleneck
- ❑ **Idealized peer selection strategy**
  - Each peer always knows to which peer  $P$  to send the content at a given time
  - Conflict resolution solved with global knowledge
  - No peer dynamics, i.e., arrival and departure
- ❑ **Idealized piece selection strategy**
  - Global knowledge
  - A peer is never blocked because it does not have the right piece to upload
- ❑ No advanced hypothesis: reciprocation, parallel download, etc.
- ❑ Read [5,6] for a more sophisticated models



# Model Discussion

- The results obtained with this model hold for more complex models
  - Stochastic
- Lesson
  - A simple model can give fundamental results
  - Understand the assumptions and limitations
  - No need for complexity if it is at the price of stronger or additional assumptions

# Discussion of the Results

- P2P is very efficient when
  - There is always a peer to send data to
  - There is always a piece to send to this peer
- Peer and piece selection are at the core of an efficient P2P protocol
  - P2P efficiency can be from the idealized model to even worse than client-server
- How to select peers and pieces discussed in the following

# Outline

□ Overview

□ Content Replication

- P2P performance
- Dynamic Parallel Download
- Piece and Peer selection

□ BitTorrent

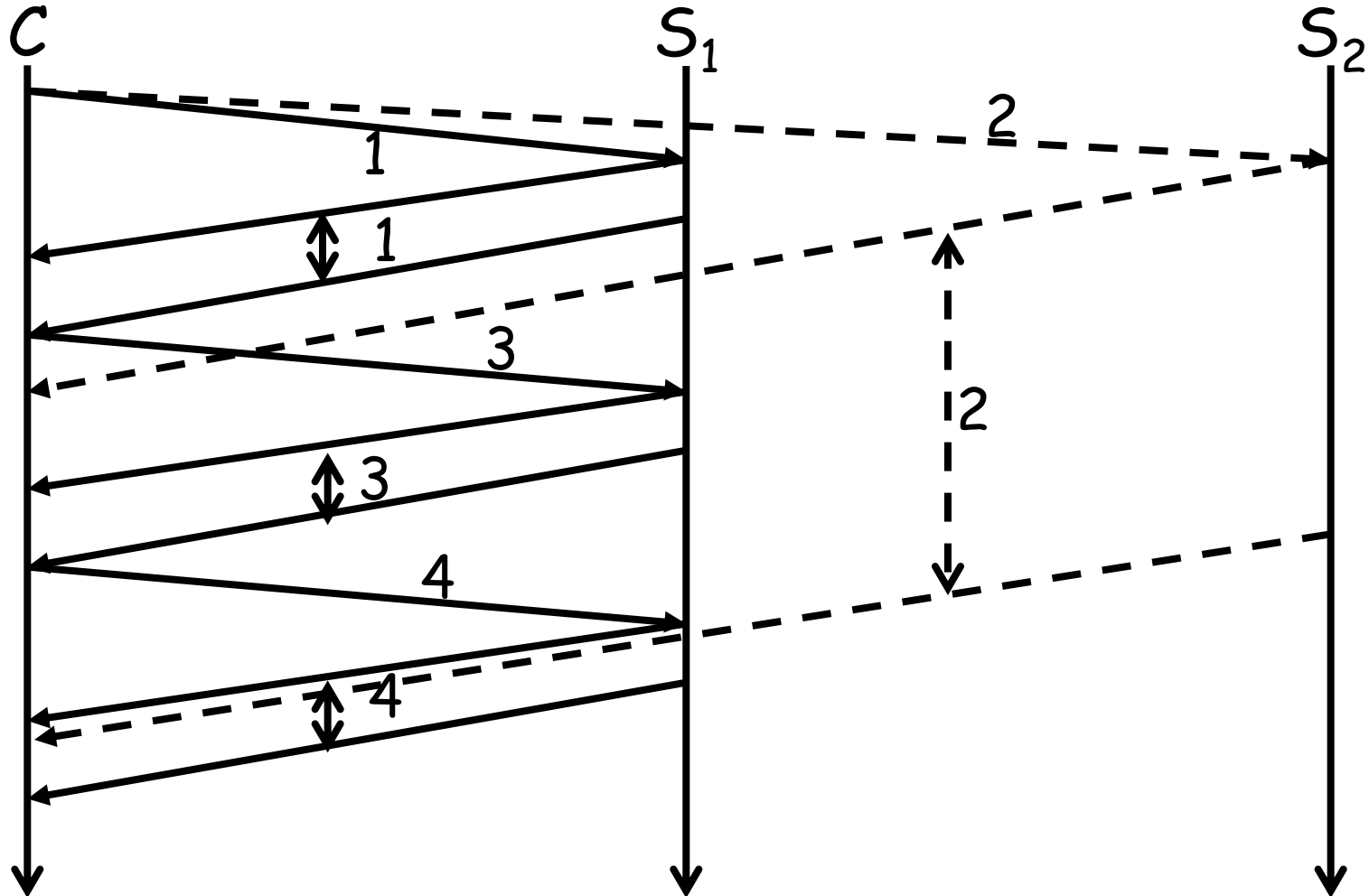
□ Security

□ Localization

# Dynamic Parallel Download

- ❑ Introduced by Rodriguez et al. [8] (2000) in the context of web cache
- ❑ Parallel download
  - The principle to download from several server in parallel
- ❑ Dynamic parallel download
  - A parallel download with the following strategy
  - Strategy
    - Request first one piece from every server with the content
    - Each time a server has completed its upload of a piece, request a piece from this server that has not yet been requested from any other server

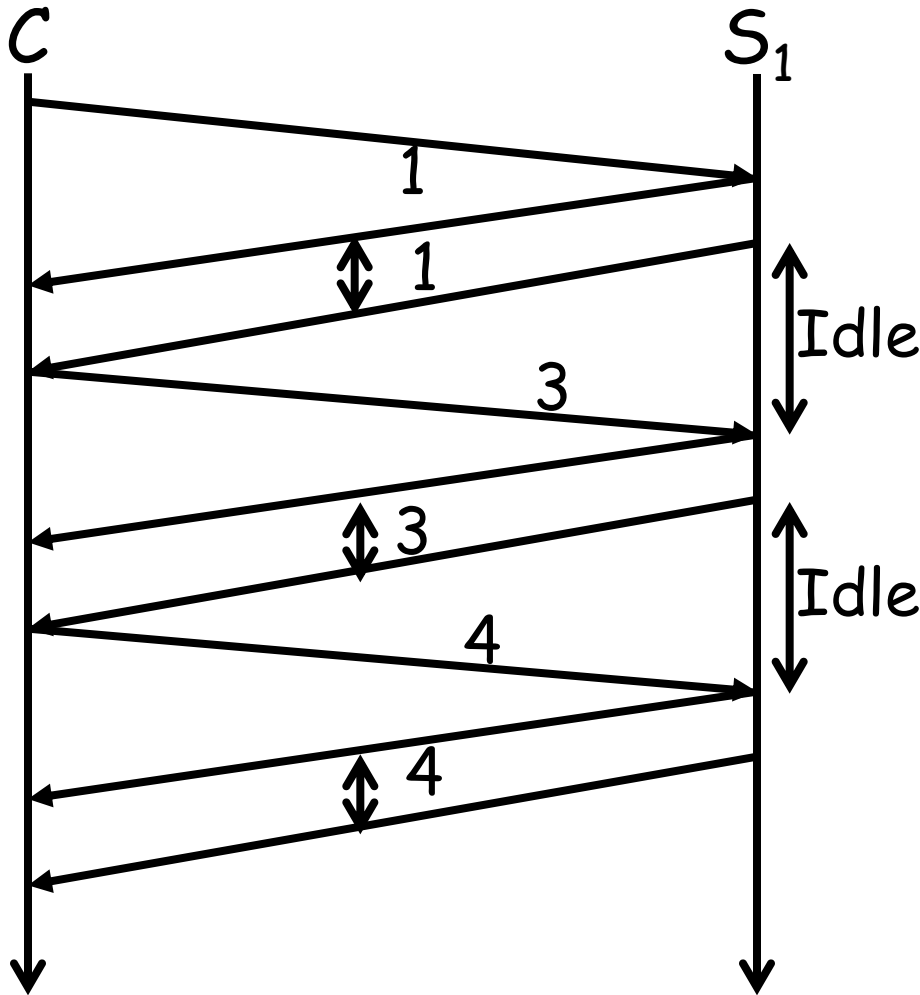
# Dynamic Parallel Download: 4 pieces example



# Performance Issues

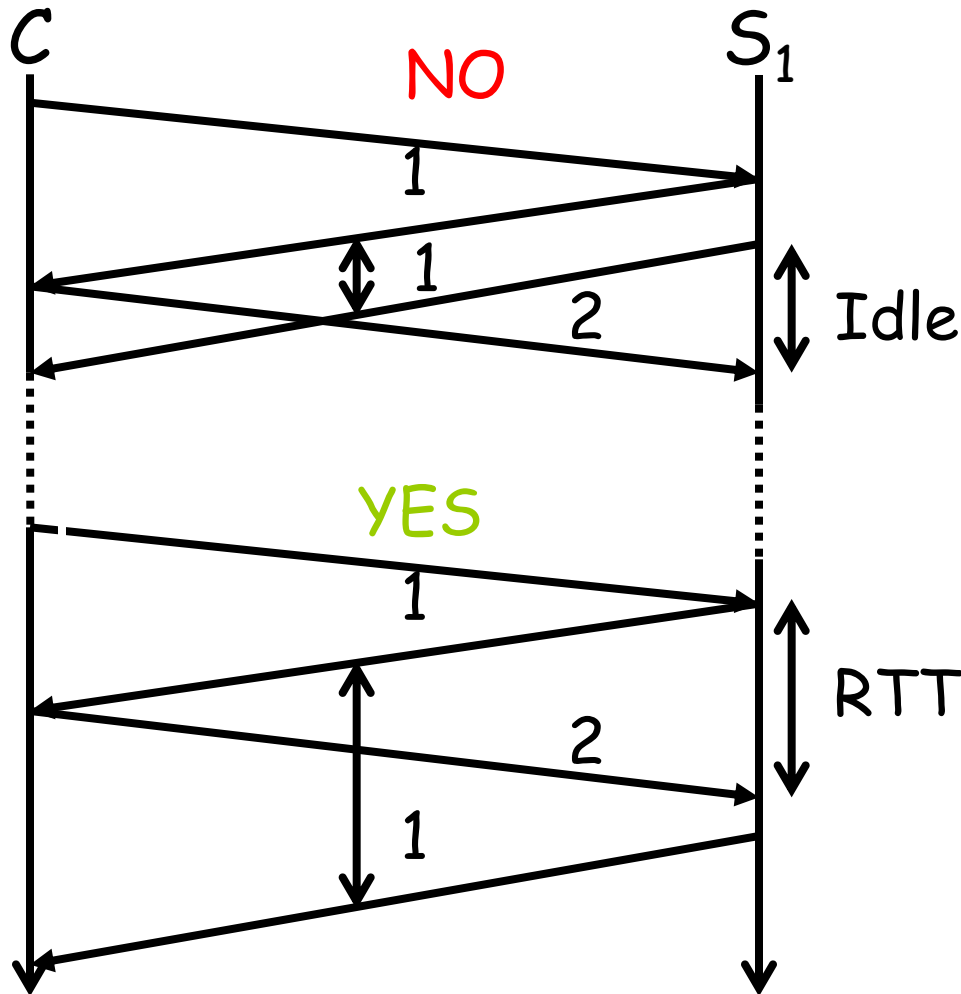
- ❑ All servers must be busy sending pieces
- ❑ Two performance issues
  - Interblock idle time
    - Pipelining
  - Termination idle time
    - End game mode (Terminology introduced in BitTorrent)

# Interblock Idle Time



- ❑ Time to receive a new request after sending the last byte of a piece
- ❑ Idle time = 1 RTT
- ❑ Problem
  - Server underutilized
- ❑ Solution
  - Pipelining

# Pipelining



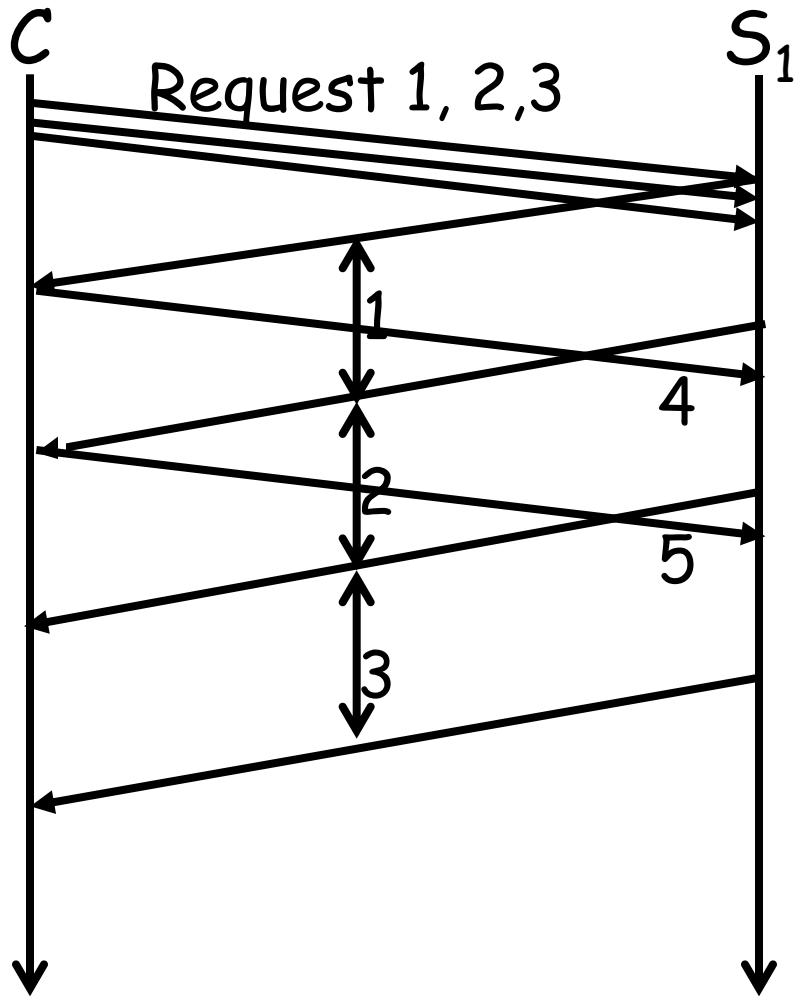
❑ Keep enough requests pending so that the server is never idle

❑ 1<sup>st</sup> solution

- Send request before the end of the current piece
- RTT estimate
- Piece transmission time > RTT



# Pipelining



## □ 2<sup>nd</sup> solution

- Always have  $n$  pending requests
- Still need RTT estimate
  - No need for accuracy
  - Overestimate does not harm

# Termination Idle Time

- ❑ For dynamic parallel download from  $M$  servers
- ❑  $P$  is the number of pieces not yet received
- ❑ When  $P < M$ ,  $M - P$  servers are idle
- ❑ Solution: end game mode
  - When  $P < M$  request pending blocks to all the idle servers
  - Several servers upload the same piece at the same time
    - The fastest win
  - Bandwidth waste: request + partial download

# Termination Idle Time

- ❑ Without end game mode
  - Last pieces download speed unknown
- ❑ With end game mode
  - Last pieces download speed equal to at least the one of the fastest server

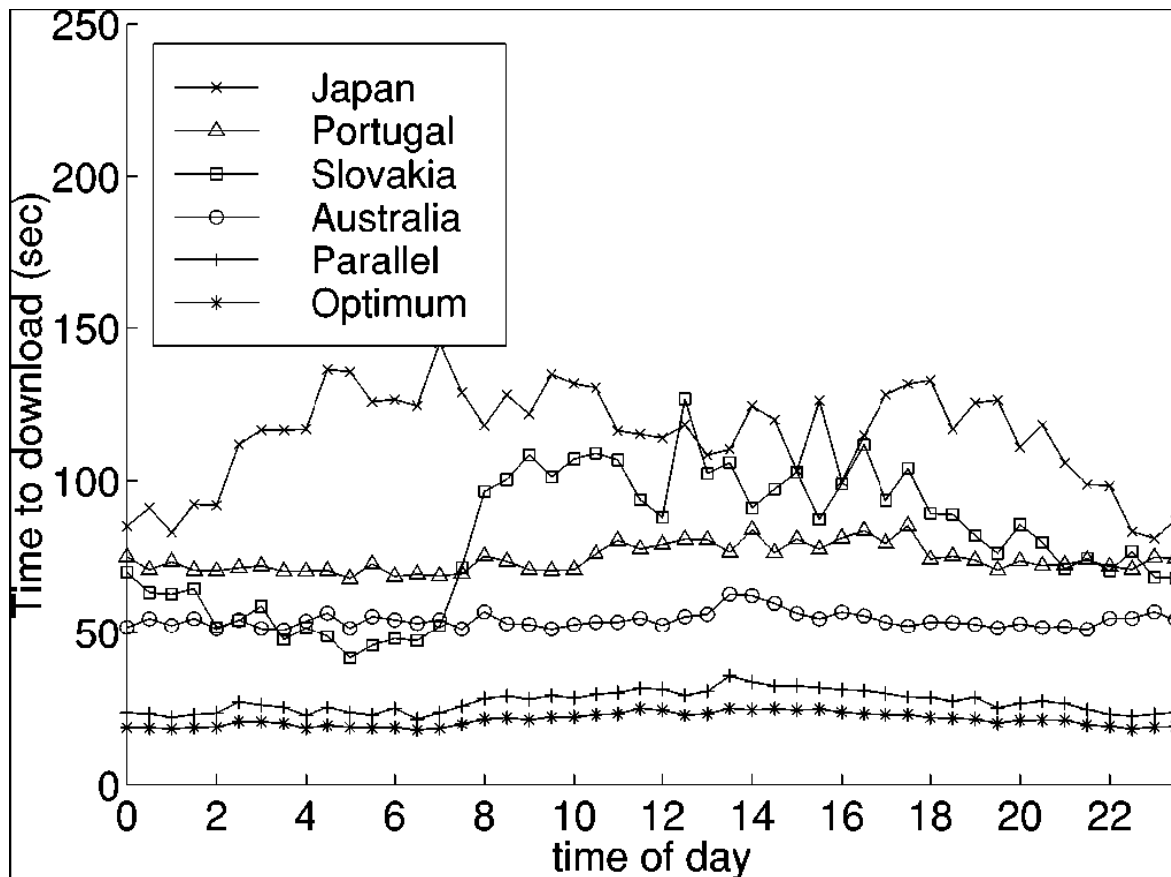
# Experimental Evaluation

- Java client that implements dynamic parallel download
  - Does not implement pipelining
  - Implement a basic version of end game mode
- Connect to real mirror of public web servers in the Internet
- Study performed in 1999/2000
- For each figure is given the optimum transmission time
  - Ideal download time that would have been achieved in case there is neither interblock nor termination idle time (computed *a posteriori*)

# No Shared Bottleneck

- The client connects to 4 mirrors spread in the Internet: Japan, Portugal, Slovakia, Australia
  - High probability of disjoint paths, which implies no shared bottleneck

# Results: No Shared Bottleneck



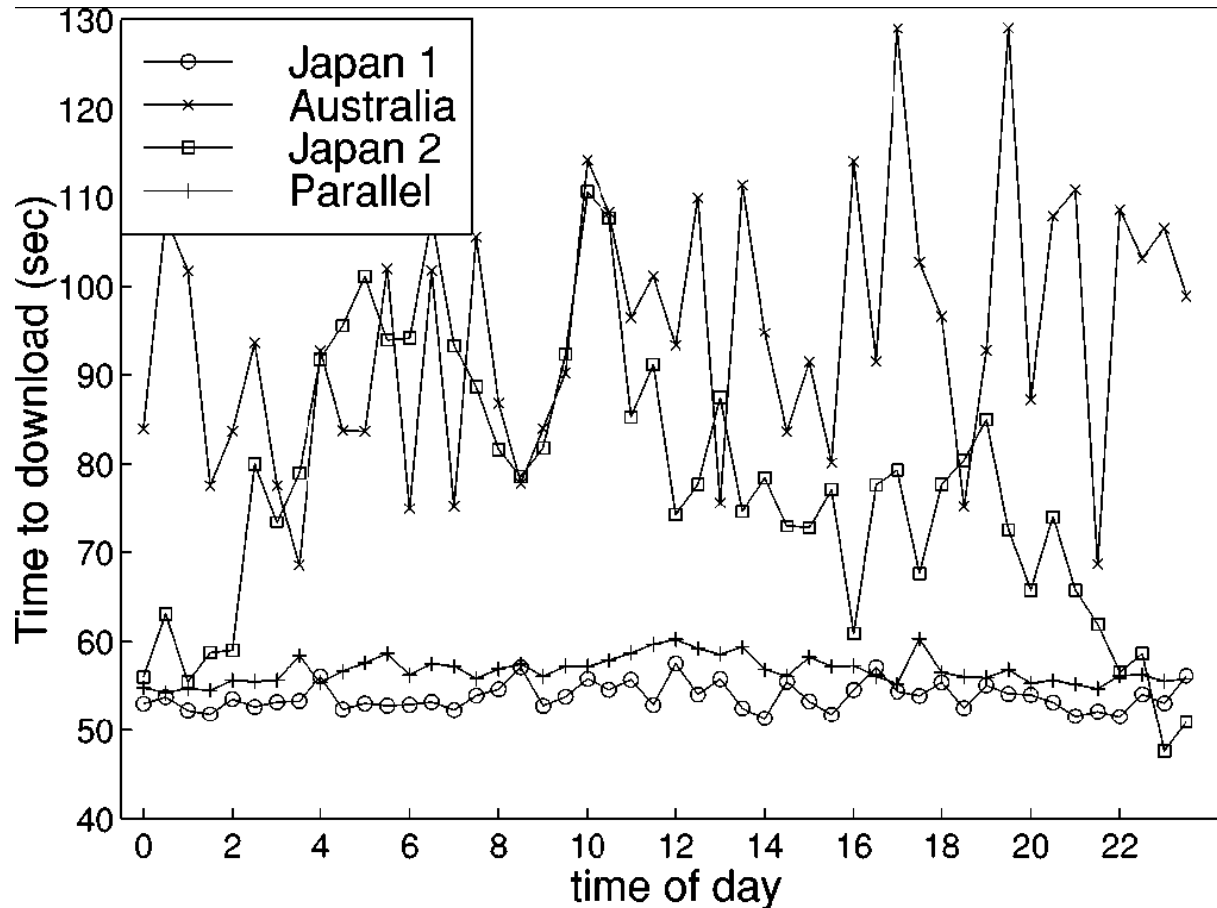
- Content size
  - 763KB
- # of pieces
  - 80
- Parallel
  - 4
- No shared bottleneck
- Parallel close to optimum

Credit: Rodriguez et al. [8]

# Shared Bottleneck

- ❑ What happens when the bottleneck is the access link?
- ❑ The client is connected through a modem link (56kbit/s)
  - Connected to two slow servers (24kbit/s) and one fast server (56kbit/s)
- ❑ The fastest server is enough to saturate the access link
  - Dynamic parallel download will create TCP competition on a saturated link. What is the impact of that?

# Results: Shared Bottleneck



- Content size
  - 256KB
- # of pieces
  - 20
- Parallel
  - 3
- Modem access line
  - Shared Bottleneck
- Close to the fastest server
  - Difference due to the interblock idle time

Credit: Rodriguez et al. [8]

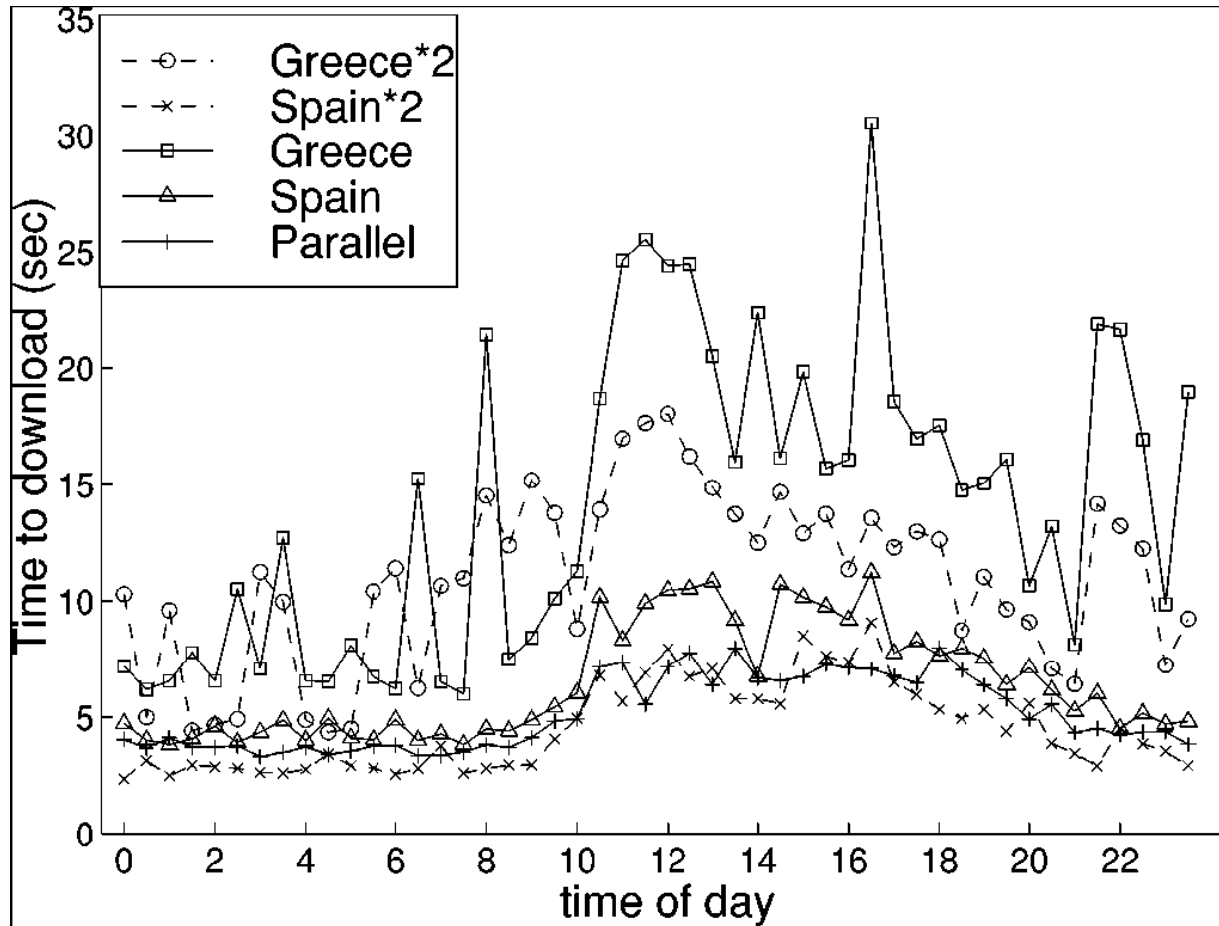


# Single Server vs. Multiple Servers

## Parallel Download

- ❑ Is it as efficient to perform parallel download from a single server as from multiple servers?
- ❑ The case of two mirrors: 1 fast and one slow
  - The client connects to a single mirror
  - The client connects to two mirrors in parallel
  - The client opens two TCP connections to the same mirror

# Results: Single Server Parallel



- Content size
  - 256KB
- # of pieces
  - 20
- Parallel
  - 2
- No shared Bottleneck
- Close to the fastest server, but no need for server selection

Credit: Rodriguez et al. [8]

# Properties

- ❑ Automatically adapt to the best servers and bottlenecks
  - No need for server selection
- ❑ No complex resource discovery
  - History based parallel access performs significantly worse
- ❑ Tradeoff
  - Piece request message overhead
  - Increase the number of TCP connections

# Dynamic Parallel Download for P2P

- Dynamic parallel download
  - In the context of client-server
  - For a small number of parallel downloads
- P2P
  - Every peer is a client and a server
    - Parallel download and parallel upload
  - Large peer set
- Very different context
  - How to apply dynamic parallel download to P2P?

# Dynamic Parallel Download for P2P

- A straightforward application to P2P
  - Every peer performs global dynamic parallel download to every other peer
- Problems
  - Not possible to maintain a large number of TCP connections per peer
  - Why a peer should send data to another peer?
    - Not viable: free rider problem

# Dynamic Parallel Download for P2P

## □ Free rider problem

- A free rider is a peer that downloads without contributing anything
- To scale, each peer in a P2P system must act as a client and a server
- With global dynamic parallel download no incentive to do so
- We do not leave in an ideal word: selfish assumption

# Dynamic Parallel Download for P2P

- Assume an ideal world
  - Each peer cooperate
  - Can we use dynamic parallel download?
- Studies on dynamic parallel upload
  - In P2P the content flow is from the initial seed toward leechers
  - Easier to model dynamic parallel upload than dynamic parallel download
  - Equivalent properties

# Dynamic Parallel Download for P2P

- Dynamic parallel upload vs. download
  - Download
    - The client want to download as fast as possible
  - Upload
    - The source want to upload as fast as possible
  - Same problem
    - Find the fastest peer among a set without any knowledge



# Dynamic Parallel Download for P2P

## □ Outdegree

- Number of parallel uploads from a peer

## □ Tradeoff

- Increasing the outdegree increases the number of peers served at the same time, but decreases the service rate to each peer [5][9]

# Dynamic Parallel Download for P2P

## □ Results

- Biersack et al. [9] showed that an outdegree of 3 is optimal. An outdegree of 2 or 4 gives almost the same result
  - Static scenario
  - Forest of tree
  - Assume uniform capacity of the peers
    - Upload=download
    - Same capacity for all peers

# Dynamic Parallel Download for P2P

## □ Results

- Yang et al. [5] showed that increasing the outdegree adversely impact the service capacity in case of static peers (no leave at the end of the download)
- But, in case of dynamic peers (peers leave the system with a given probability after completing the download) parallel upload can improve the service capacity. Outdegree should be less than 10, marginal gain above 4
- Branching process model (stochastic)

# Conclusion

- ❑ Even in an ideal world the outdegree should be small
  - Around 4
- ❑ This number might be increased in case of high upload capacity, but no study to understand the real impact
  - Probably makes sense in case of heterogeneous peers
    - The fast peer increases its number of parallel uploads to saturate its upload capacity
  - Probably dangerous in case of homogeneous peers
    - All peers increase their number of parallel uploads to saturate their upload capacity. But, in this case the global efficiency decreases as shown in [5,9]
  - Used by BitTorrent mainline if max upload > 42 kB/s
    - uploads =  $\text{int}(\text{math.sqrt}(\text{rate} * .6))$

# Conclusion

- ❑ We are not in an ideal world
  - In case of free riders
    - The system is not viable
    - The analytical results do not hold
    - Dynamic Parallel upload cannot be used
- ❑ How to prevent free riders?

# Outline

□ Overview

□ Content Replication

- P2P performance
- Parallel Download
- Piece and Peer selection

□ BitTorrent

□ Security

□ Localization

# Why a Peer and Piece Selection?

- Lets go back to the simple model
- Assumptions
  - Always find a peer with an interesting piece to upload from (Global knowledge)
    - Never idle or seeking for a peer
  - A peer never refuses to upload a piece
    - No free riders
- If any of these assumptions is relaxed
  - No model for the system
  - No idea of its performance (at least worse)
  - No parallel download (selfish environment)

# Why a Peer and Piece Selection?

- Additional assumptions
  - All peers have the same upload capacity
    - Always the best match
  - No network bottleneck
    - Still the best match
- This is not reality
- If best match relaxed
  - Performance decreases
- But, a good match is still possible in real life



# Which Peer and Piece Selection?

- ❑ No specification except for BitTorrent
  - Always focus on content localization
- ❑ No similar problem in another field
- ❑ No general study of the problem
  - Always based on BitTorrent

# Which Peer and Piece Selection?

## □ Gnutella

- Designed for efficient content localization
- No file splitting in the specification 0.6 [16]
- Partial file transfer introduced in [17]
  - Allows peers with partial content to answer queries
- Same heuristic for piece and peer selection
  - Select the first peer that answers the content request
  - Possibility of parallel download
- Poor overall performance
  - No specific study of the file transfer efficiency
  - Mostly used for small contents (mp3)

# Which Peer and Piece Selection?

## □ Edonkey2000/Emule/Overnet

- Designed for efficient content localization
- Only differ by their localization protocol
- File splitting [13]
  - Rarest pieces first + other criteria with lesser priority
- Peer selection
  - (Time spent in the priority queue) \* (credit modifier based of upload and download rate)
  - Slow reactivity
  - Possibility of parallel download
- Average overall performance
  - No specific study of the file transfer efficiency

# Which Peer and Piece Selection?

- BitTorrent (described in details later in the course)
  - Designed for efficient file transfer
  - File splitting [13]
    - Rarest pieces first
  - Peer selection
    - Choke algorithm based on short term peer upload speed estimation
    - Fast adaptation
    - Use of parallel download
  - Good overall performance
    - Several specific studies

# Which Peer and Piece Selection?

- ❑ Common mistakes made about BitTorrent (BT)
  - With BT contents are hard to find
    - Right, BT is a file transfer protocol not a localization protocol
    - Does it make sense to say that with HTTP contents are hard to find?
  - With BT a torrent die when there is no more seed
    - Right, BT is a file transfer protocol, not an infrastructure that manage persistency
    - Does it make sense to say that HTTP does not guarantee that your web server is always up?
- ❑ BT is a P2P file transfer protocol, nothing more

# Which Peer and Piece Selection?

- In the following, general discussion, but based on the experience gathered with BitTorrent
  - BitTorrent is the state of the art
  - Might be improved, but need a deep understanding

# Which Peer and Piece Selection?

## □ Peer selection task

- Always find a peer to upload from
- Prevent free riders
- Converge to the **best** upload-download match

## □ Piece selection task

- Piece diversity is called **entropy**
- With ideal entropy, each peer always has an interesting piece for any other peer

# Which Peer and Piece Selection?

- Peer selection must not have a piece constraint
  - Ideal entropy is the target
  - Peer selection should be based on capacity only, not on piece availability



# Selection Order

- Performs piece then peer selection
  - Puts a constraint on the peer to select
  - The selected peer is unlikely to be a good match
  - Depending on the piece selection, may create hot spots
    - Rare piece selection, with an **initial seed** with all the pieces
  - Focus on the piece then on the capacity of service

# Selection Order

- Performs peer then piece selection
  - Select first the best match peer
  - Then apply the piece selection on that peer
  - Focus on the capacity of service first
- Peer and piece selections are interlinked
  - To find the best match you may need pieces to download to test if it is the best match
  - No sense to select a peer with no interesting pieces

# Selection Order

- ❑ Peer selection first is a better strategy
  - Important to maximize the capacity of service
  - No study on this order
- ❑ No general reflection on the role of peer and piece selection
  - Results are given as reasonable rules
  - Experience on existing P2P protocols

# Piece Selection

## □ Random piece selection

- Each peer selects at random a piece to download

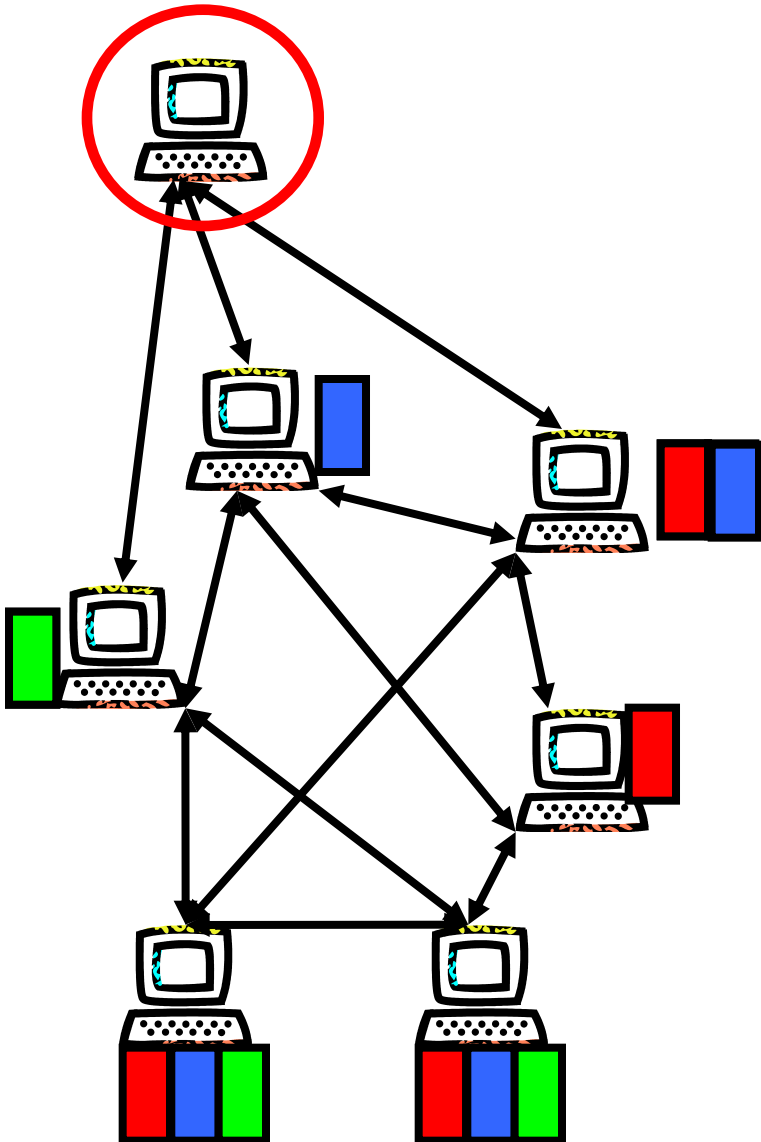
## □ Global rarest first

- Each peer maintains globally the number of copies of each piece
- Select the globally rarest piece to download
- Require global knowledge

# Piece Selection

- Local rarest first (LRF)
  - Approximation of global rarest first
  - Each peer maintains the number of copies in its peer set of each piece
  - Select the locally rarest piece to download
- When peer selection is performed first, rarest first piece selection is applied on the pieces available on the selected peer

# Piece Selection: LRF



- Local rarest first algorithm
  - Choose the pieces that are **locally** rarest

coolContent.xvid



# Piece Selection Properties

- ❑ Random piece selection performs poorly [33]
  - Last pieces problem
  - Poor entropy, i.e., high constraint on peer selection
- ❑ Global rarest first [33]
  - Good entropy
- ❑ Local rarest first [18, 33, 34]
  - Good entropy with a large peer set
  - Care should be taken to the graph construction (random graph)
  - Inspire yourself from BitTorrent

# Piece Encoding

- Do not use raw pieces but encoded pieces
  - To solve the **piece scheduling (selection) problem**
  - Initial seed erasure code [15]
    - $k$  is the number of original pieces
    - $n$  is the number of encoded pieces
    - Any  $k$  among the  $k+n$  pieces are enough to reconstruct the content
    - Still need piece selection
  - Network coding: avalanche [14]
    - Each node computes erasure code
    - Each piece sent is a linear combination of all the already received pieces. Coefficients are chosen at random
    - No more need for piece selection

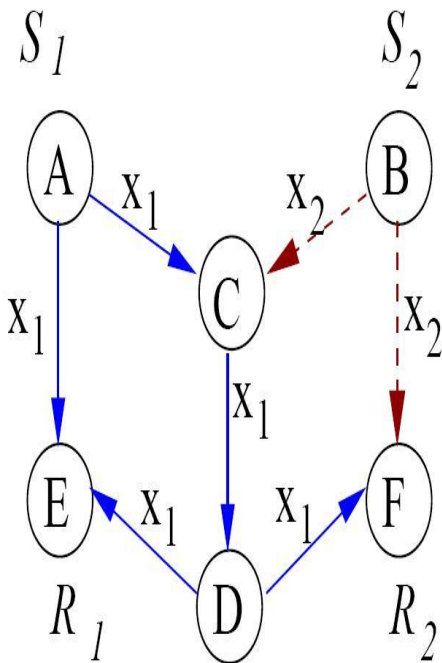


# Network Coding (NC)

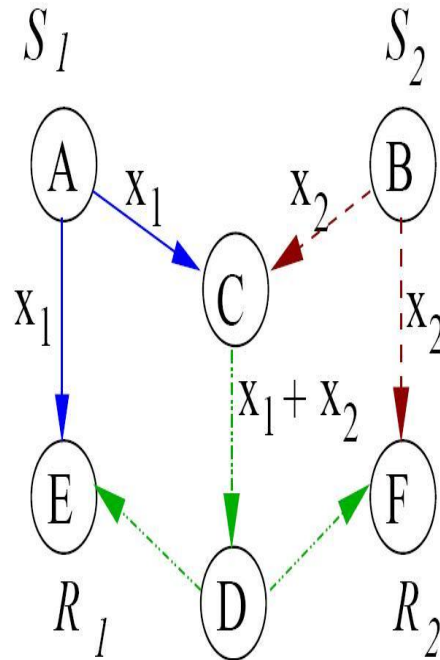
- Theorem[30]: Assume that the source rates are such that without network coding, the network can support each receiver in isolation (i.e., each receiver can decode all sources when it is the only receiver in the network). With an appropriate choice of linear coding coefficients, the network can support all receivers simultaneously

# Network Coding (NC)

*Traditional Method*



*Network Coding*



- ❑ Each link has a capacity of 1
- ❑ When  $R_i$ ,  $i=\{1,2\}$  is alone, it can receive at 2
- ❑ When both  $S_1$  and  $S_2$  broadcast to  $R_1$  and  $R_2$ 
  - Without network coding
    - Both receivers receive at 1.5, e.g.,  $R_1$  receives at 1 from  $S_1$  and at 0.5 from  $S_2$ , while  $R_2$  receives at 1 from  $S_2$  and 0.5 from  $S_1$
  - With network coding
    - Both receivers receive at 2

Credit: Fragouli et al. [30]

# Network Coding (NC)

- For a good general introduction to NC read [30]
- Simple example for a file  $F = [x_1 \ x_2]$ , where  $x_i$  is a piece
  - Define code  $E_i(a_{i,1}, a_{i,2}) = a_{i,1} * x_1 + a_{i,2} * x_2$ , where  $a_{i,1}, a_{i,2}$  are numbers
  - There is an infinite number of  $E_i$ 's
  - Any two linearly independent  $E_i(a_{i,1}, a_{i,2})$  can recover  $[x_1 \ x_2]$
  - Similar as solving a system of linear equations

# Network Coding (NC)

- In practice, coefficients are chosen in a finite field, such as  $GF(2^{16})$ 
  - On a finite field the size of the encoded packets is constant
  - The size of the field gives the overhead
    - On a  $GF(2^{16})$  there are 2 bytes per coefficient
    - For 1000 pieces, there are 2 bytes \* 1000 = 2000 bytes of overhead per packet
    - For a packet size of 256 kB there is a 1% overhead

# Network Coding

- ❑ No more need for piece selection as any encoded piece is useful for anybody
  - Low probability of linearly dependent coefficient
  - Always close to optimal entropy
- ❑ Is it the universal solution?

# Network Coding: Drawbacks

- Heavy computations (each node needs to solve linear systems at each packet received)
  - Authors shown it can run on Pentium 4 desktops with 20% CPU usage [31]
  - Thus, cannot run on PDAs, cell phones, sensors, or desktops that cannot afford this load
- Security issues (a single corrupted block propagate fast among peers)
  - Authors proposed a solution based on secure random checksum [32]
  - But, as for any security issue, needs time to convince and demonstrate

# Network Coding : Drawbacks

## □ Overall complexity

- Authors argue that it was surprisingly easy to implement because there is no piece selection issues
- But, the algorithmic of local rarest first is much simpler than the one of network coding
  - Network coding is harder to assess

# NC vs. LRF

- NC performs well in any situation (at least theoretically)
  - But, no large deployment to validate
  - Computation, security, and complexity
- LRF requires a large peer set
  - 80 is enough and practical for millions of peers
  - Validated on large deployment
  - Close to optimality in practice (details in the following)



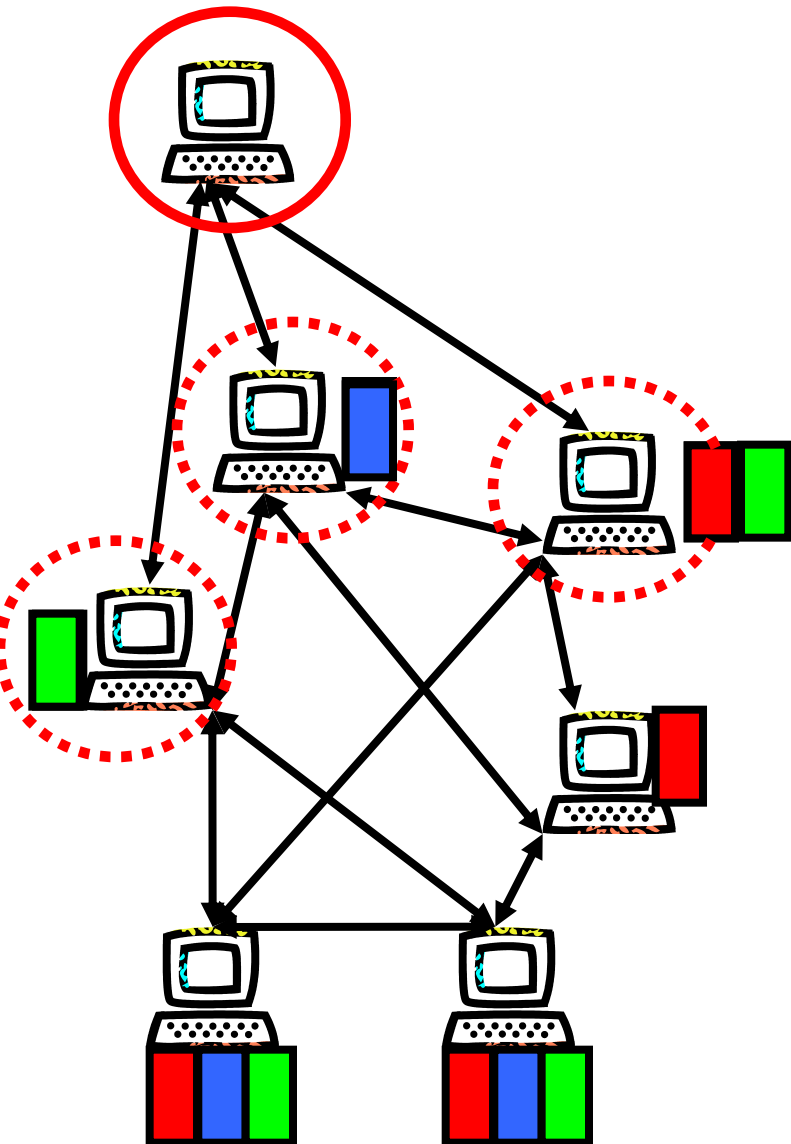
# NC vs. LRF

- ❑ My recommendation (I may be wrong, this is an active and complex research issue)
  - Always use LRF when a large peer set is feasible
- ❑ In some cases a large peer set is not feasible, thus NC can be an appropriate solution in such cases
  - Windows update requires SSL, but a PC machine cannot maintain 80 SSL connections at a time (complex crypto)
  - Ad hoc networks may have limited connectivity when in a sparse environment
- ❑ LRF is experimentally validated for no more than a few 100 000 of peers, not clear how NC will perform

# Peer Selection

- No serious alternative to the BitTorrent Choke algorithm
- **Choke** algorithm [18]
  - Different algorithm in leecher and seed state
  - Peer selection performed in the **peer set**
  - Choke/unchoke
    - A chokes B if A decides to do not upload to B
    - A unchokes B if A decides to upload to B
  - Interested/not interested
    - A is interested in B if B has at least one piece that A does not have
    - A is not interested in B if B has a subset of the pieces A already has

# Peer Interest



- Peer X is interested in peer Y if peer Y has at least 1 piece that peer X does not have

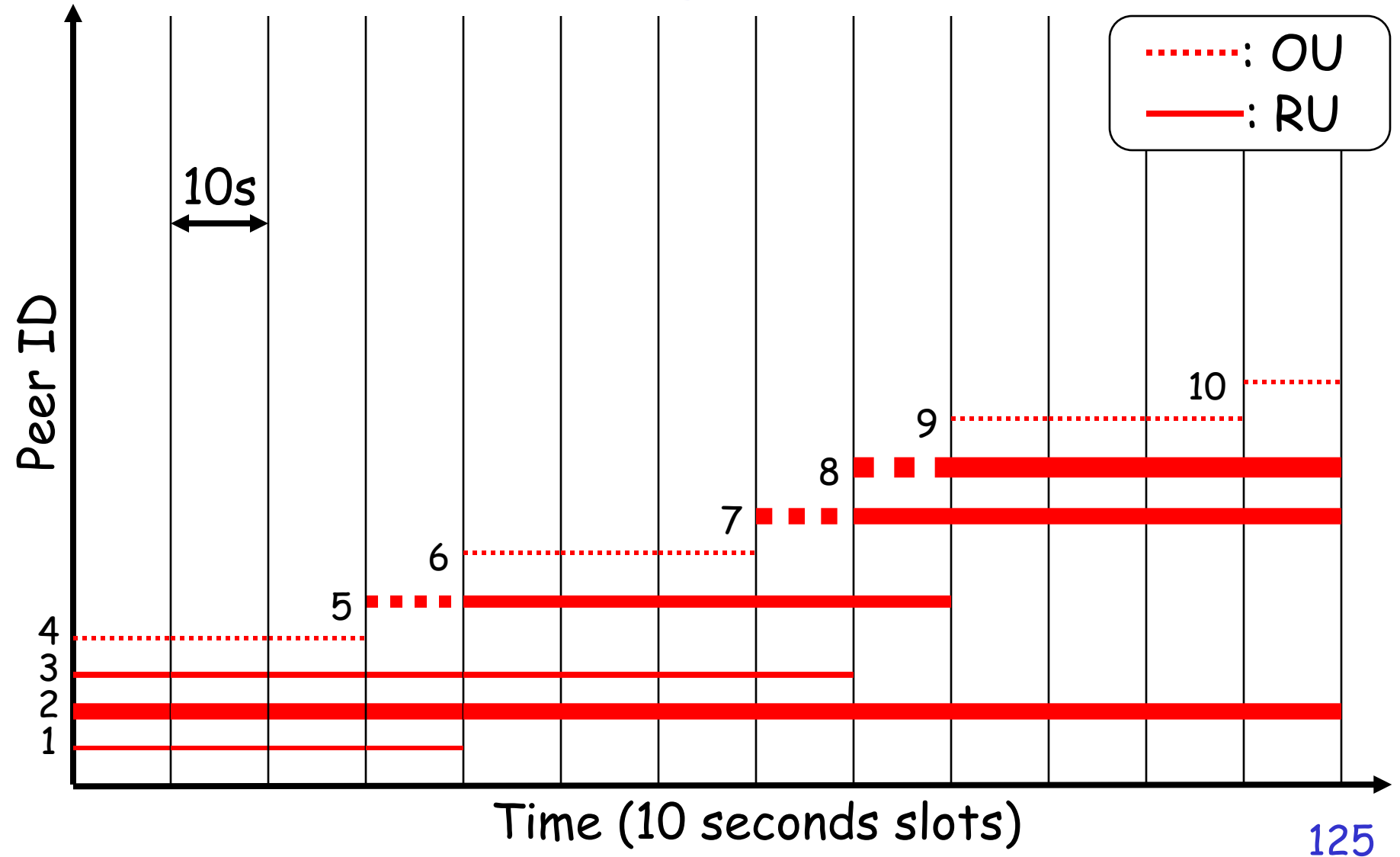
# Choke Algorithm: LS

- Leecher state (high level description)
  - Every 10 seconds the peers are ordered according to their download rate to the local peer
  - The 3 **fastest** and **interested** peers are unchoked
  - Every 30 seconds, one additional interested peer is unchoked at random
    - Called the optimistic unchoke

# Choke Algorithm: LS

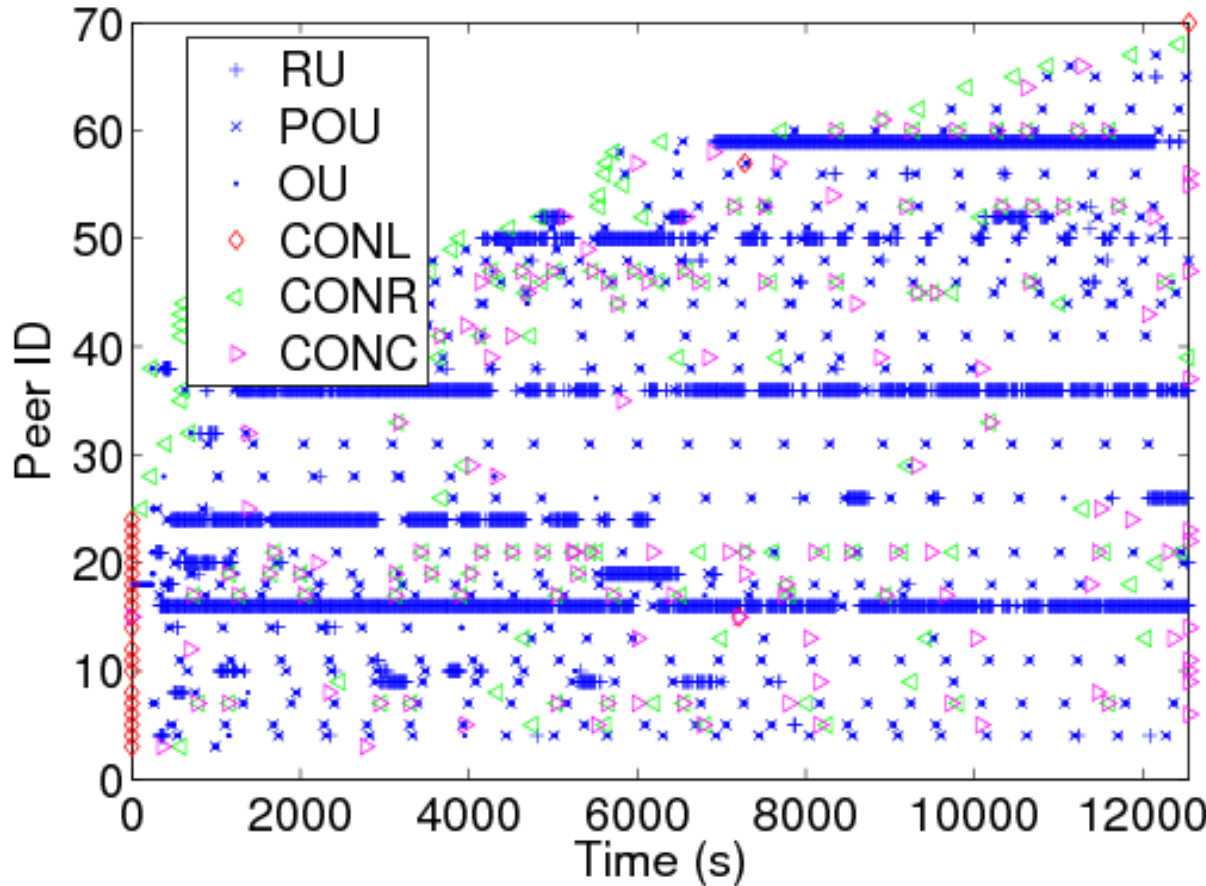
- Leecher state (high level description)
  - No more than 4 interested peers are unchoked at the same time

# Choke Algorithm: LS



# Real Torrent, LS

Important Protocol Events and Messages, LS



□ 1 seed, 26 leechers

▪ At torrent startup

□ 350 MB

□ Few peers with a lot of RU

□ Uniform OU

# Choke Algorithm: SS

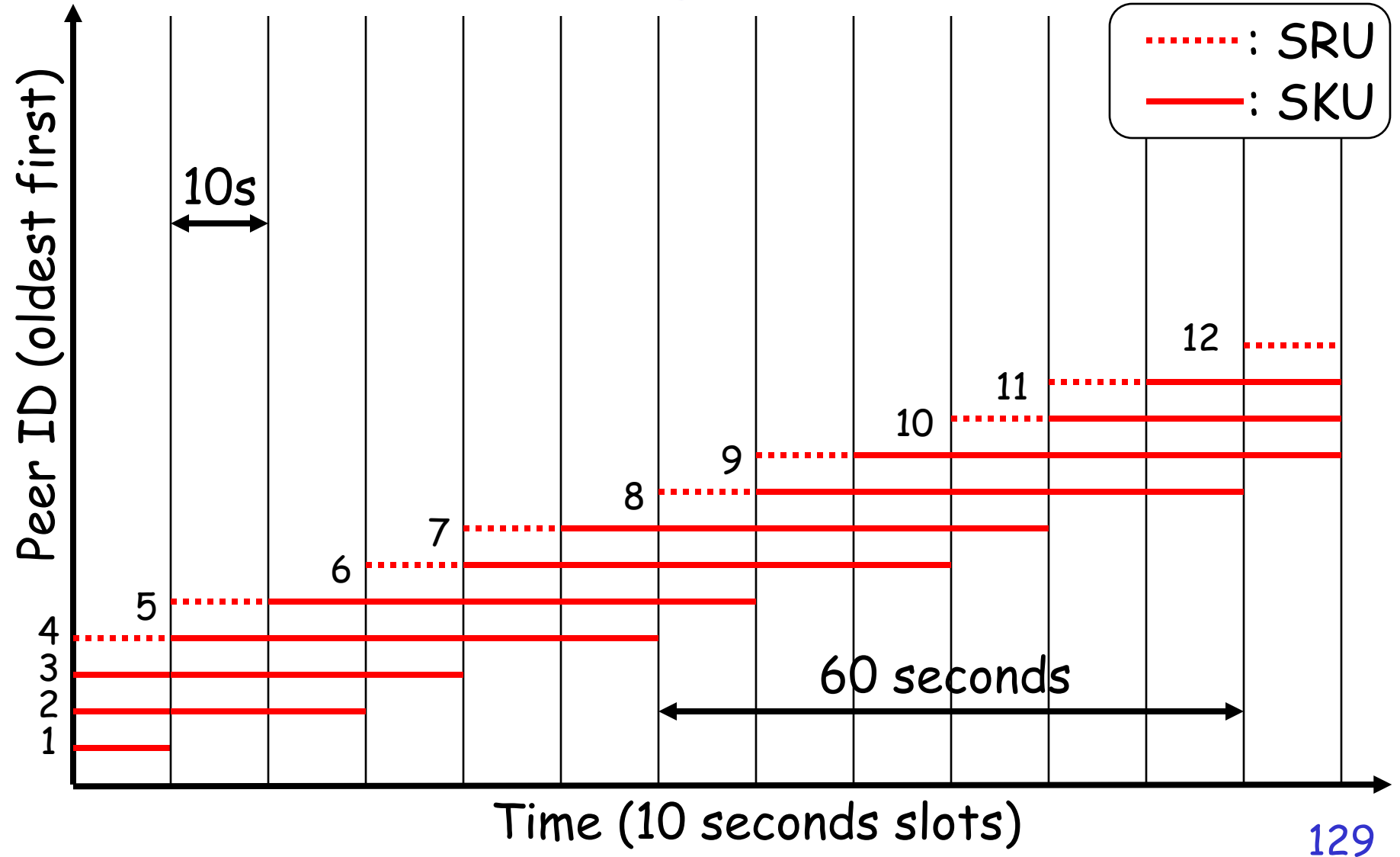
- ❑ Seed state, version FU (high level description)
  - FU: Favor Upload
  - Oldest version, but still largely used today
  - Same as in leecher state, but peers are ordered according to their upload rate from the local peer
  - In seed state, there is no download



# Choke Algorithm: SS

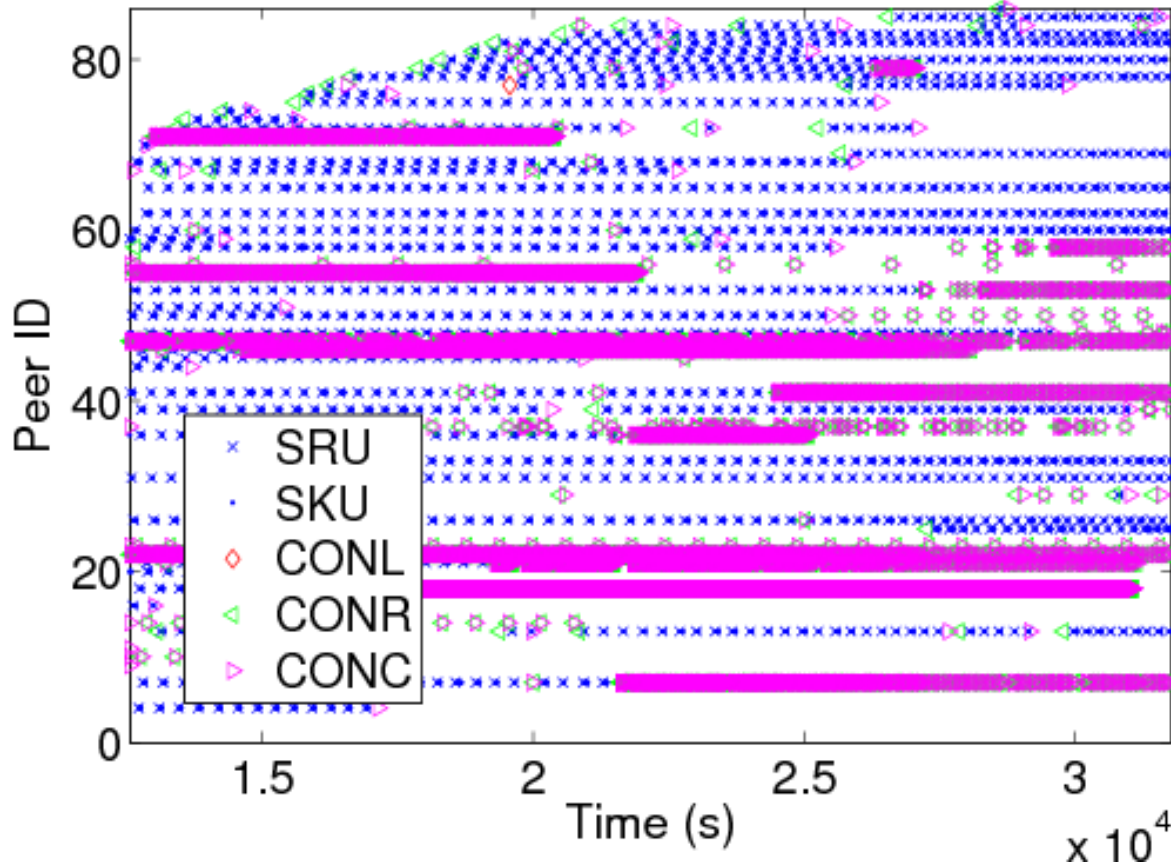
- Seed state, version RR (high level description)
  - RR: Round Robin
  - Appeared after the version FU, but not largely used today
  - Every 10 seconds the interested peers are ordered according to the time they were last unchoked
  - For two consecutive periods of 10 seconds, the 3 first peers are unchoked and an additional 4<sup>th</sup> interested peer is unchoked at random
  - For the third period of 10 seconds, the 4 first peers are unchoked

# Choke Algorithm: SS



# Real Torrent, SS

Important Protocol Events and Messages, SS



- 1 seed, 26 leechers
  - At torrent startup
- 350 MB
- Random SRU

# Choke Algorithm Properties

## □ Leecher state

- Robust to free riders
  - Only contributing peers get a good service from a leecher
- Leechers unchoked based on download evaluation
  - Selects the fastest interested peers
- Optimistic unchoke
  - Capacity discovery mechanism
- Global properties (see later)
  - Clusters peers according to their upload speed
  - Ensures efficient sharing incentive
  - Achieves high upload utilization

# Choke Algorithm Properties

## □ Seed state

- Algorithm FU still implemented in every client except
  - mainline 4.x.y, Ctorrent
- Algorithm FU
  - Not robust to free riders
    - The fastest peers get the service even if they do not contribute anything
  - Bad piece diversity
    - A single free rider can get most of the pieces

# Choke Algorithm Properties

## □ Seed state

- Algorithm RR robust to free riders
  - Every peer gets the same service time
- Increases the piece diversity

## □ Why FU more popular than RR

- RR is not well known (deployed experimentally on mainline 4.x.y)
- FU is more efficient in the present context
  - Few contributing peers with a large capacity

# Peer and Piece Selection

- Local rarest first piece selection [34]
  - Close to ideal entropy
  - Simple and efficient
  - No alternative today except network coding
- Choke algorithm (seed algorithm RR) [34,35]
  - Achieves high upload utilization
  - Clusters peers with similar upload capacity
  - Ensures effective sharing incentive
    - Robust to free riders
  - P2P fair

# Peer and Piece Selection

- ❑ Detailed description during BitTorrent presentation
  - Many important implementation details
- ❑ Extensive evaluation during BitTorrent presentation
  - Rarest first entropy
  - Choke algorithm fairness and efficiency

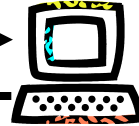


# Outline

- ❑ Overview
- ❑ Content Replication
- ❑ BitTorrent
  - Protocol Overview
  - Algorithm details
  - Evaluation
  - Advanced subjects
- ❑ Security
- ❑ Localization

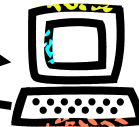
# BitTorrent Overview

Get a .torrent file that  
contains the address of

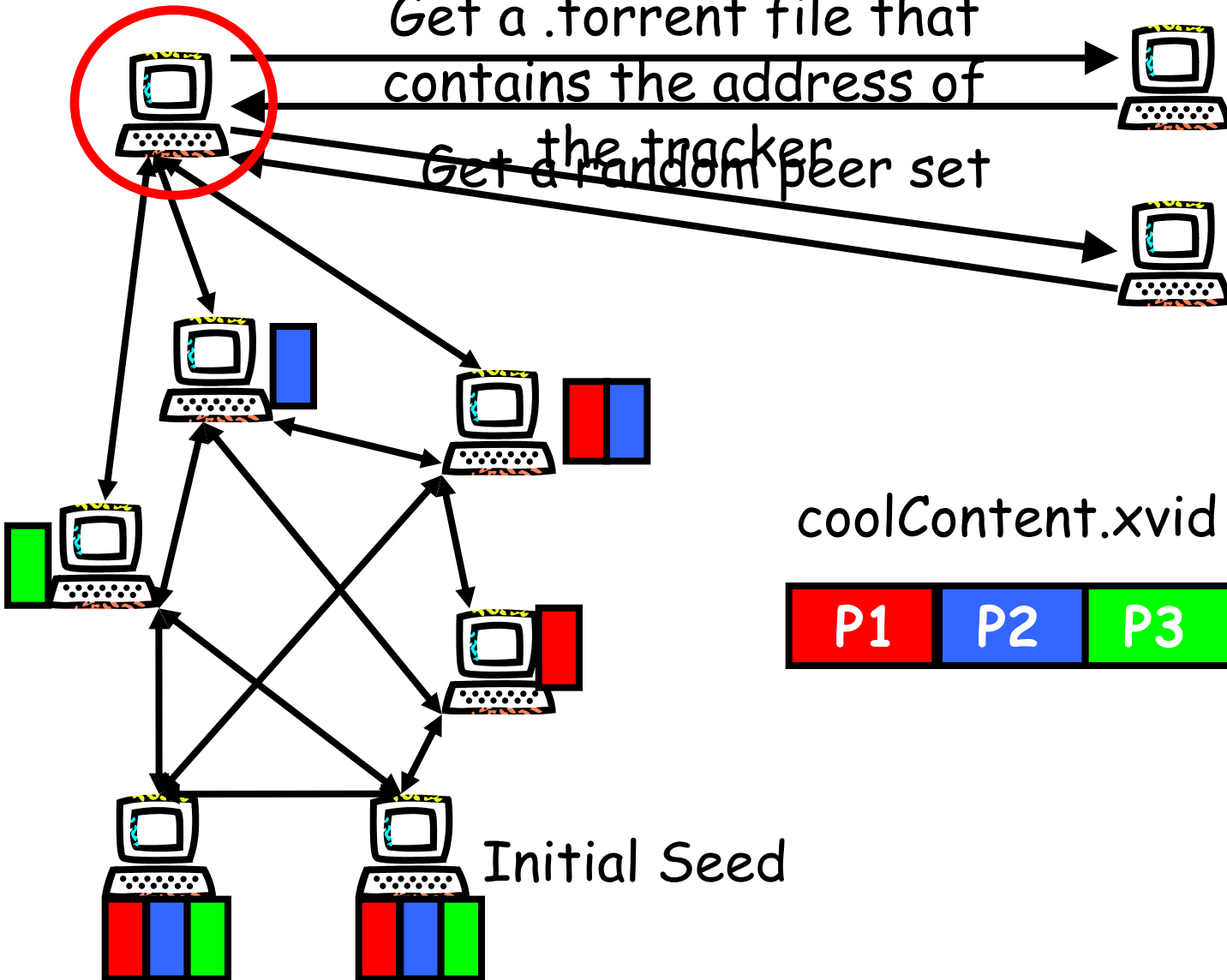


Web server

Get a random peer set



Tracker



coolContent.xvid



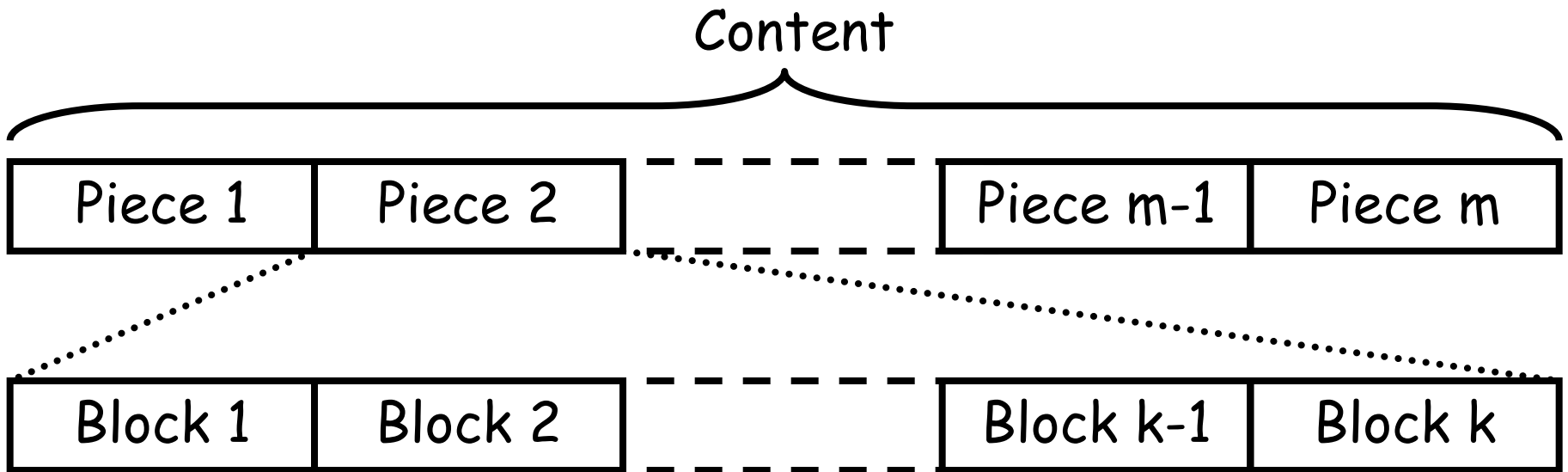
Initial Seed

# BitTorrent Specificities

- ❑ Specification [48] (obsolete [18][19])
- ❑ Unlike any other P2P protocol, there is one session per content
  - A session is called a torrent
  - Torrents are content based
- ❑ Torrents are independent
  - You get no benefit from previous or current torrents
  - No enforcement to stay as a seed

# Pieces and Blocks

- Content is split into pieces, which are split into blocks



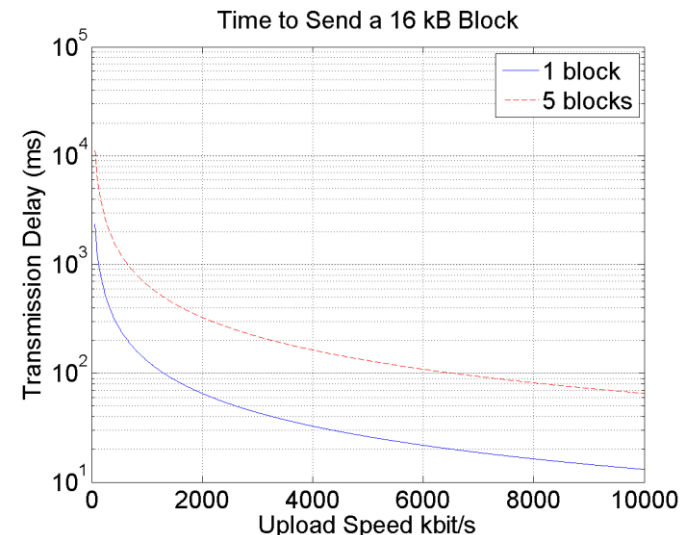
# Pieces and Blocks

## □ Pieces

- The smaller unit of retransmission
- Typically 256/512/1024/2048 kByte
- Size adapted to have a reasonably small .torrent file
  - One SHA-1 hash per piece in the .torrent file

## □ Blocks

- 16kB (hard coded)
- Used for pipelining
  - Always 5 requests pending



# .torrent file

□ .torrent file encoded using bencoding  
[48][18]

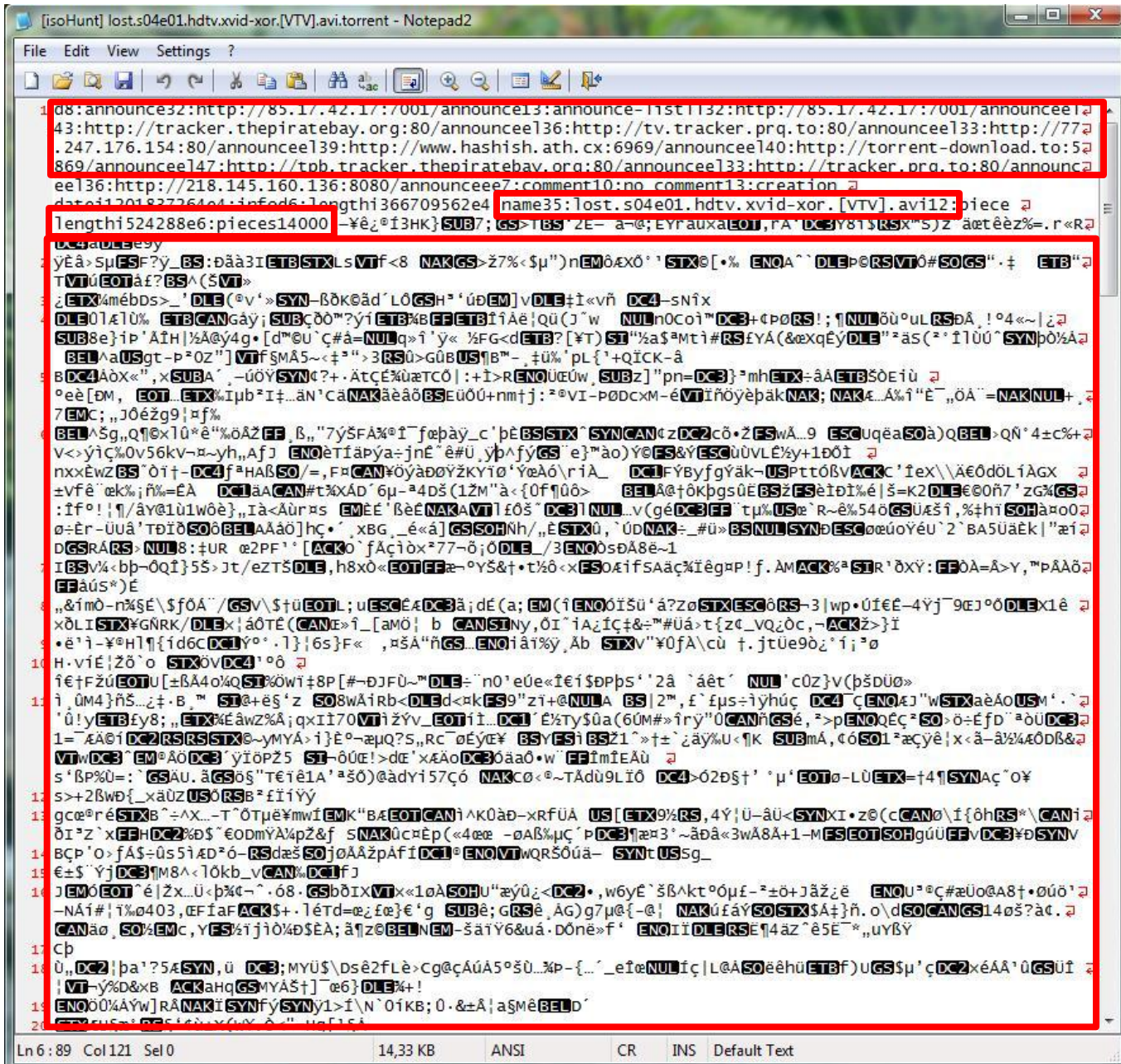
- **Info key**

- Length on the content in bytes
- md5 hash of the content (optional)
  - Not used by the protocol
  - pieces SHA-1 hash are enough
- File Name
- Piece length (256kB, 512kB, 1024kB, etc.)
- Concatenation of all pieces SHA-1 hash

# .torrent file

□ .torrent file encoded using bencoding  
[48][18]

- Info key slightly different for multi file content
- Announce URL of the tracker (HTTP)
  - Possibility of announce list for backup trackers
    - See [http://www.bittorrent.org/beps/bep\\_0012.html](http://www.bittorrent.org/beps/bep_0012.html)
- Some optional fields
  - Creation date, comment, created by





# Peer ID

- Peer ID = client ID + random string
  - Client ID: name of the client + its version
  - Random string: different each time the client is restarted (take as seed the system clock + the IP address of the peer)
- Examples [49]:
  - M4-0-2--41dabfd4760b
  - AZ2306-LwkWkRU95L9s
  - AZ2402-YbqhPheosA4a
  - BC0062->\*\xb1\xfdMm\xb9\x96\x96\xf0\xb8\xd9
  - UT1500-\xb5\x81\xf1+\xa3\xd3\xc7\xf3\x7f|\x1a\xb0

# Peer Bootstrap

- ❑ A peer download a .torrent file from a web server
- ❑ The peer client retrieves the tracker's URL and connect to it (HTTP GET)
- ❑ The peer sends
  - Info\_hash (SHA-1 of the info key)
  - Peer ID
  - Port the client listen on
  - Number of peers wanted (default 50)
  - Various statistics
    - Uploaded, downloaded, left, event (started, stopped, completed)

# Peer Bootstrap

- The tracker returns
  - Failure raison
  - Interval between statistics
  - A random list of peers already in the torrent
    - Peer ID, peer IP, peer port
    - Typically 50 peers
  - Statistics
    - Complete/incomplete (seed/leechers)

# Peer Bootstrap

- ❑ Tracker not involved in file distribution
  - Low overhead
- ❑ Web server acts as a certification authority
  - If a .torrent file is certified by the web server, there is no way to corrupt the torrent
    - Each piece SHA-1 hash is in the .torrent file
  - No strict certification

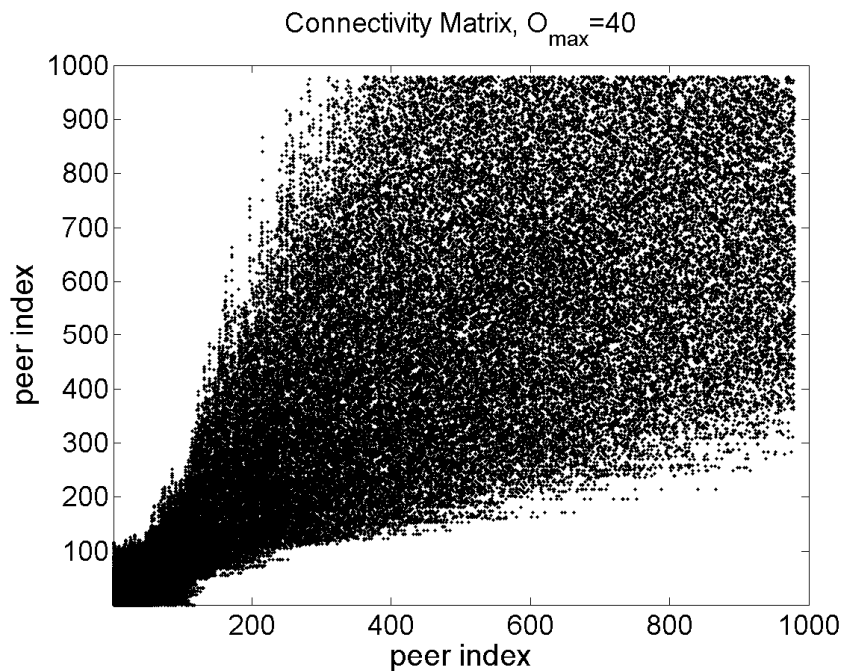
# Peer Bootstrap

- Peer connects to each peer returned by the tracker
  - At most 40 outgoing connections
  - Remaining peers kept as a pool of peers to connect to
- **Peer set** (neighbor set) size 80
  - Maximum 40 outgoing connections
  - Maximum 80 connections in total
  - Results in a well connected graph (random graph)
    - Recent peers get a chance to be connected to old peers

# Graph Construction

- The tracker allows to construct a random graph
  - Robust to **churn**
  - Low diameter  $D$ 
    - In  $\log_{(d-1)}N$  where  $d$  is the outdegree and  $N$  is the number of peers
    - For  $d=80$  and  $N=10\ 000$ ,  $D=2.1$
    - For  $d=80$  and  $N=100\ 000$ ,  $D=2.63$
    - For  $d=80$  and  $N=1\ 000\ 000$ ,  $D=3.16$
  - Low diameter is important for LRF

# Tracker connectivity matrix



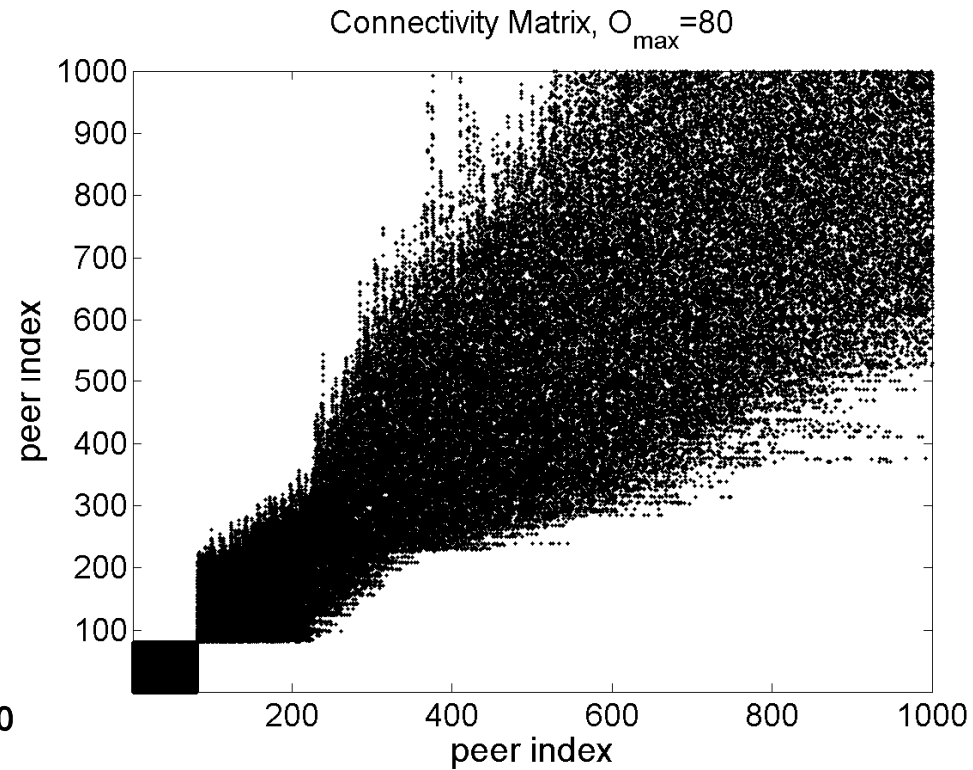
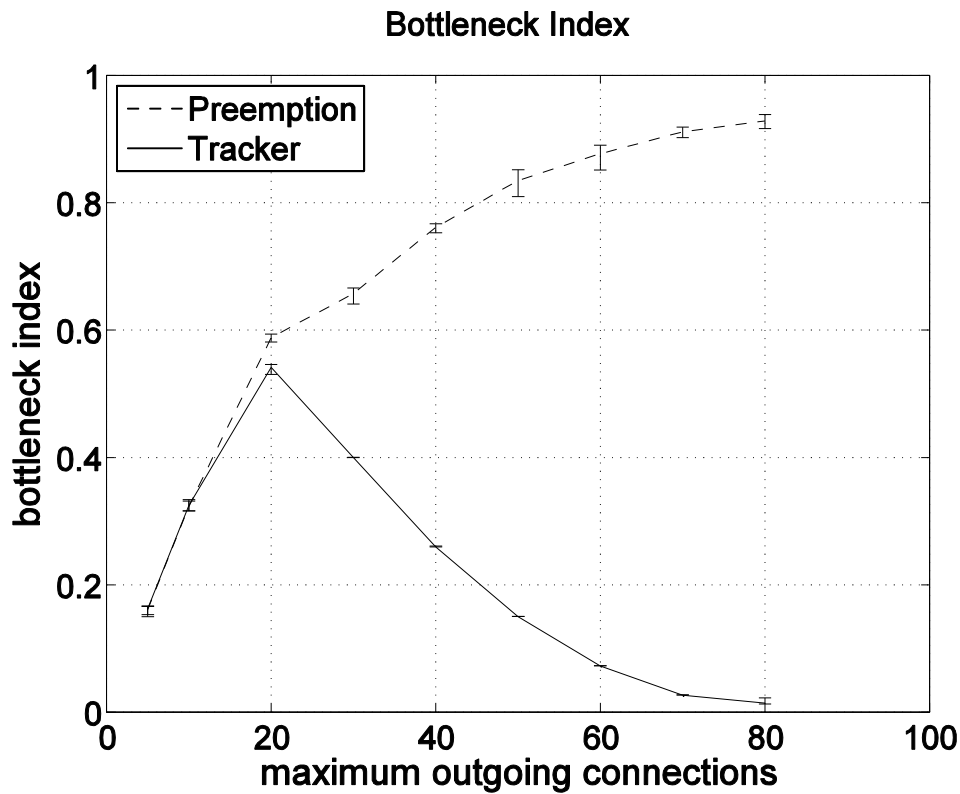
- 1000 peers
- 40 maximum number of outgoing connections
- Not a random graph
- Still well connected
- Refer to [44]
  - Impact of the maximum number of outgoing connections

# Bottleneck index

- ❑ Ratio of the number of connections between the first 80 peers and the rest of the peers, to the maximum possible number of such connections ( $80 \times 80 = 6400$  connections).
- ❑ Indication of the presence of a bottleneck between the initial seed peer set and the rest of the torrent



# Tracker connectivity matrix [44]



# Tracker Overhead

- Reconnect to the tracker
  - If the peer set size falls below 20
    - Ask for new peers
    - Small enough to avoid frequent tracker requests
  - Every 30 minutes
    - For statistics: amount of bytes uploaded/downloaded, number of bytes left
  - When the peer leaves the torrent
    - For statistics
    - To update the list of peers

# Tracker Peer Maintenance

- NAT check is performed in the mainline client
  - Not described in the protocol
- Not in the protocol
  - Peer health check
    - A peer that crashes will not report its disconnection
  - Intelligent random peer selection
    - No predefined ratio of leechers and seeds
    - Seeds can be returned to a seed

# Tracker Scrape Mode

- Used to get statistics on torrents
  - **complete**: # of seeds
  - **incomplete**: # of leechers
  - **downloaded**: # of peers who completed a download
  - **name**: (optional) name of the torrent in the .torrent file (not used in most popular tracker)

# Tracker Scrape Mode

- ❑ If the scrape request contains an infohash
  - Returns statistics for this torrent
- ❑ If no infohash is given
  - Returns statistics for all torrents hosted by the tracker
  - Might be a huge amount of data
    - Some trackers return a compressed list (using, e.g., gzip)

# State of the Art Tracker

## ❑ Opentracker

- <http://erdgeist.org/arts/software/opentracker/>
- <http://opentracker.blog.h3q.com/>

❑ Used by ThePirateBay

❑ Used by the opentracker infrastructure

# Alternative Peer Bootstrap Strategies

- Most recent clients (uTorrent, mainline, Vuze) might use a different bootstrap
  - Magnet link (BEP 9)
  - DHT (BEP 5)
  - PEX
  - No limit on the number of incoming connections
- Not well studied
  - Might impact the connectivity matrix, thus the efficiency

# How to Setup a Torrent?

## □ Create a .torrent using

- The content for this torrent. Used to compute the SHA-1 of each piece and infohash
- The URL of the tracker (you can create your own tracker or use a public tracker)
- Example of command line
  - `btmaketorrent.py http://my.tracker.fr:16661/announce myContent.tgz`



# How to Setup a Torrent?

- ❑ Upload the .torrent file to a torrent discovery site (Mininova, ThePirateBay, etc.)
- ❑ Seed the content
  - Simply start a client using the .torrent file and the content to be seeded
  - Example of command line
    - `btdownloadheadless.py --minport 16662 --maxport 16662 --save_as myContent.tgz myContent.tgz.torrent`

# Tracker Side Effects

- ❑ No explicit declaration to the tracker
  - The first peer to join the tracker implicitly declares the torrent (using the infohash) to the tracker
- ❑ Public trackers can be used for any torrent

# Tracker Side Effects

- ❑ Possible to download a torrent without the .torrent file
  - You can get the infohash using scrape mode
  - Need to guess the piece size
    - Not so many possibilities
  - Impossible to check piece integrity
    - But unlikely to have many corrupted pieces
    - Can try several times pieces from different peers

# Peer-to-Peer Protocol State

- Each peer maintains for each remote peer it is connected to the following state
  - `am_choking`: the local peer is choking this remote peer
  - `am_interested`: the local peer is interested in at least one piece on this remote peer
  - `peer_choking`: this remote peer is choking the local peer
  - `peer_interested`: this remote peer is interested in at least one piece of this local peer

# Peer-to-Peer Protocol State

- The local peer can receive data from a remote peer if
  - The local peer is interested in the remote peer
  - The remote peer unchoked the local peer

# Peer-to-Peer Protocol [48]

- Unlike client-server architectures, this is not the client who decides when to receive data
  - A peer can always refuse to serve another peer
- The decision to unchoke a peer is taken by the choke algorithm
- The choice of the piece to request is taken using the rarest first algorithm

# Peer-to-Peer Protocol Messages

- ❑ All the connections among the peers are over TCP
  - TCP/IP header overhead: 40 bytes
- ❑ To initiate connections and maintain the state, there are 11 messages in the P2P protocol

# Peer-to-Peer Protocol Messages

## □ HANDSHAKE

- Two way handshake
- To initiate a connection between two peers
- Once initiated, the connection is symmetric
- Contains (68 bytes)
  - Pstrlen=19 (protocol string identifier length)
  - Pstr="BitTorrent protocol"
  - Reserved (8 bytes)
  - Info\_hash
  - Peer ID



# Peer-to-Peer Protocol Messages

- Once a connection is initiated, all the messages on this connection are of the form
  - `<length prefix><message ID><payload>`
    - `<length prefix>`: 4 bytes (max length:  $2^{32}$ )
    - `<message ID>`: decimal char
    - `<payload>`: message dependant

# Peer-to-Peer Protocol Messages

## □ KEEP ALIVE (4 bytes)

- <len=0000>
- Sent every 2 minutes unless another message is sent
- Because there is no way to find that a TCP connection is dead unless sending a message

## □ CHOKE (5 bytes)

- <len=0001><ID=0>
- Sent from A to B when A choke B

# Peer-to-Peer Protocol Messages

## □ UNCHOKER (5 bytes)

- <len=0001><ID=1>
- Sent from A to B when A unchoke B

## □ INTERESTED (5 bytes)

- <len=0001><ID=2>
- Sent from A to B when A is interested in B

## □ NOT\_INTERESTED (5 bytes)

- <len=0001><ID=3>
- Sent from A to B when A is not interested in B

# Peer-to-Peer Protocol Messages

## □ HAVE (9 bytes)

- `<len=0005><ID=4><piece index>`
- Sent from a peer to all the peers in its peer set when it just received the piece with ID `<piece index>`, and that the SHA-1 of this piece is checked
- Send HAVE to peers that already have the piece
  - Send HAVE to the seeds

# Peer-to-Peer Protocol Messages

## □ HAVE

- HAVE sent to each peer in the peer set is not required for the correct protocol operation
  - Suppress HAVE for all peers that already have the piece?
- However, this information is useful for torrent monitoring
  - Exactly knows who has what
- Mainline 5.0.5 comment in source code
  - # should we send a have message if peer already has the piece?
  - # yes! it is low bandwidth and useful for that peer.

# Peer-to-Peer Protocol Messages

□ BITFIELD ( $\text{Ceil}[(\# \text{ of pieces})/8] + 5$  bytes)

- $\langle \text{len} = 0001 + X \rangle \langle \text{ID} = 5 \rangle \langle \text{bitfield} \rangle$
- First message sent after the handshake
  - No more sent in the following
  - Sent by both peers once the connection is initialized
- Bit  $i$  in the bitfield is set to 1 if the peer has piece  $i$ , 0 otherwise

# Peer-to-Peer Protocol Messages

## □REQUEST (17 bytes)

- `<len=0013><ID=6><index><begin><length>`
- Sent from peer A to peer B to request to peer B the block
  - Of the piece with index `<index>`
  - Starting with an offset `<begin>` within the piece
  - Of length `<length>`

# Peer-to-Peer Protocol Messages

## □REQUEST

- The message allows to specify the block length, but it is hard coded in the client
- Changing the block size may be useful to improve pipelining under certain conditions
  - No study on the block size impact
- Block size larger than  $2^{17}$  is forbidden



# Peer-to-Peer Protocol Messages

□PIECE ( $2^{14} + 13$  bytes for a standard block size)

- $\langle \text{len} = 0009 + X \rangle \langle \text{ID} = 7 \rangle \langle \text{index} \rangle \langle \text{begin} \rangle \langle \text{block} \rangle$
- Only one message used to send blocks
- Sent from peer A to peer B to send a block of data to peer B
  - Of the piece with index  $\langle \text{index} \rangle$
  - Starting with an offset  $\langle \text{begin} \rangle$  within the piece
  - Payload is  $\langle \text{block} \rangle$

# Peer-to-Peer Protocol Messages

## □ CANCEL (17 bytes)

- `<len=0013><ID=8><index><begin><length>`
- Used in end game mode only
- Sent from peer A to peer B to cancel a request already sent to peer B for the block
  - Of the piece with index `<index>`
  - Starting with an offset `<begin>` within the piece
  - Of a length `<length>`

# BitTorrent BEP

- BEP: BitTorrent Enhancement Proposal
  - <http://bittorrent.org/>
- List of current accepted and draft BEP
  - [http://bittorrent.org/beps/bep\\_0000.html](http://bittorrent.org/beps/bep_0000.html)
- BEP 6: Fast Extension
  - [http://bittorrent.org/beps/bep\\_0006.html](http://bittorrent.org/beps/bep_0006.html)
  - Based on experience on the protocol
  - No standard yet

# Outline

- ❑ Overview
- ❑ Content Replication
- ❑ BitTorrent
  - Protocol Overview
  - **Algorithm details**
  - Evaluation
  - Advanced subjects
- ❑ Security
- ❑ Localization

# BitTorrent Version

- All results in this section from mainline client 4.0.2
  - Mainline is the reference BitTorrent implementation
    - Written in python
    - Developed by Bram Cohen, still maintained by his company
  - Mainline 4.0.2 was released in may 2005
  - Major modifications since 2005
    - Tracker less extension
    - GUI improvement
    - Localization
    - UPnP
    - Etc.
  - No major modification to the core algorithms (peer and piece selection)

# BitTorrent Version

- Are the results given in this course still relevant?
  - 4.0.2 is the BitTorrent implementation as specified in the BitTorrent official specification (BEP 3) + the RR choke algorithm in seed state
  - Current (2010) uTorrent/mainline algorithms are the same as 4.0.2 with FU seed state algorithm

# Piece Selection

- 4 policies
  - Strict priority
  - Random first piece
  - Local rarest first
  - Endgame mode

# Strict Priority

- ❑ Once a block of a piece has been requested, request all the other blocks of the same piece before a block of any other piece
- ❑ Rationale
  - Pieces are the unit of replication
    - It is important to download a piece as fast as possible, only complete pieces can be retransmitted



# Strict Priority

- ❑ Strict priority improves piece download speed
- ❑ Never blocking
  - If a peer is choked during a piece download and this piece is not available on any other peer, a new piece will be requested.
  - As soon as the partially downloaded piece is available, request the remaining blocks with highest priority

# Random First Piece

- ❑ For the first 4 downloaded pieces, the pieces are selected at random
- ❑ Rationale
  - Rare pieces may be slower to download
    - In particular if present on a single peer
  - A peer without a piece cannot reciprocate
    - Must wait for an optimistic unchoke
    - The first piece is most of the time received from several other peers that performs OU

# Random First Piece

- When a peer starts, it receives its first piece with an OU
  - With a typical upload speed of 20kB/s, each unchoked peer receives at 5kB/s (4 in parallel)
  - For a piece of 256kB, needs 51 seconds at 5kB/s to receive a piece
  - But, an OU lasts for 30s only
    - An OU never becomes a RU when the peer has no piece to reciprocate
  - Faster to complete the last blocks of the piece from another peer that makes an OU if the piece is not the rarest one

# Random First Piece

- ❑ Random first piece makes more likely to complete the first piece faster
- ❑ Not optimal, but a good tradeoff between simplicity and efficiency (the random piece may be a rarest one)
- ❑ Only impacts the startup phase of a peer
- ❑ Then switches to local rarest first

# Local Rarest First

- ❑ Download first the pieces that are rarest in the peer set of the peer
- ❑ Rationale
  - Cannot maintain the state for all peers
    - Require a connection to all peers or a centralized component
  - Peer selection should not be constrained by piece availability
    - Entropy
  - The initial seed should send as fast as possible a first copy of the content

# Local Rarest First

- Improve the entropy of the pieces
  - Peer selection is not biased
  - Better survivability of the torrent
    - Even without a seed the torrent is not dead
- Increase the speed at which the initial seed delivers a first copy of the content
  - The seed can leave early without killing the torrent

# Endgame Mode

- ❑ When all blocks are either received or have pending requests, request all not yet received blocks to all peers. Cancel request for blocks received.
- ❑ Rationale
  - Prevent the termination idle time

# Endgame Mode

- ❑ Improve the termination idle time
- ❑ Not a major impact at the scale of a download
- ❑ Do not solve the last pieces problem
  - An overloaded peer remains overloaded



# Piece Selection Code

□ For each remote peer (Downloader.py)

```
def _want(self, index):
```

```
# self.have[index] is true if the remote peer has the piece  
# with ID index
```

```
# do_I_have_requests returns true if there are still  
# blocks missing for the piece with ID index, or if the  
# piece was not downloaded at all yet.
```

```
    return self.have[index] and  
           self.downloader.storage.do_I_have_requests(index)
```

```
# self.have.numfalse == 0 true if the remote peer is a seed  
self.downloader.picker.next(self._want, self.have.numfalse  
                             == 0)
```

# Piece Selection Code

□ PiecePicker.py

```
def next(self, havefunc, seed = False):
```

```
    #bests: pieces to be selected with strict priority
```

```
    bests = None
```

```
    bestnum = 2 ** 30
```

```
    # s is a list of the partially downloaded pieces.
```

```
    # not clear why they make a distinction between
```

```
    # seeds and leechers.
```

```
    if seed:
```

```
        s = self.seedstarted #pieces partially downloaded from seeds
```

```
    else:
```

```
        s = self.started #pieces partially downloaded from leechers
```

# Piece Selection Code

Number of peers  
with piece i

❑ Strict priority  
for i in s:

*#havefunc(i) (means want\_(i)): selects the rarest pieces  
among the ones already requested*

if havefunc(i):

if self.numinterests[i] < bestnum:

bests = [i]

bestnum = self.numinterests[i]

elif self.numinterests[i] == bestnum:

bests.append(i)

if bests:

*#returns one element of bests at random*

return choice(bests)

# Piece Selection Code

□ Random first

if self.numgot <

self.config['rarest\_first\_cutoff']:

*#scrambled: random list of pieces*

for i in self.scrambled:

if havefunc(i):

return i

return None

Number of pieces  
already received

# Piece Selection Code

❑ Rarest first

*#xrange(1,n) returns elements from 1 to n*

*#without list creation, unlike range()*

```
for i in xrange(1, min(bestnum, len(self.interests))):
```

*#start with the rarest pieces*

```
for j in self.interests[i]:
```

```
    if havefunc(j):
```

```
        return j
```

```
return None
```

interests is a list that contains at the indice i the list of the pieces that are copied i times in the peer set.

# Peer Selection

- Choke algorithm
  - Leecher state
  - Seed state

# Choke Algorithm Leecher State

- ❑ Algorithm Called (**round**)
    - Every 10 seconds
  - ❑ In **addition** algorithm called
    - Each time an unchoked and interested peer leaves the peer set
    - Each time an unchoked peer becomes interested or not interested
- ⇒ Shorten the reactivity

# Choke Algorithm Leecher State

- Every 3 rounds (30 seconds)
  - An interested and choked peer is selected at random
    - Planned optimistic unchoke
  - It will be unchoked later in the algorithm



# Choke Algorithm Leecher State

- Each time the algorithm is called
  - Order the peers interested and **not snubbed** according to their download rate (to the local peer)
    - Snubbed
      - Did not send a block in the last 30 seconds
      - Favor peers that have contributed recently
  - Unchoke the 3 fastest peers
  - If the planned optimistic unchoke is not part of the 3 fastest, it is unchoked, **DONE**

# Choke Algorithm Leecher State

- Each time the algorithm is called
  - If the planned optimistic unchoke is one of the 3 fastest
    - Choose another peer at random
      - New planned optimistic unchoke
    - If this new planned optimistic unchoke is interested, unchoke it, **DONE**
    - If this new planned optimistic unchoke is not interested, unchoke it and chose another planned optimistic unchoke at random, loop again



# Choke Algorithm Leecher State

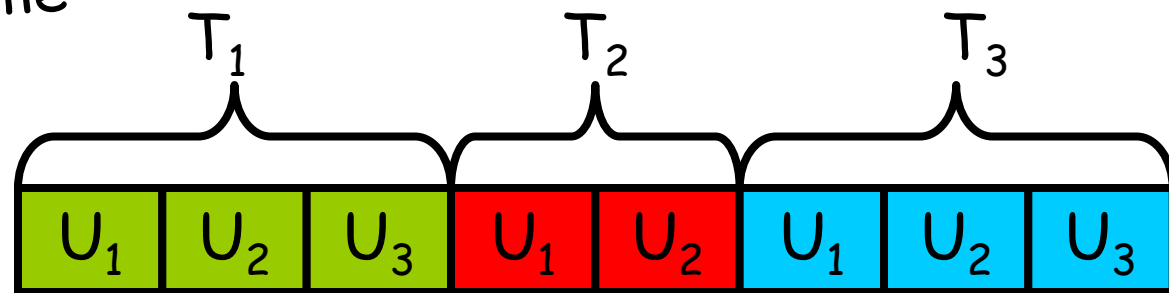
- ❑ At most 4 peers can be interested and unchoked
- ❑ But, more than 4 peers can be unchoked
  - In case an unchoked peer becomes interested, the choke algorithm is called immediately
  - Improve the reactivity in case there are few interested peers

# Choke Algorithm Seed State

- Algorithm RR
  - Algorithm Called (**round**)
    - Every 10 seconds
  - In **addition** algorithm called
    - Each time an unchoked and interested peer leaves the peer set
    - Each time an unchoked peer becomes interested or not interested
- ⇒ Shorten the reactivity

# Choke Algorithm Seed State

- ❑ Peers **unchoked** and **interested** less than 20 seconds ago or that have pending requests for blocks are ordered according to the time they were last unchoked, most recently unchoked peers first
  - Peers should be active or recent
- ❑ Upload rate discriminate among peers with the same unchoke time

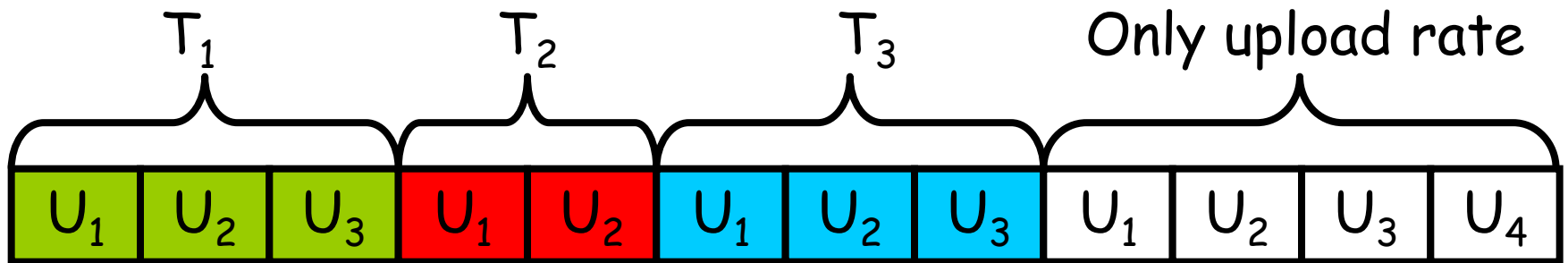


Last unchoked time:  $T_1 < T_2 < T_3$

Upload rates:  $U_1 > U_2 > U_3$

# Choke Algorithm Seed State

- All the other peers unchoked and interested are ordered according to their upload rate, with the lowest priority

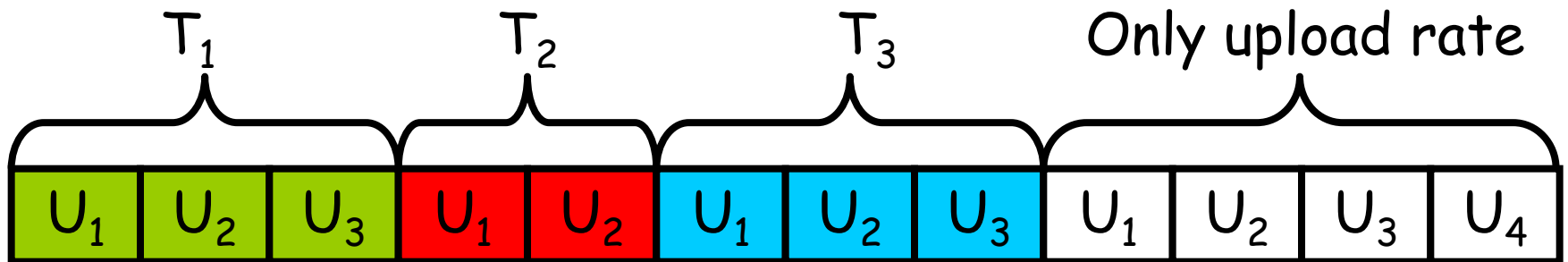


Last unchoked time:  $T_1 < T_2 < T_3$

Upload rates:  $U_1 > U_2 > U_3 > U_4$

# Choke Algorithm Seed State

- ❑ For two rounds out of three the three first peers are kept unchoked, and an additional peer choked and interested is unchoked at random
- ❑ For the third round, the four first peers are kept unchoked



Last unchoked time:  $T_1 < T_2 < T_3$

Upload rates:  $U_1 > U_2 > U_3 > U_4$

# Maximum Number of Interested Peers to Unchoke

## □ Default behavior

- A maximum of 4 interested peers to unchoke in parallel

## □ Depending on the implementations

- Increase this number according to the upload capacity
  - Rational is that the higher your upload capacity the higher the number of parallel uploads
- Increase this number with a configuration parameter



# Maximum Number of Interested Peers to Unchoke

- No clear evaluation of the benefit to increase the number of parallel uploads when the upload capacity is high
  - May be beneficial if your upload capacity is larger than 4 times the mean download capacity of the peers
    - For a mean maximum download speed of 1 Mbit/s your upload speed must be higher than 4 Mbit/s
    - No study on the mean maximum download speed
  - Studies on the number of parallel uploads do not take into account this asymmetry

# Peer Selection Code

- ❑ We study only the leecher state
- ❑ For the seed state, directly refer to `choker.py, _rechoke_seed()`

# Peer Selection Code

## ❑ Choker.py

```
class Choker(object):  
    def __init__(self, config, schedule, done = lambda:False):  
        schedule(self._round_robin, 10)
```

## ❑ Build connections[]

**#maintain the list in a random order**

```
def connection_made(self, connection):  
    p = randrange(len(self.connections) + 1)  
    self.connections.insert(p, connection)
```

# Peer Selection Code

□ Call to `_rechoke()`

```
def connection_lost(self, connection):  
    self.connections.remove(connection)  
    if connection.upload.interested and not  
        connection.upload.choked:  
        self._rechoke()
```

```
def interested(self, connection):  
    if not connection.upload.choked:  
        self._rechoke()
```

```
def not_interested(self, connection):  
    if not connection.upload.choked:  
        self._rechoke()
```

# Peer Selection Code

- Round and planned optimistic unchoke

```
def _round_robin(self):
    self.schedule(self._round_robin, 10)
    self.count += 1
    if self.done(): #test if it is a seed
        self._rechoke_seed(True)
        return
    if self.count % 3 == 0: #planned optimistic unchoke
        for i in xrange(len(self.connections)):
            u = self.connections[i].upload
            if u.choked and u.interested:
                #i first in connections[], no optimistic unchoke flag
                self.connections = self.connections[i:] +
                                    self.connections[:i]
            break
    self._rechoke()
```

# Peer Selection Code

□ `_rechoke()`

```
def _rechoke(self):
```

```
    if self.done(): #test if it is in seed state
```

```
        self._rechoke_seed()
```

```
    return
```

```
preferred = [] #sorted uploaders, fastest first
```

```
for i in xrange(len(self.connections)):
```

```
    c = self.connections[i]
```

```
    if c.upload.interested and not
```

```
        c.download.is_snubbed():
```

```
        preferred.append((-c.download.get_rate(), i))
```

```
preferred.sort()
```

# Peer Selection Code

**#SLICING: A[a,b] is inclusive in a, but exclusive in b**

**#A=(1,2,3,4), A[2:]=(3,4) A[:2]=(1,2)**

**#prefcount: nb of RU to perform (3 if len(preferred)>2)**

prefcount = min(len(preferred), 3)

mask = [0] \* len(self.connections)

```
for _, i in preferred[:prefcount]:  
    mask[i] = 1
```

# Peer Selection Code

```
count = max(1, 2 - prefcnt) #nb of OU to perform (2 if prefcnt=0)
for i in xrange(len(self.connections)):
    c = self.connections[i]
    u = c.upload
    #RU
    if mask[i]:
        u.unchoke(self.count)
    #OU
    elif count > 0:
        if u.interesting:
            count -= 1
            u.unchoke(self.count) #self.count != count
    else:
        u.choke()
```



# Outline

- Overview
- Content Replication
- BitTorrent
  - Overview
  - Algorithm details
  - Evaluation
    - Torrent Scale
    - Algorithms
  - Advanced subjects
- Security
- Localization

# BitTorrent Use Case

- ❑ BitTorrent is for efficient file replication
- ❑ It is not for
  - Content localization
    - May be hard to find a content
  - Content availability
    - Torrents can die fast, no incentive to keep a torrent up
  - Both issues are important, but orthogonal

# BitTorrent Evaluation Studies

- ❑ Tracker log analysis [20,21,22]
  - Old ones (2003-2004), historical interest
- ❑ Large scale crawl [42, 43]
- ❑ Client side instrumentation [18,34,35]
- ❑ Choke and rarest first algorithm evaluation
  - Simulations [33]
  - Experimentation [18,34,35]

# Outline

- ❑ Overview
- ❑ Content Replication
- ❑ BitTorrent
  - Overview
  - Algorithm details
  - Evaluation
    - Torrent Scale
      - Tracker log analysis
      - Large scale crawl
    - Algorithms
- ❑ Advanced subjects
- ❑ Security
- ❑ Localization

# Focus of This Section

- Give old but well known and referenced results on BitTorrent
  - Between 2003 and 2004
  - Historically interesting
  - Shows the famous supernova infrastructure
- Up-to-date results given in next section
  - Large BitTorrent crawls

# Architecture Availability [20]

- ❑ What is the availability of architecture web site/tracker?
- ❑ Study conducted on Suprnova
  - Around 46000 available files (October 2004)
  - Discontinued December 2005

# Architecture Availability [20]

## ❑ Suprnova architecture

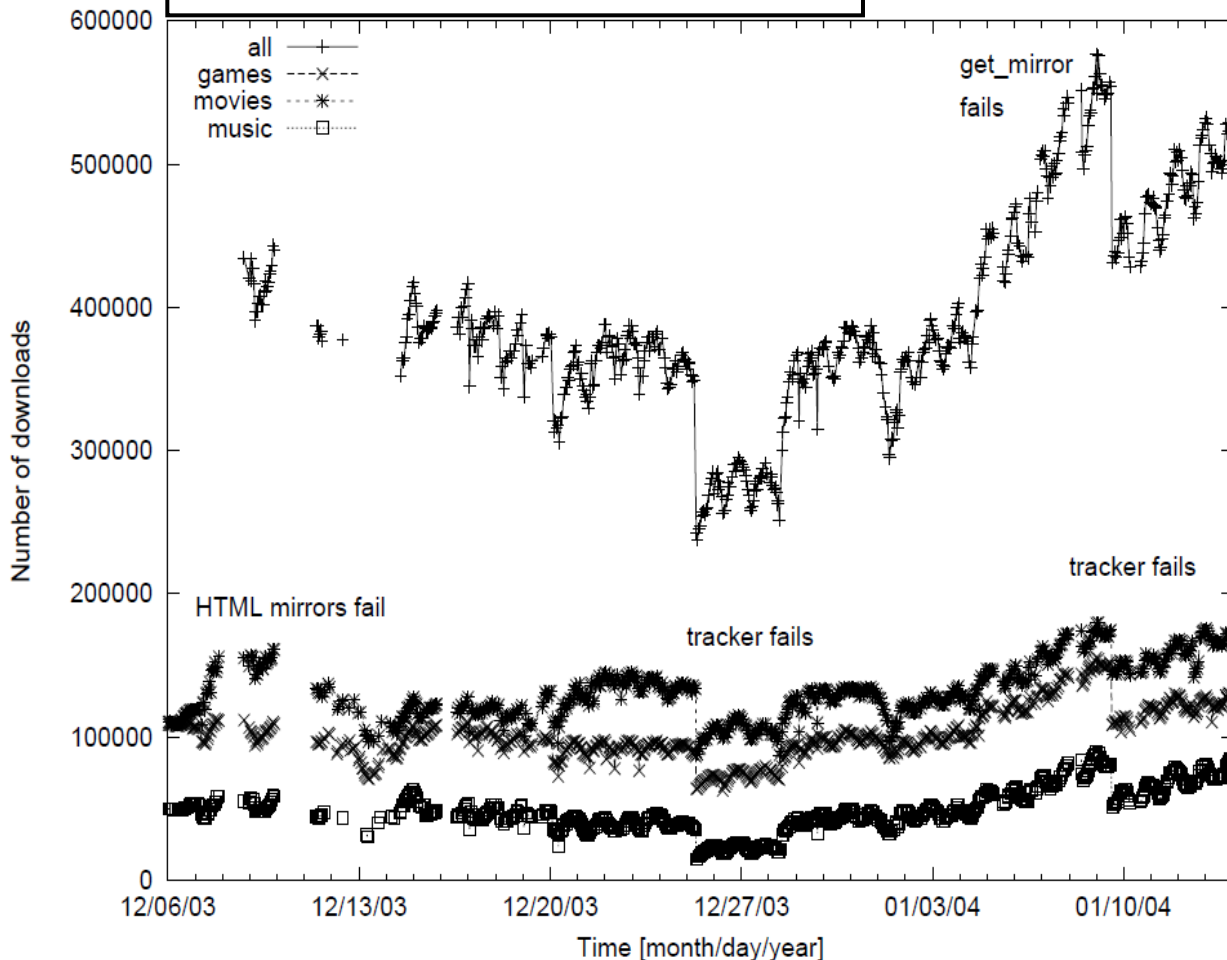
- Web site is mirrored for load balancing
  - 1 200 000 visitors per day (Oct 2004)
- Different servers host the .torrent files
- Different servers are trackers
  - Frequent DoS attacks, GB daily bandwidth

## ❑ Manual moderation

- Very efficient fake detection
- Moderated/unmoderated submitters

# Architecture Availability [20]

Credit: Pouwelse et al. [20]

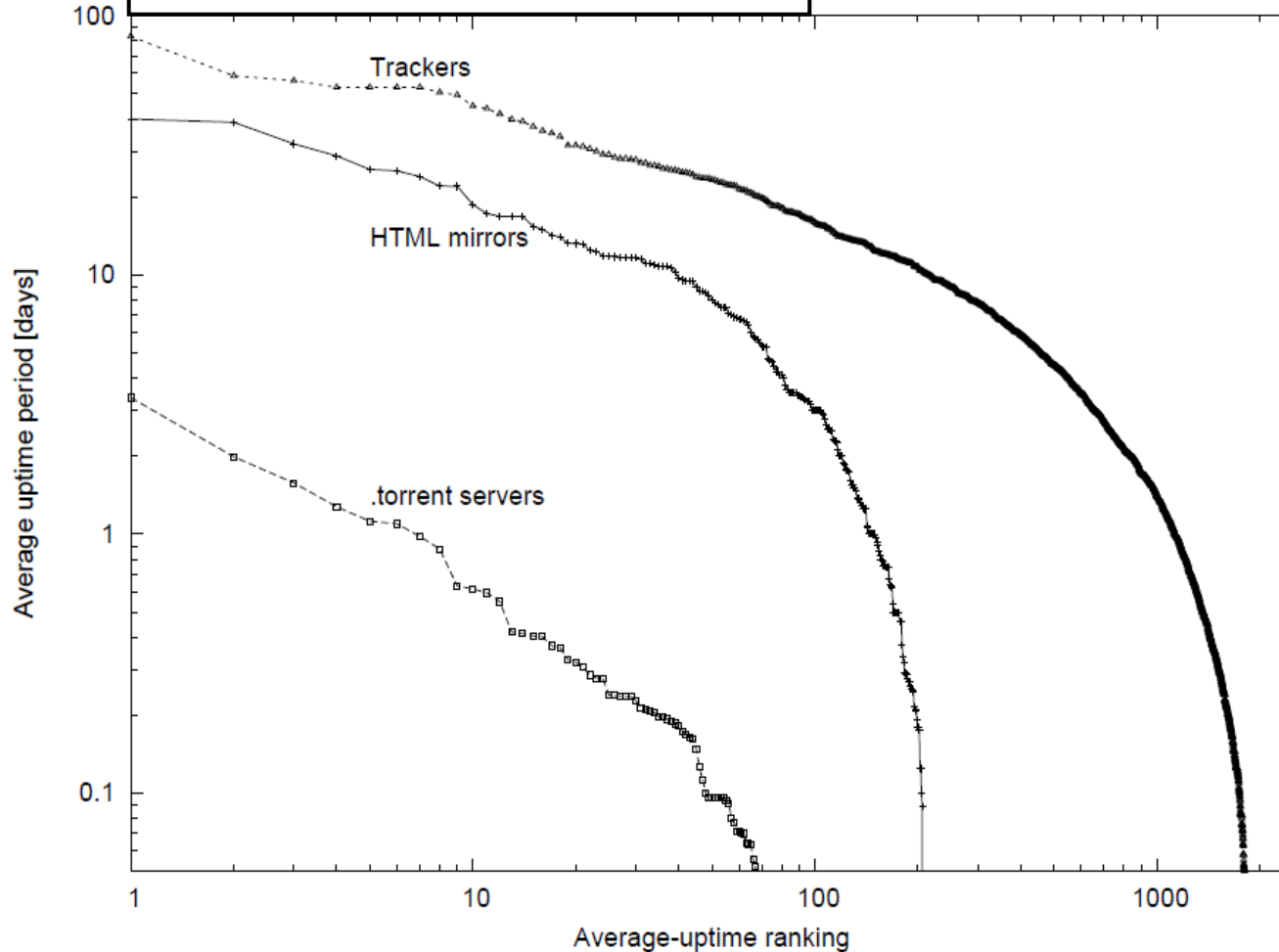


- Dec03-Jan04
- Daily cycle
- Number of user depends on availability



# Architecture Availability [20]

Credit: Pouwelse et al. [20]



□ 1941 trackers  
▪ 50% have average uptime lower than 1.5 day

□ 234 mirrors  
▪ 50% have average uptime lower than 2.1 days

□ 95 .torrent servers

□ Poor availability

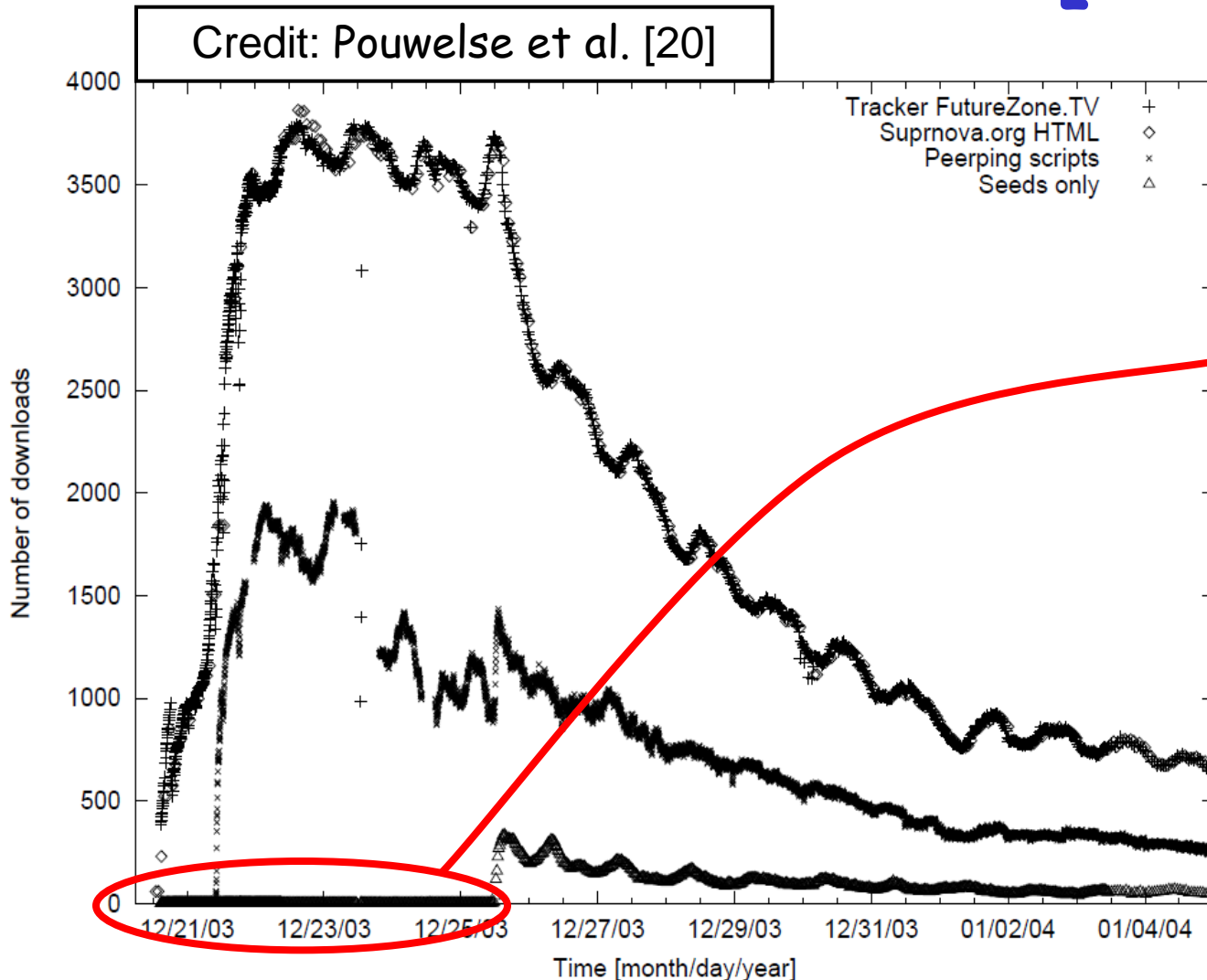
# Architecture Availability [20]

- ❑ Centralized component failures impact availability
- ❑ Need for decentralized components
- ❑ But, centralized component made the strength of Suprnova
  - Moderation
  - Single interface

# Peer Arrival Time

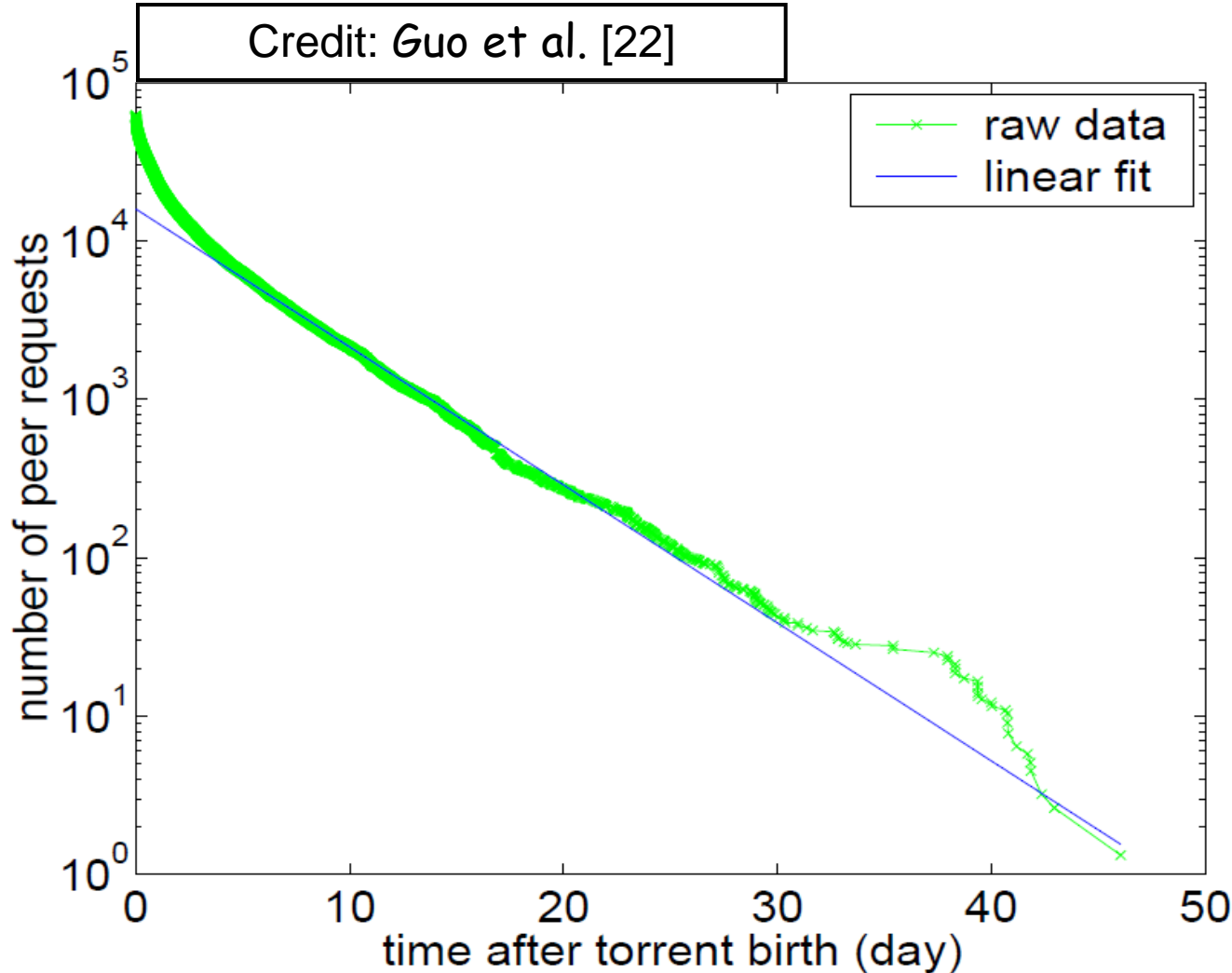
- How peers arrive during the torrent lifetime?

# Flash Crowd [20]



- Lord of the ring III
- 1.87GB
- Only 1 seed for 5 days
- Peering scripts give lower number due to firewall issues
- See also [21]

# Peer Requests With Time [22]



- Trace collected from a single tracker (550 torrents) during 48 days end of 2003
- Exponential decrease with time

# Peer Arrival Time

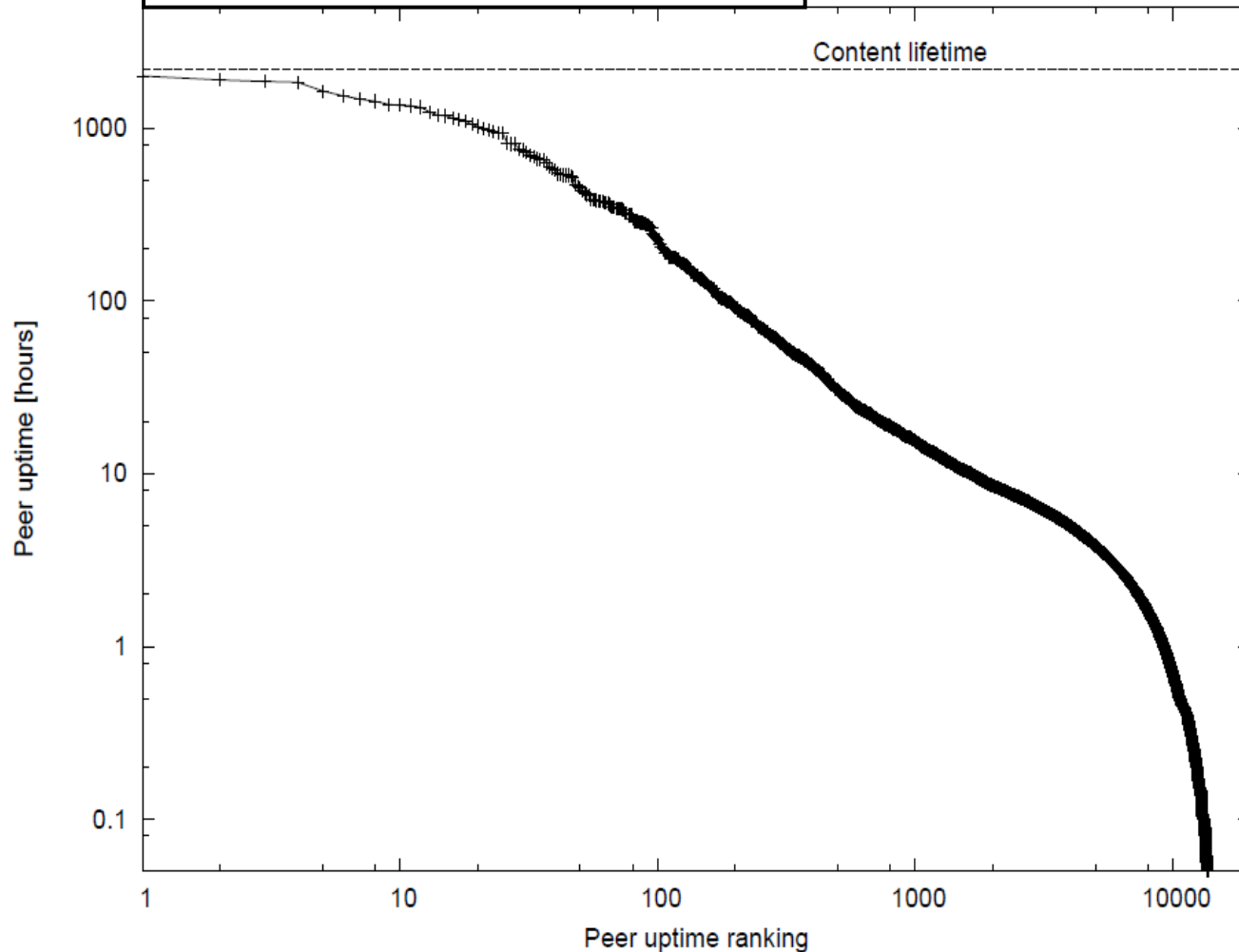
- ❑ Number of peers increases exponentially during the flash crowd
- ❑ Number of peers decreases exponentially after the flash crowd, but at a slower rate

# Peer Availability [20]

- How long does a peer stay in seed state?
- Ubisoft PC game "Beyond Good and Evil"
  - Available using BT in December 10, 2003
  - Torrent died on March 11, 2004
  - 90 155 different peers identified, but only 53 883 can be tracked (firewall problem)

# Peer Availability in Seed State [20]

Credit: Pouwelse et al. [20]



- 17% have uptime > 1 hour
- 3.1% have uptime > 10 hours
- 0.34% have uptime > 100 hours

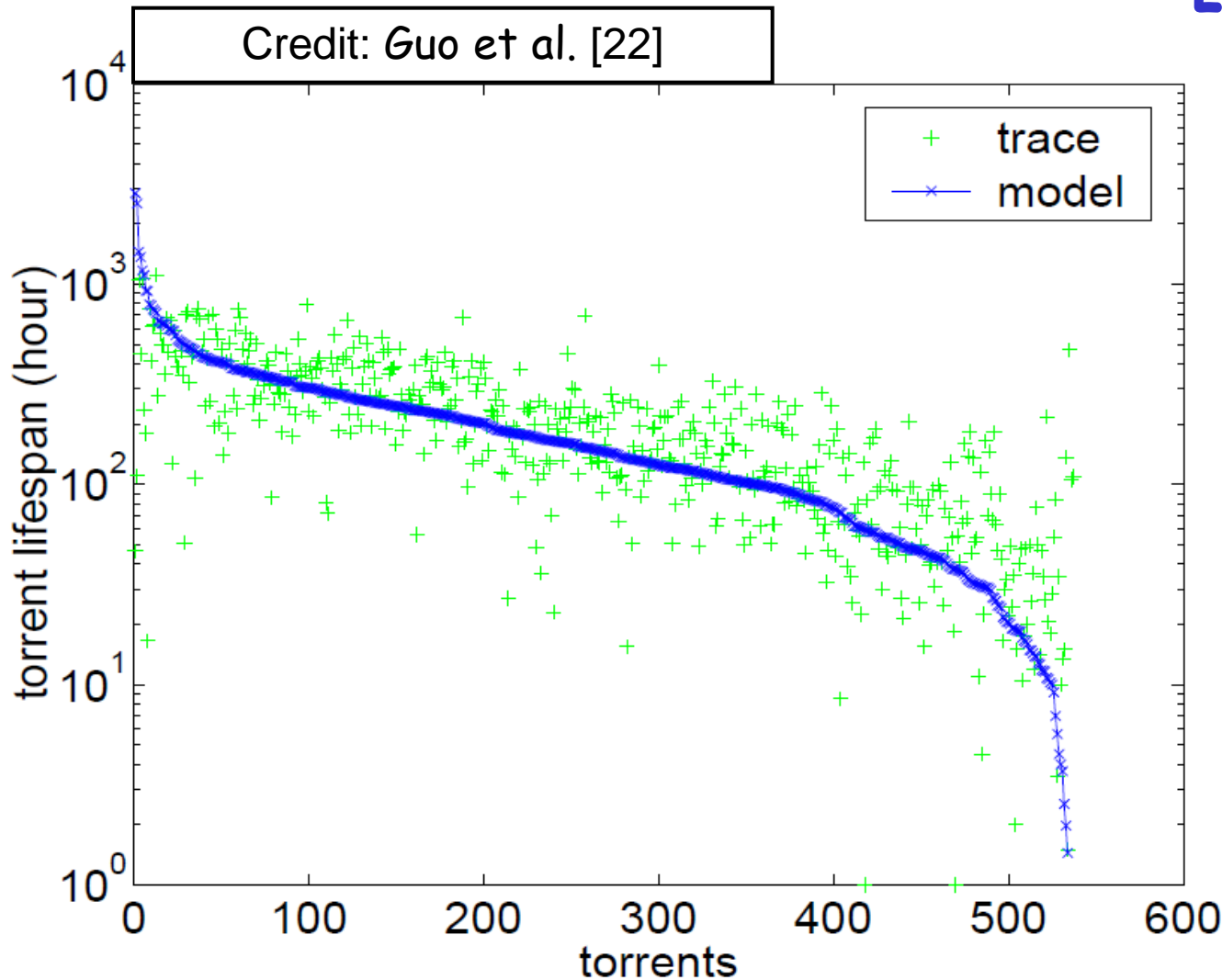


# Torrent Lifetime

## □ How long is a torrent alive?

- Can we extrapolate this result to any torrent?
  - All studies on copyrighted contents: high incentive to do not stay as a seed
  - But, even for legal content availability may be poor as if many contents are downloaded, not all can be seeded by the peer at the same time
- Torrent availability must be provided by a dedicated infrastructure

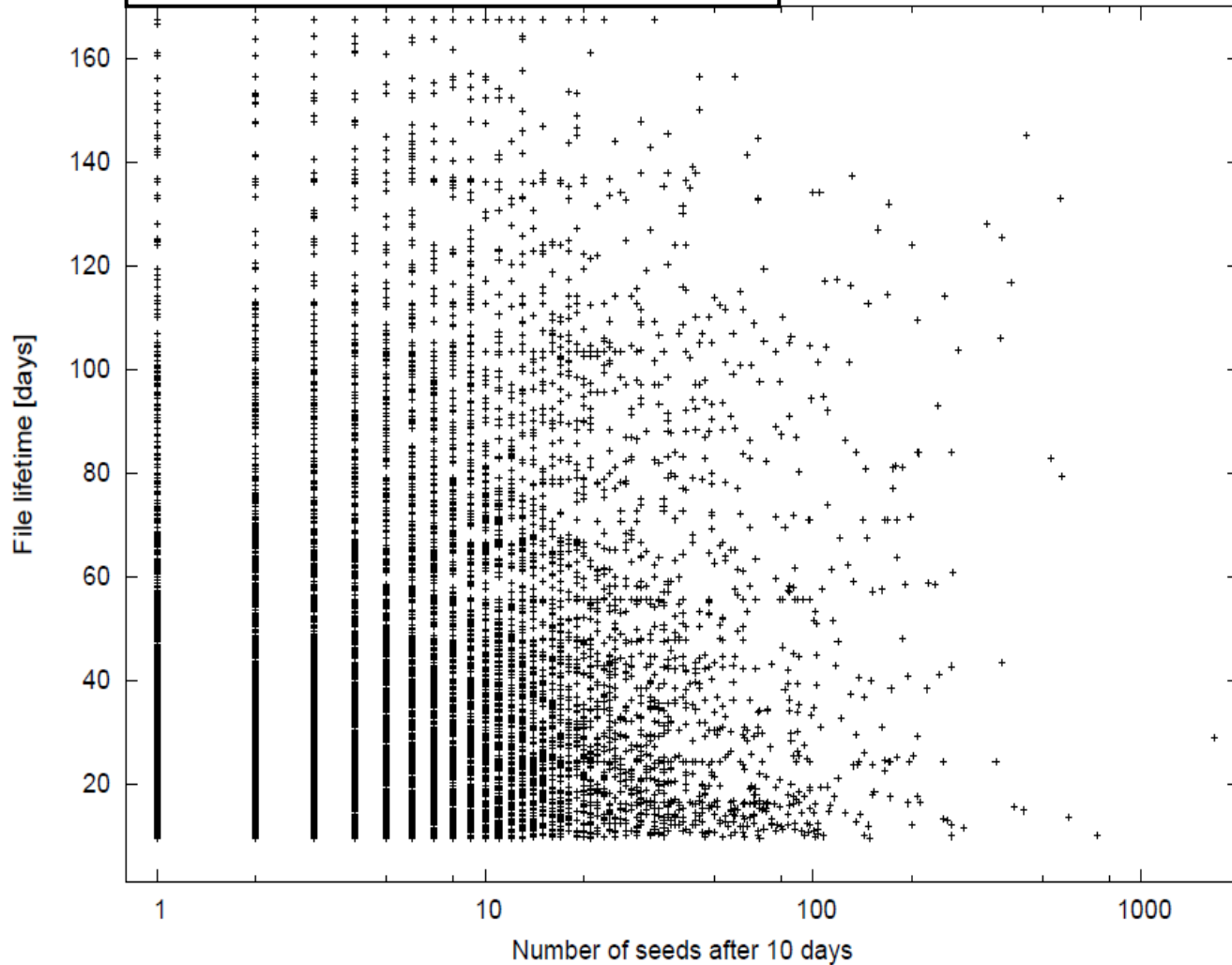
# Torrent Lifetime [22]



- Most torrents between 30 and 300 hours
- Mean: 8.89 days
- Trace collected from a single tracker (550 torrents) during 48 days end of 2003

# Torrent Lifetime [20]

Credit: Pouwelse et al. [20]



□ Number of seeds after 10 days is not an accurate predictor of the file lifetime

# Torrent Lifetime [20]

## □ Possible reasons

- The initial seed mainly determines the life time of the torrent
  - Therefore, the number of seeds after 10 days does not bring any information
  - Check whether there is still the initial seed?
- One seed mainly determines the life time of the torrent
  - Check how long each seed is a seed

# Torrent Lifetime [20]

## □ Possible reasons

- The initial seed might just have sent the last piece at day 10
  - Many seeds are due to leechers completing that will leave soon
  - Need 10 days to seed a content of 1 GB at 9.7kbit/s

# Outline

- Overview
- Content Replication
- BitTorrent
  - Overview
  - Algorithm details
  - Evaluation
    - Torrent Scale
      - Tracker log analysis
      - Large scale crawl
    - Algorithms
- Advanced subjects
- Security
- Localization

# Focus of This Section

- Give up-to-date results on BitTorrent large scale measurements
  - [42] from July 2008 to May 2009
  - [43] December 2008
- Introduce large scale crawl techniques
  - Do not need access to tracker logs
    - Most trackers do not keep any logs
  - Not linked to a specific tracker or site

# Large Scale BitTorrent Crawl

- Three main components to crawl
  - Torrent discovery sites (Mininova, ThePirateBay, Isohunt, Torrent Reactor, BTmonster, Torrent Portal, etc.)
    - .torrent files, meta information (comments, uploader, etc.)
  - Trackers (ThePirateBay, Mininova, etc.)
    - Scrape information, list of (IP,port) per infohash
  - Peers
    - Contribution (free rider?), upload/download speed



# BitTorrent Ecosystem [42]

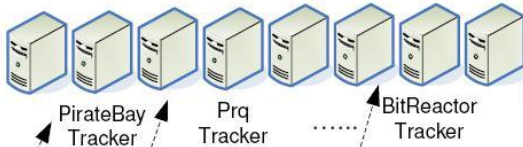
Public Discovery Sites



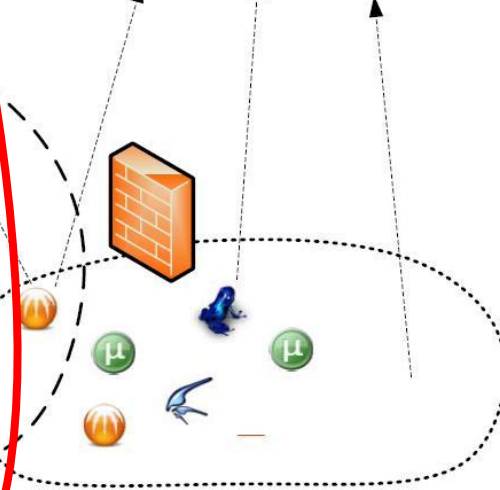
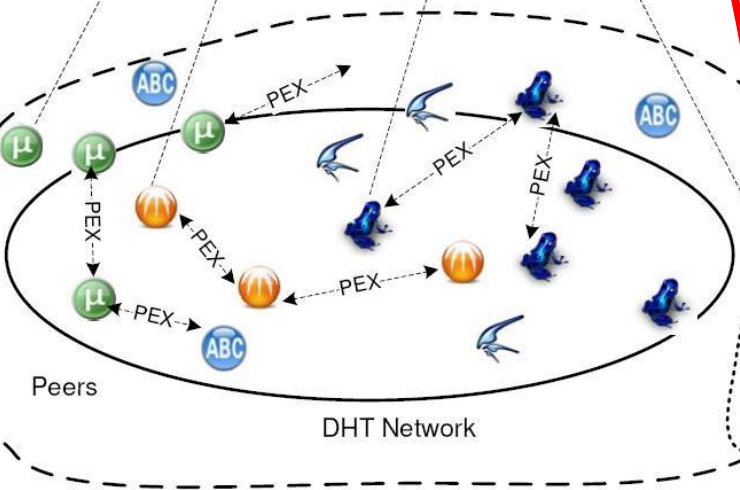
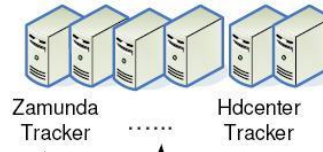
Private Discovery Sites



Public Trackers



Private Trackers



- Azureus
- Mainline
- uTorrent
- Xunlei
- BitComet
- ABC
- Tribler

Public Torrents

Private Torrents

Credit: Zhang et al. [42]

Focus of [42]

240

# Torrent discovery sites [42]

- ❑ All sites are web portals
- ❑ Most sites also provide .torrent files
- ❑ Some sites also provide a tracker infrastructure
- ❑ Much different from the supernova infrastructure
  - Today sites have a dedicated heavy infrastructure

# Most Popular Torrent Discovery Sites [42]

Site	Alexa rank	Location	Comment
Mininova.org	88	Netherlands	Tracker
Thepiratebay.org	109	Sweden	Tracker
IsoHunt.com	219	Canada	-
Torrentz.com	225	Netherlands	No .torrent
Torrentreactor.net	454	Germany	-
Btjunkie.org	551	Sweden	-
Sumotorrent.com	1,271	Netherland	Tracker
Btmon.com	1,410	Germany	-
TorrentPortal.com	1,525	United States	-
GamesTorrents.com	2,247	Germany	-

TABLE I  
TOP-10 MOST POPULAR PUBLIC ENGLISH-LANGUAGE  
TORRENT-DISCOVERY SITES

Credit: Zhang et al. [42]

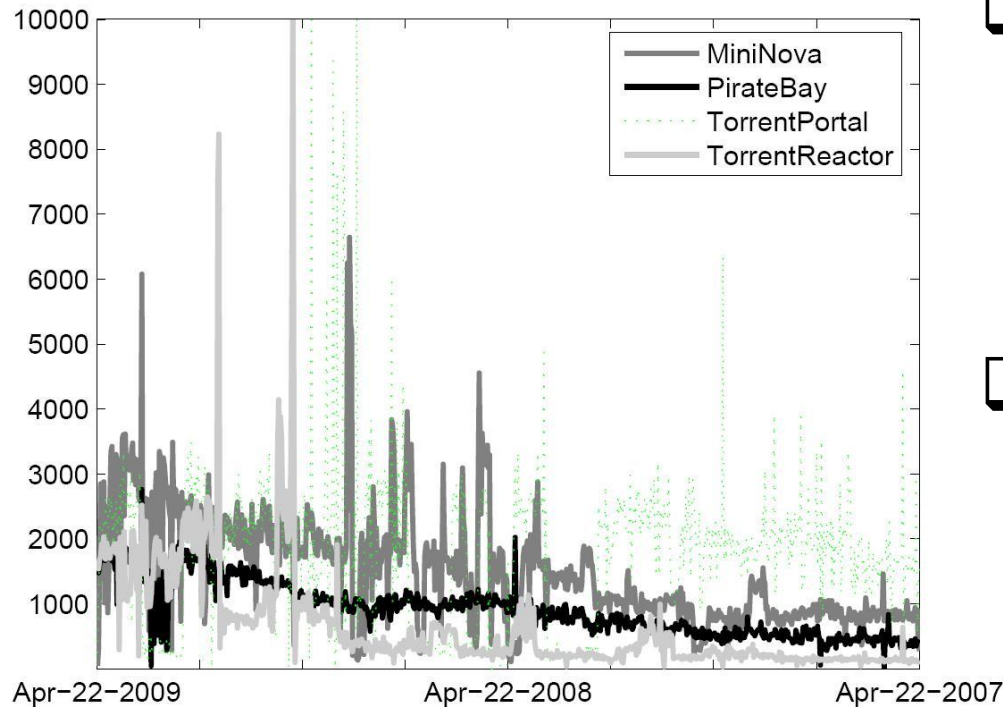
# Torrent Discovery Sites Crawl [42]

- Torrent discovery sites (Mininova, ThePirateBay, Torrent Reactor, BTmonster, Torrent Portal)
  - Crawl once the entire site (huge amount of data)
  - Then just get new torrents: all sites have a new torrent web page from which one can extract what is new

# Torrent Discovery Sites Crawl [42]

- In [42] they crawled 4.6 million unique infohash on nine months (july 2008-may 2009)
  - 8.8 million .torrent, but some overlap (different .torrent file for the same infohash)
  - Only 1.2 million torrents are active
    - At least one peer

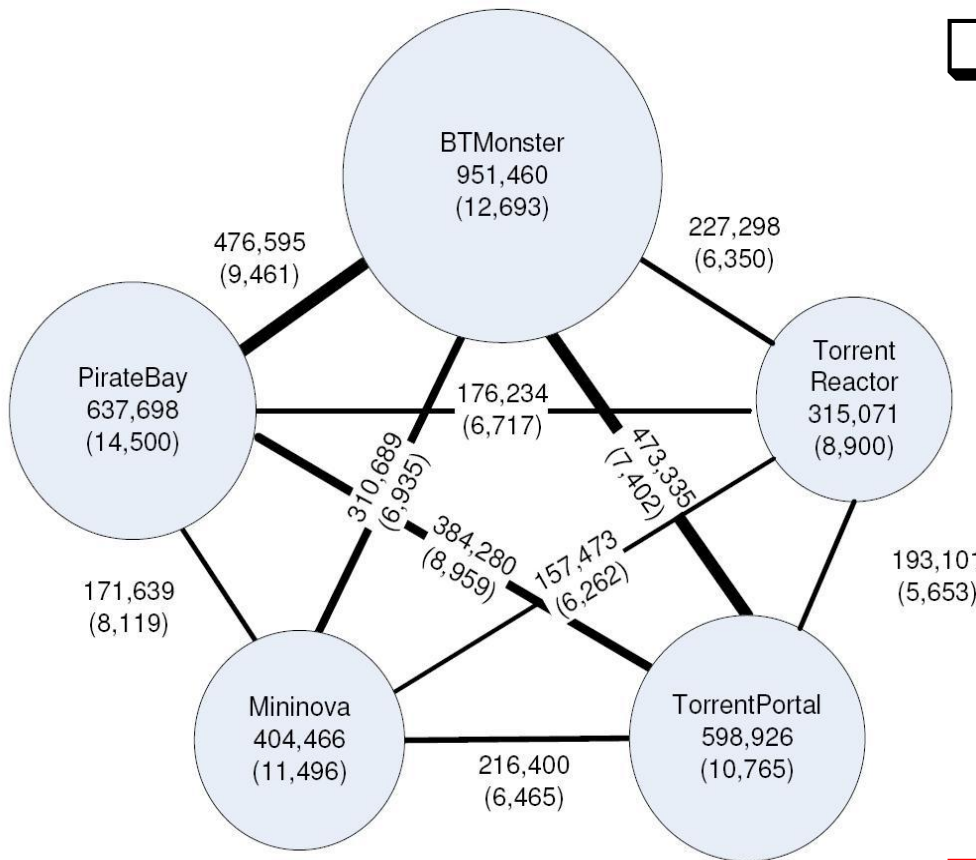
# New .torrent File per Day [42]



Credit: Zhang et al. [42]

- There is currently a few thousands of new .torrent files indexed per day
- .torrent are either crawled (from other sites) or inserted by an **uploader**

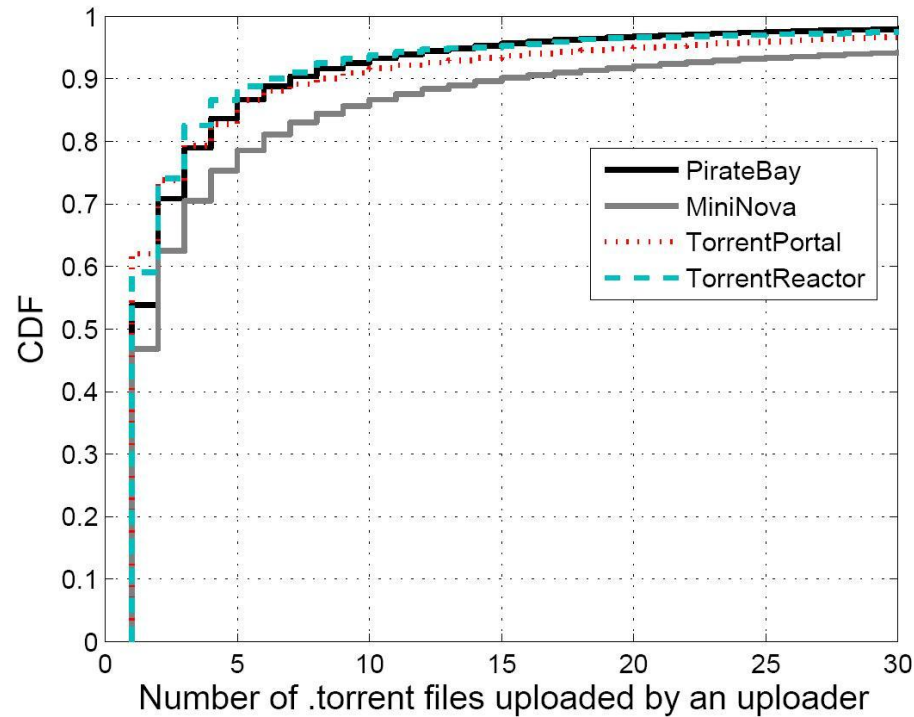
# Redundancy of Torrent Discovery Sites [42]



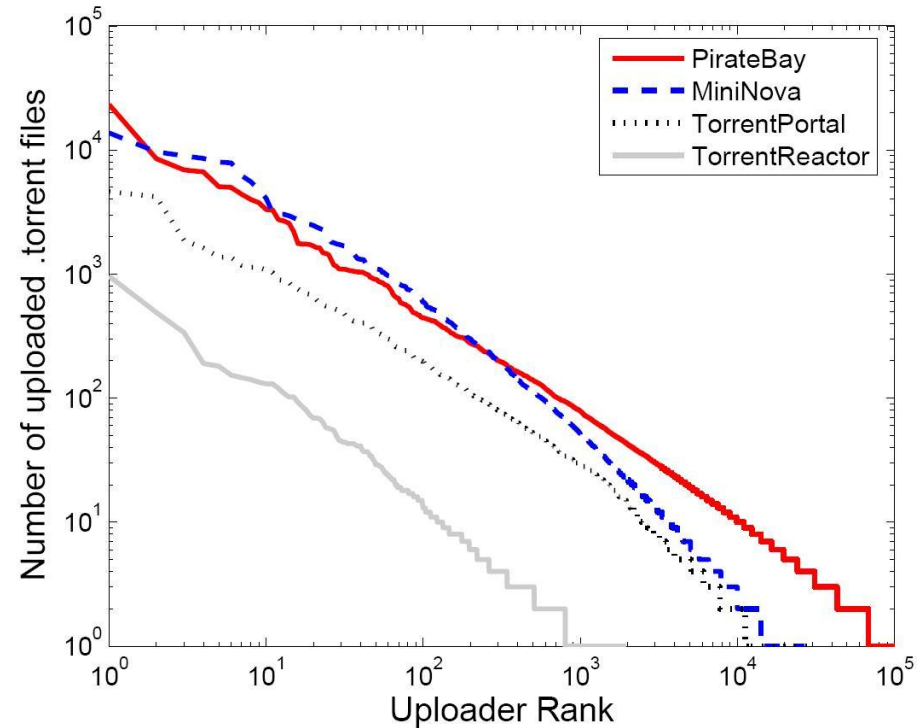
Credit: Zhang et al. [42]

- ❑ Pairwise intersection of active torrents (at least one peer) and highly active torrents (at least 100 peers) shown in paratheses
- ❑ Large overlap

# Uploaders Activity [42]



Credit: Zhang et al. [42]



Credit: Zhang et al. [42]

□ Uploader: user that inserted the .torrent file in the torrent discovery site. Identified by its login



# Trackers Crawl

- Reminder: a infohash is known by a tracker when the tracker URL is in the .torrent file for this torrent and at least one peer contacted the tracker for this torrent
  - The tracker don't know any information on the torrent, only its infohash and the peers (IP, port) currently on that torrent

# Trackers Crawl: Scrape

- Trackers provide **scrape** information
  - Specific request to the tracker
  - (infohash, # seeds, #leechers) for all infohash hosted by this tracker
  - Very large amount of data (several GBytes)

# Trackers Crawl: Scrape

## □ Usage of this crawl

- Aggregate statistics per torrent
- Discover infohash not referenced in torrent discovery sites
- # seed + # leechers used as a stop criterion for (IP,port) crawls

# Trackers Crawl: (IP,port)

- Trackers provide all (IP,port) per torrent
  - Trackers return by default at random 50 (IP,port), but they return up to 200 (IP,port) when asked for (set number of peers wanted to 200 in the request sent to the tracker)
  - For large torrents need to request several times the tracker

# Trackers Crawl: (IP,port)

- ❑ How to stop a tracker crawl for a torrent?
  - You can capture churn instead of a snapshot of peers
- ❑ Stop criteria
  - Number of discovered peers equals the number of peers found with a scrape request
  - No new peer is discovered in two consecutive requests

# Trackers Crawl: (IP,port)

## □ Usage of this crawl

- Get an accurate picture of peers identified by (IP,port) for each torrent hosted by the tracker
- (IP,port) gives
  - Geographical information
  - AS information
  - Follow up with time

# Trackers Statistics [42]

- 38,996 different trackers found in the in the 8.8 million crawled .torrent files
  - Only 728 are active (with at least one active torrent)
  - Most BitTorrent clients (uTorrent, Vuze) allows to create trackers. Thus, most trackers are ephemeral

# Most Popular Trackers [42]

Tracker organization	Number of Tracked Peers	Number of Tracked Torrents	Location
thepiratebay.org	12,883,329	1,025,864	Sweden
rarbg.com	949,584	88,000	Sweden
torrent.to	540,476	44,084	Luxembourg
bitreactor.to	346,521	17,371	Sweden
mightynova.com	313,743	41,782	United States
torrent-downloads.to	269,104	26,776	Netherlands
paradise-tracker.com	253,851	14,827	Netherlands
torrent-download.to	245,952	14,386	Netherlands
tntvillage.org	212,319	17,943	Czech Republic
bittorrent.am	182,472	6,998	Russian Federation
9you.com	172,349	12,996	China
sharego.net	118,226	3,120	Netherlands
1337x.org	108,064	3,666	Sweden
divxfinal.com	107,979	3,520	Spain
megashara.com	94,135	3,692	Russian Federation
taquilladivx.com	93,147	1,243	Sweden
smartorrent.com	77,081	801	Canada
nyaatorrents.org	76,023	9,373	Netherlands
spanishtracker.com	62,680	5,554	Netherlands
xuntv.cn	54,013	11,002	China

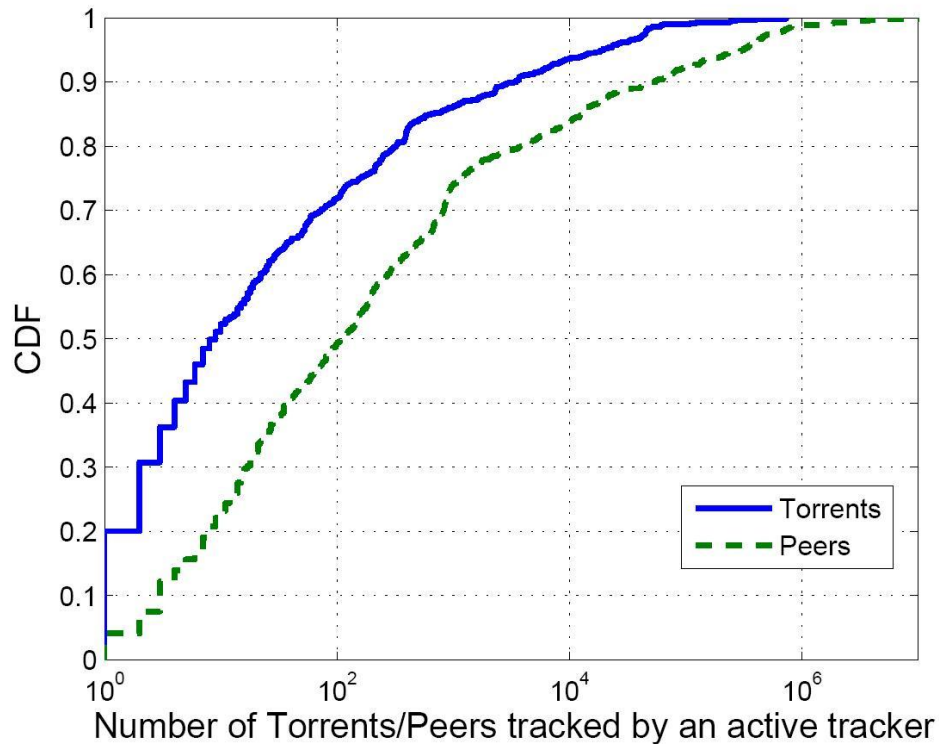
TABLE IV

TOP 20 TRACKER ORGANIZATIONS RANKED BY THE NUMBER OF TRACKED PEERS

Credit: Zhang et al. [42]



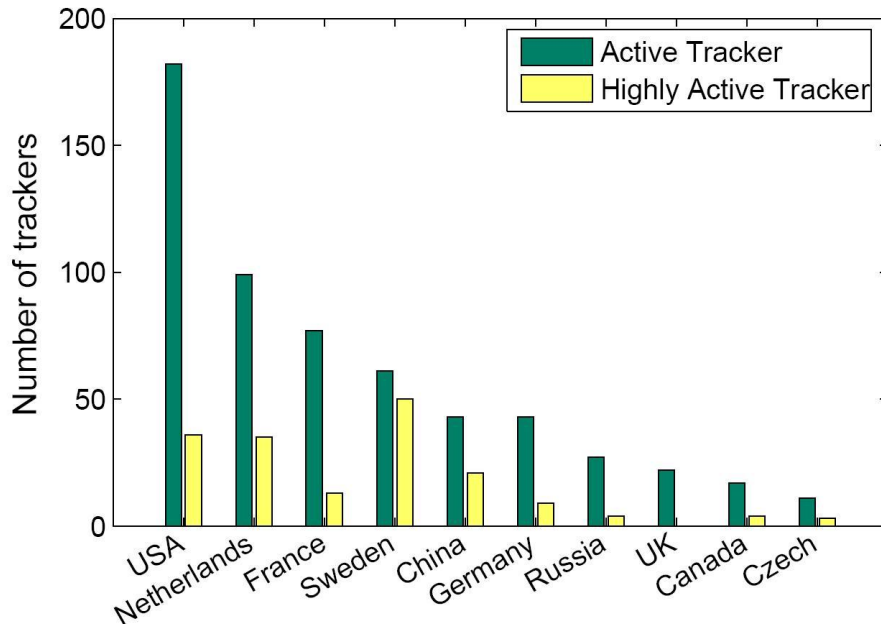
# Most Popular Trackers [42]



Credit: Zhang et al. [42]

- 26% (190) of the trackers track more than 1000 peers
- 28% tracks more than 100 torrents

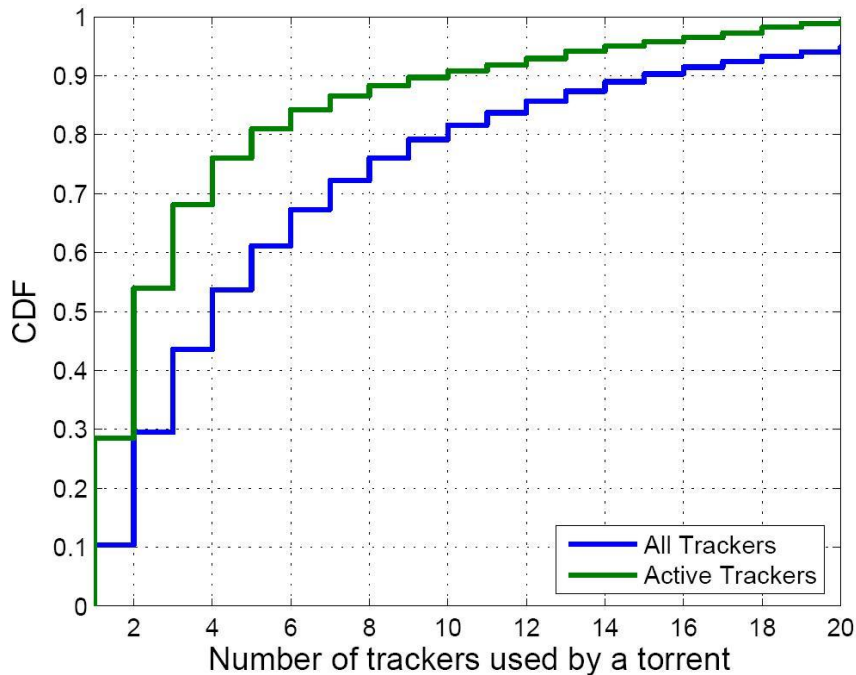
# Tracker Location [42]



Credit: Zhang et al. [42]

- ❑ Active tracker: at least on active torrent
- ❑ Highly active tracker: at least 1000 peers
- ❑ Sweden has the largest number of highly active trackers (ThePirateBay)

# Number of Trackers per .torrent [42]



Credit: Zhang et al. [42]

- A large number of tracker URLs improves torrent reliability in case of tracker failure
  - BT Clients test URLs sequentially until one works
- 71% of torrents are tracked by at least 2 active trackers

# Tracker Crawl (IP,port) [42]

## □ Methodology

- Performed on 22 April 2009
- Single snapshot captured on 12 hours

## □ Collected 5 millions unique (IP,port)

- Corresponding to 1 million torrents

# How to Uniquely Identify a Peer in Crawls?

- ❑ Modern trackers do not return a Peer ID, but only (IP,port)
  - ❑ Compact mode
    - ❑ Reduce bandwidth overhead on trackers
- ❑ NAT and dynamic IP addresses prevent using an (IP,port) as a unique identifier of a peer
- ❑ Most popular BT client multiplex all torrents on a single port chosen at random at client installation
  - ❑ Adding the port to IP improves uniqueness

# How to Uniquely Identify a Peer in Crawls?

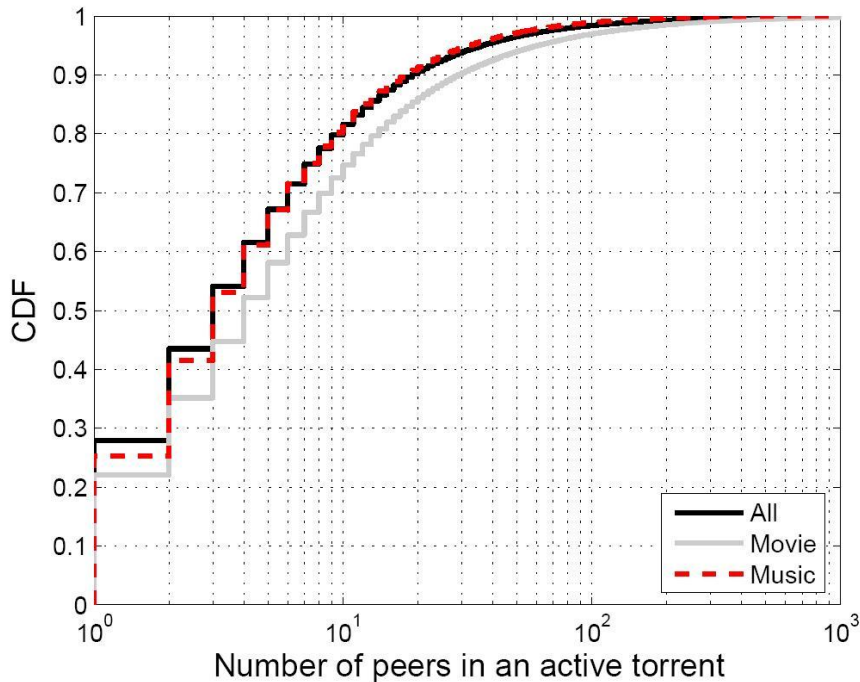
- For a single snapshot
  - (IP,port) is a reasonable unique identifier
  - Two peers cannot share the same (IP,port) at the same time even with NAT and dynamic IP addresses
  - In reality snapshots are not instantaneous
    - May take a few hours to capture

# How to Uniquely Identify a Peer in Crawls?

## □ For multiple snapshots

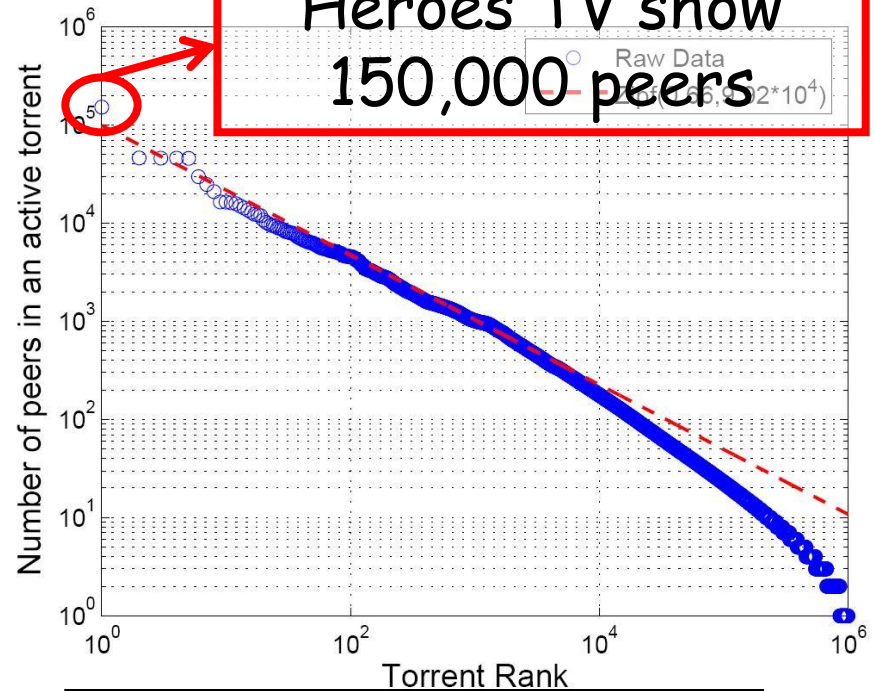
- Two different (IP,port) can be the same peer at different moment in time
- One (IP,port) can represent two different peers at different moment in time
- Hard to conclude in that case
  - Need clever heuristics
    - Content correlation with time
    - Statistical analysis of port distribution per IP

# Number of Peers per Torrent [42]



Credit: Zhang et al. [42]

□ 1% of torrents have more than 100 peers

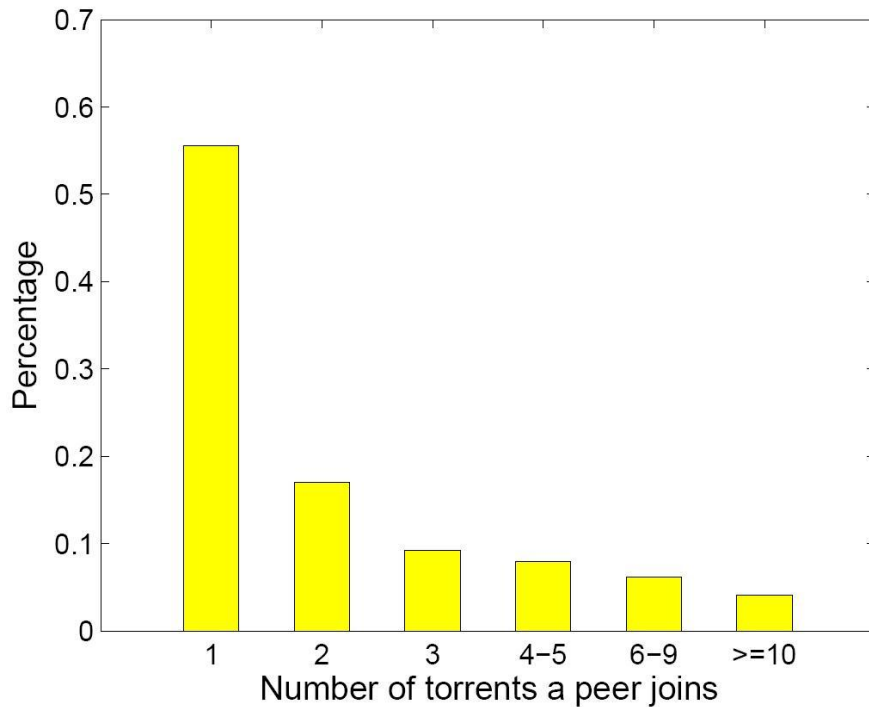


Credit: Zhang et al. [42]

□ 82% of the torrents have less than 10 peers



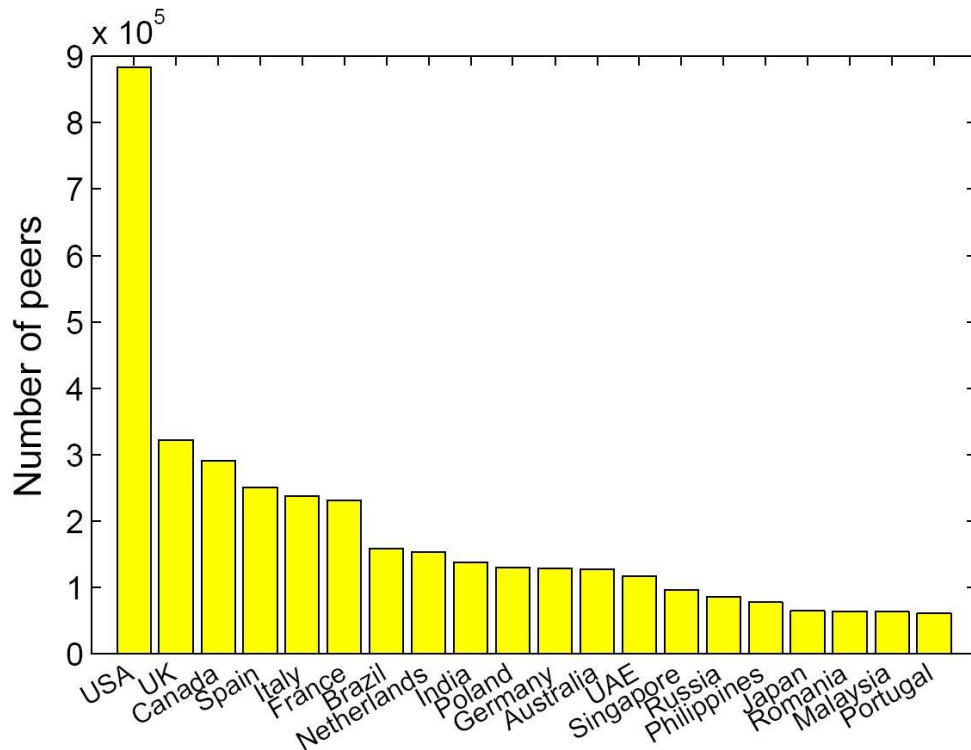
# Number of Parallel Torrents per Peer



Credit: Zhang et al. [42]

- Parallel torrents for the 12 hours snapshot
- Only 4% of peers join more than 10 torrents at the same time

# Distribution of Peer per Country

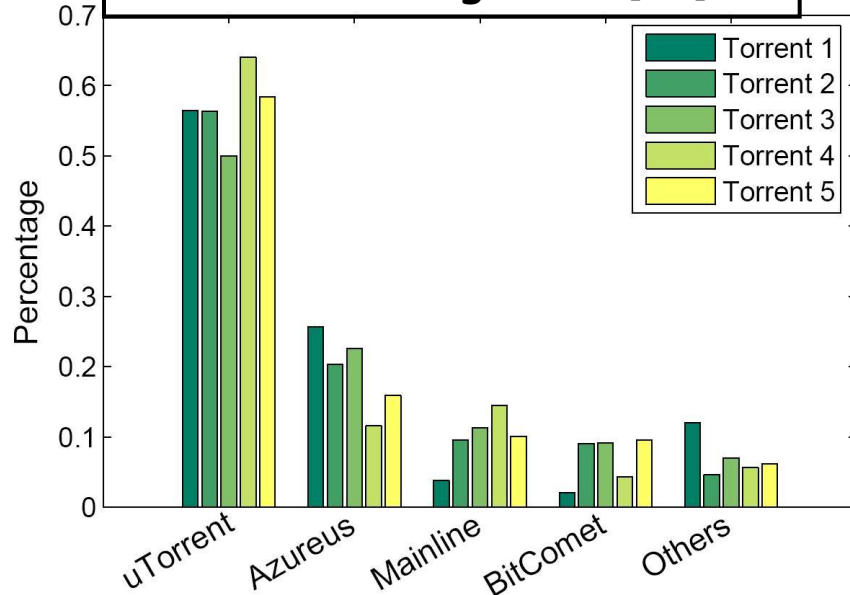


Credit: Zhang et al. [42]

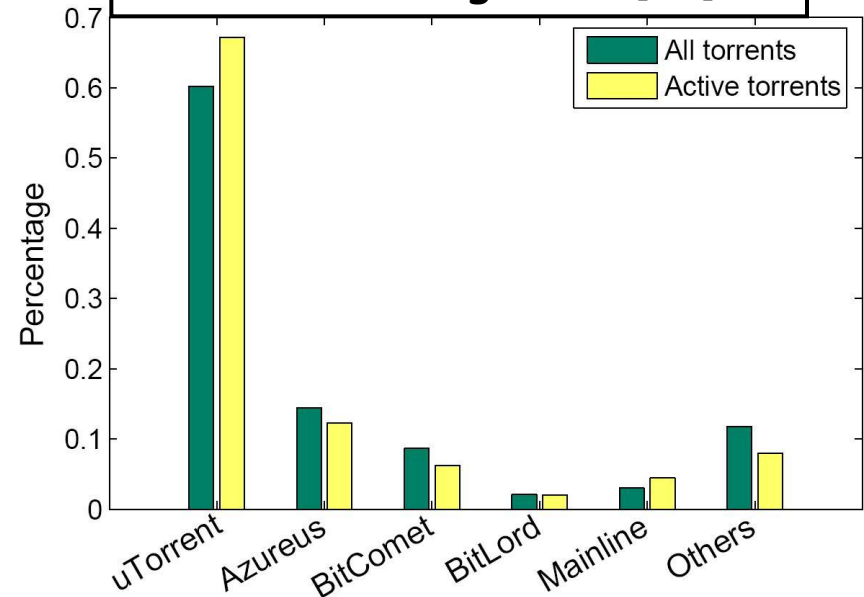
- Diurnal pattern might bias this result
  - No information of the start time of the 12 hours period

# BitTorrent Clients Popularity

Credit: Zhang et al. [42]



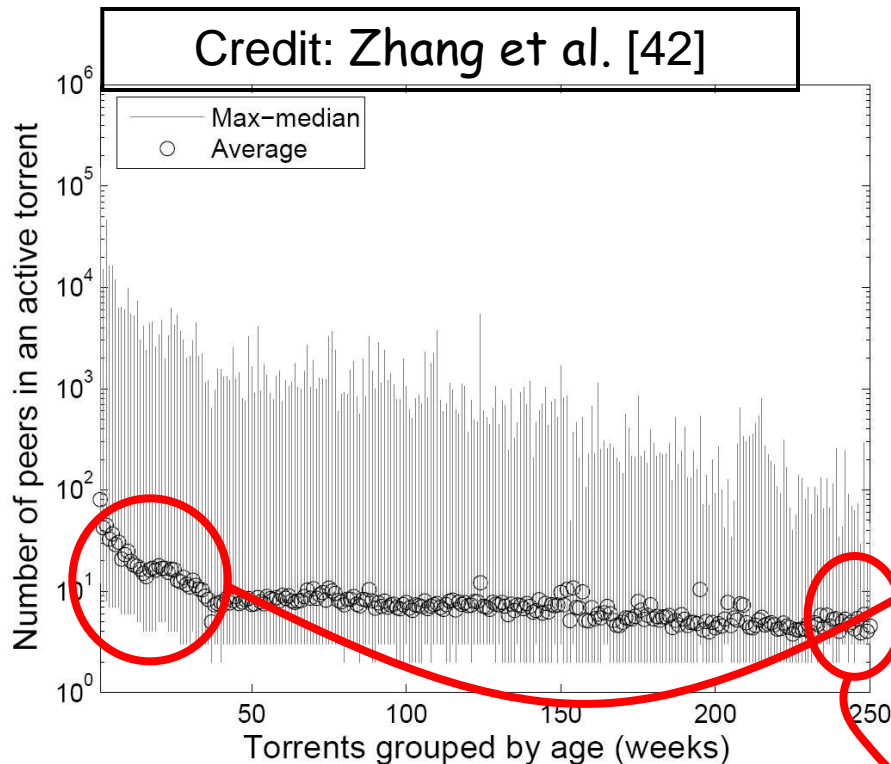
Credit: Zhang et al. [42]



- BT clients popularity for content sharing
- 5 torrents selected at random with different size and content type

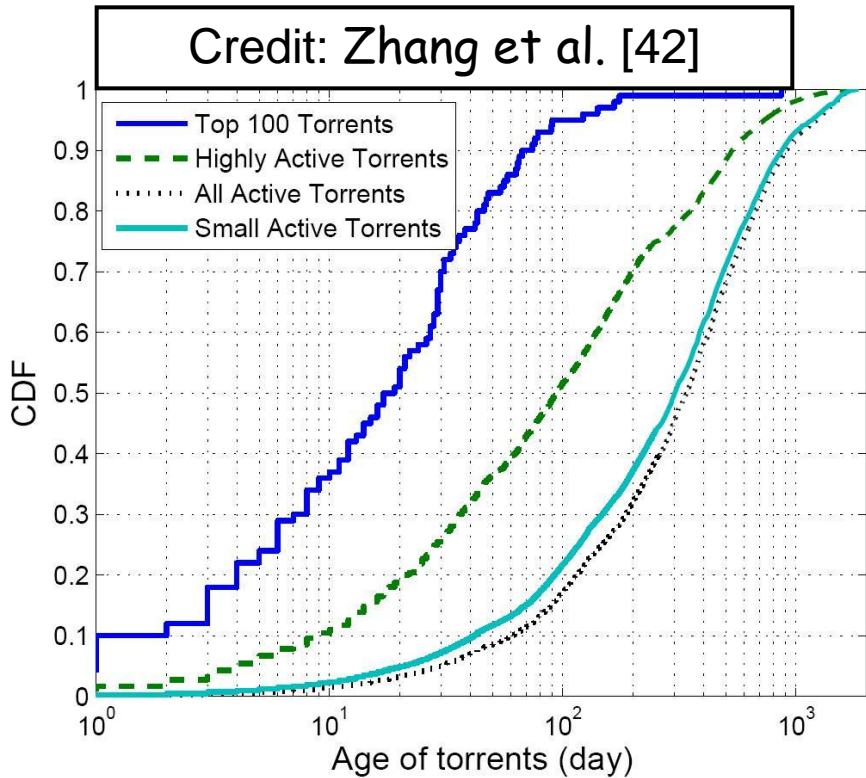
- BT clients popularity to create .torrent files

# Correlation Torrent Age and Number of Peers



- ❑ Torrents are grouped by periods of 1 week
  - Torrents appearing the same week are grouped together
- ❑ Number of peers depends on the age up to 40 weeks
- ❑ Torrents still alive after 250 weeks

# Torrents Age Distribution



## Legend

- Top 100 torrents
  - largest torrents
- Highly active
  - More than 100 peers
- Small active
  - Lower than 10 peers

## Median age

- Top 100: 20 days
- Highly active: 100 days
- Active: 300 days

# Conclusion on Crawls

- ❑ It is very easy to crawl all public torrents in the Internet
  - Privacy issues
- ❑ Current torrent discovery sites and trackers do not take any step to solve those crawling issues
  - They want to be crawled for replication and indexing

# Outline

- Overview
- Content Replication
- BitTorrent
  - Overview
  - Algorithm details
  - Evaluation
    - Torrent Scale
    - Algorithms
- Advanced subjects
- Security
- Localization

Study 1 [18,34]



# Why Studying BitTorrent Peer and Piece Selection?

- ❑ Implemented in all BitTorrent clients
  - Very popular protocol
  - Large fraction of the internet traffic
  - Focus on efficient data dissemination
- ❑ Very simple algorithms
  - Fast to compute
  - Minimal state
  - Easy to implement

# Why Studying BitTorrent Peer and Piece Selection?

- ❑ But, doubts on the efficiency
- ❑ Rarest first
  - Poor pieces diversity (in specific scenarios) resulting in low efficiency
- ❑ Proposed solutions
  - Source coding: Bullet' (Kostic et al.)
  - Network coding: Avalanche (Gkantsidis et al.)

# Why Studying BitTorrent Peer and Piece Selection?

## □ Choke algorithm

- Unfair
- Favors free riders

## □ Proposed solutions

- Based on strict byte reciprocation

## □ Do we see the claimed deficiencies in real torrents?

## □ Study [34], some results come from [18]

# Methodology: Experiments

- Instrumented a BitTorrent client (mainline 4.0.2)
  - In 2009, second most downloaded BT client at SourceForge (51 millions downloads)
    - Azureus was the first one (135 millions downloads), second most downloaded soft of all time at SourceForge (emule was the first one with 281 millions downloads)
  - Log
    - All messages
    - Seed state event
    - End game mode
    - Algorithms internals
    - Bandwidth estimators
  - Use default parameters (20kB/s upload)

# Methodology: Experiments

- ❑ Connected this client to real torrents
  - Single client to be unobtrusive
    - No assumption on the other real peers
  - Connected to 80 peers selected at random
- ❑ 8 hours experiments per torrent

# Methodology: Torrents

## □ Real torrents (26)

- Both free and copyrighted contents
  - TV series, movies, live concerts, softwares
- Large variety in the number of seeds and leechers
  - 0 seed, 66 leechers
  - 1 seed, 1411 leechers (low seed to leecher ratio)
  - 160 seeds, 5 leechers
  - 12612 seeds, 7052 leechers

# Methodology: Torrents

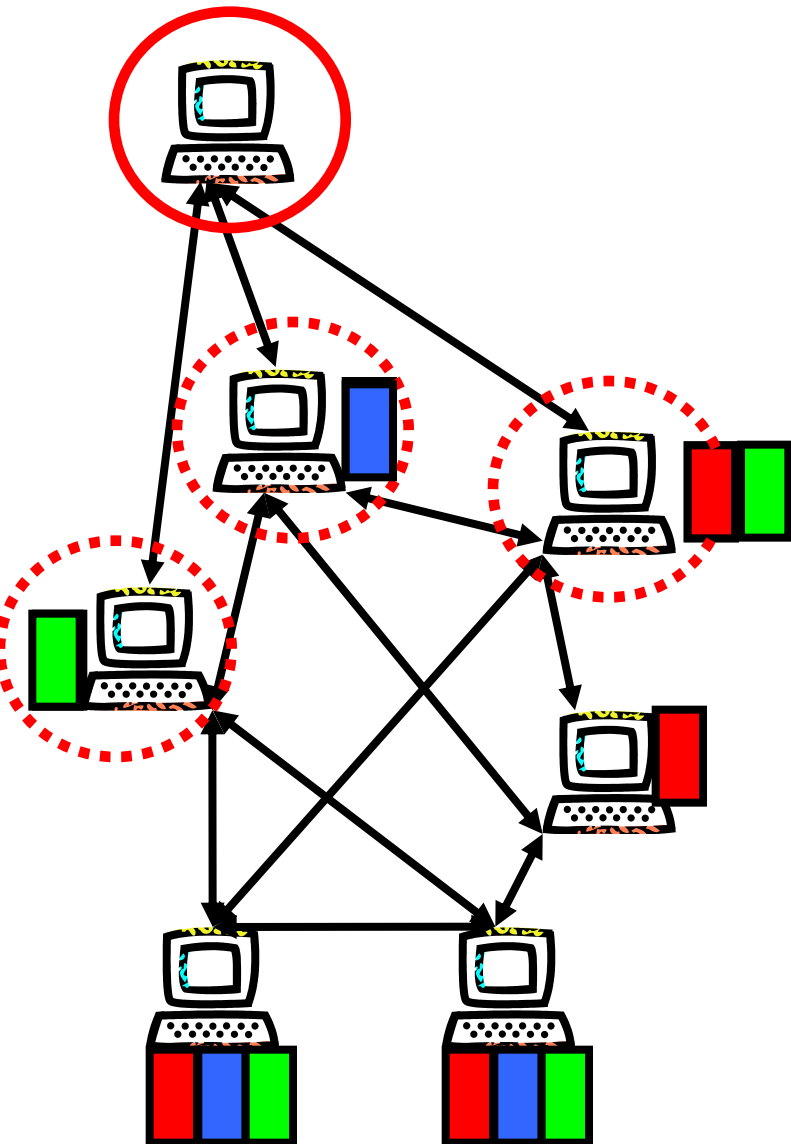
<b>ID</b>	<b># of S</b>	<b># of L</b>	<b>Ratio <math>\frac{S}{L}</math></b>	<b>Max. PS</b>	<b>Size</b>
1	0	66	0	60	700
2	1	2	0.5	3	580
3	1	29	0.034	34	350
4	1	40	0.025	75	800
5	1	50	0.02	60	1419
6	1	130	0.0078	80	820
7	1	713	0.0014	80	700
8	1	861	0.0012	80	3000
9	1	1055	0.00095	80	2000
10	1	1207	0.00083	80	348
11	1	1411	0.00071	80	710
12	3	612	0.0049	80	1413
13	9	30	0.3	35	350
14	20	126	0.16	80	184
15	30	230	0.13	80	820
16	50	18	2.8	40	600
17	102	342	0.3	80	200
18	115	19	6	55	430
19	160	5	32	17	6
20	177	4657	0.038	80	2000
21	462	180	2.6	80	2600
22	514	1703	0.3	80	349
23	1197	4151	0.29	80	349
24	3697	7341	0.5	80	349
25	11641	5418	2.1	80	350
26	12612	7052	1.8	80	140

# Methodology: Limitations

- ❑ Single client instrumentation
  - Partial view
  - Is it representative?
    - ⇒ Unobtrusive, behavior of a new peer
- ❑ Only real torrents
  - No reproducibility
  - No statistics
    - ⇒ Work on a representative set



# Peer Interest



- Peer X is interested in peer Y if peer Y has at least 1 piece that peer X does not have

# Peer Availability

□ Peer availability of  $Y$  (according to peer  $X$ )

$\frac{\text{Time peer } X \text{ is interested in peer } Y}{\text{Time peer } Y \text{ spent in the peer set of peer } X}$

---

□ Peer availability=1

- $X$  is always interested in peer  $Y$

□ Peer availability=0

- $X$  is never interested in peer  $Y$

□ Peer availability=0.5

- $X$  interested in peer  $Y$  half of the time peer  $Y$  has spent in the peer set of peer  $X$

# Peer Availability

- What is the peer availability achieved by rarest first?
  - Peer availability is a characterization of piece entropy

# Ideal Piece Selection

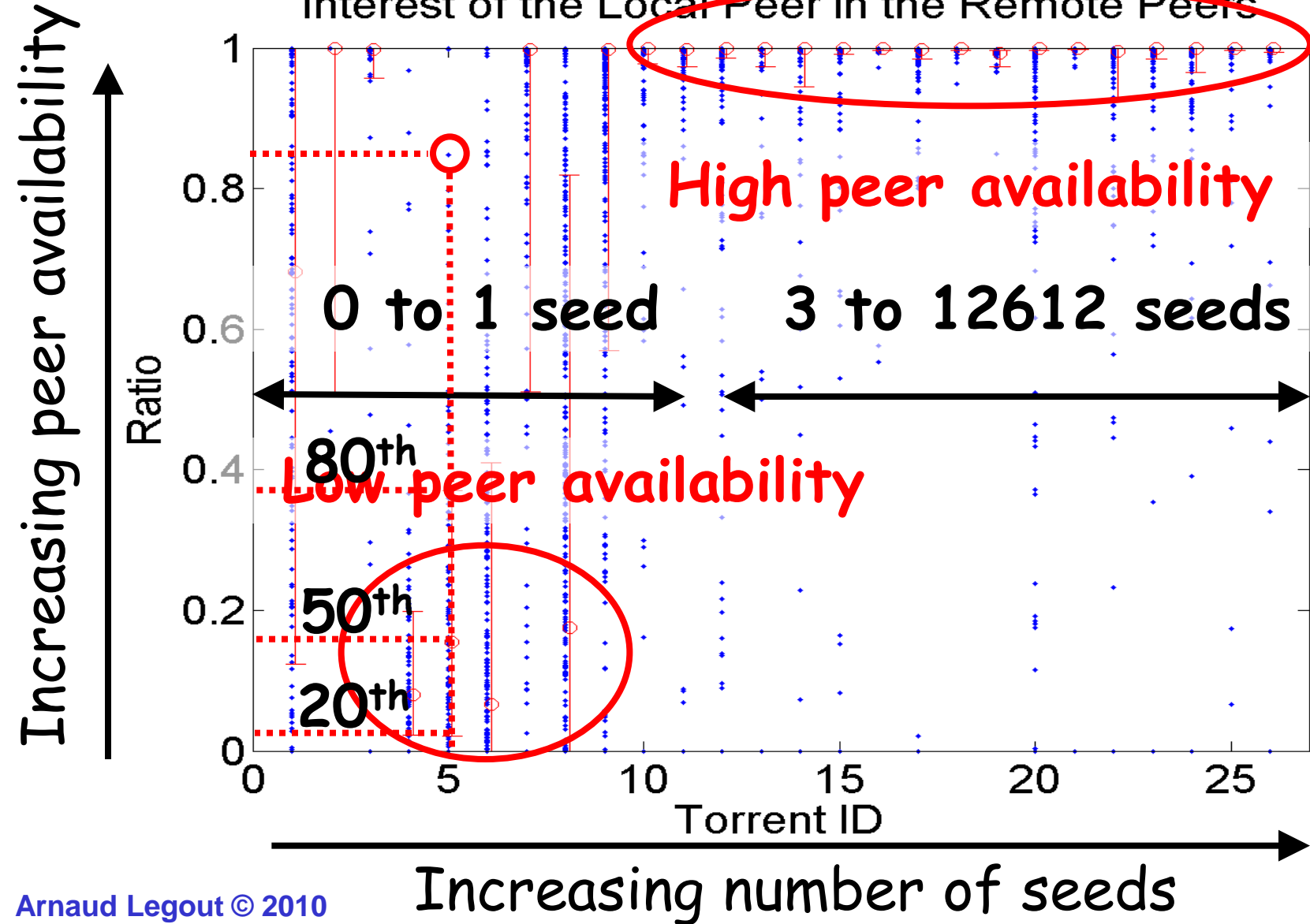
- For each peer  $X$  the peer availability of all peers  $Y$  (according to  $X$ ) must be 1
  - How far is rarest first to an ideal piece selection strategy?

# Peer Availability

- ❑ Local peer always in leecher state
  - Case of seeds not relevant for the peer availability
- ❑ Local peer point of view
  - Cannot conclude on the entire torrent
  - But, yet an important result

# Peer Availability

Interest of the Local Peer in the Remote Peers

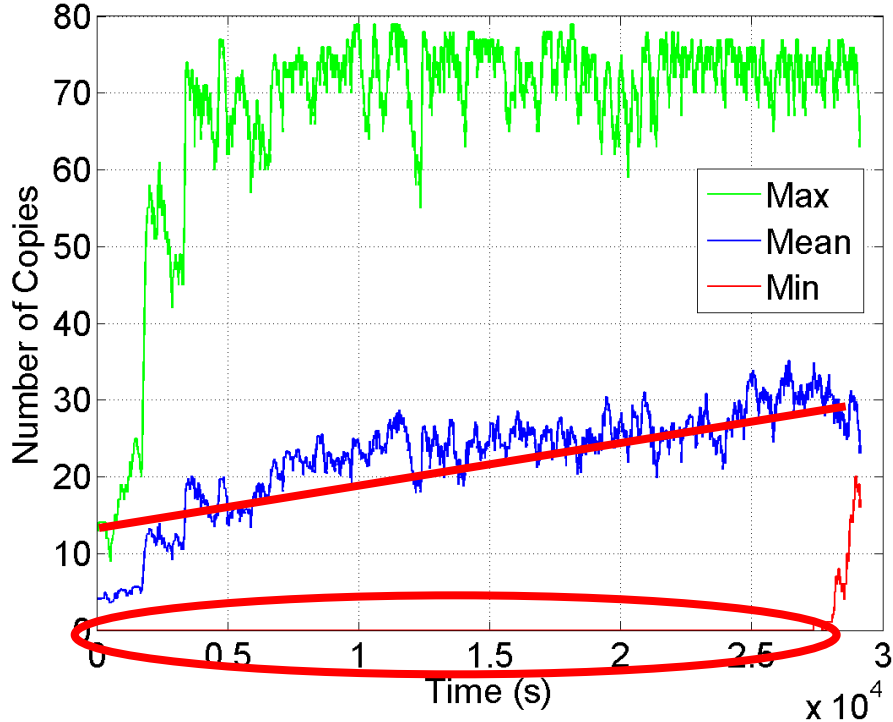


# Peer Availability

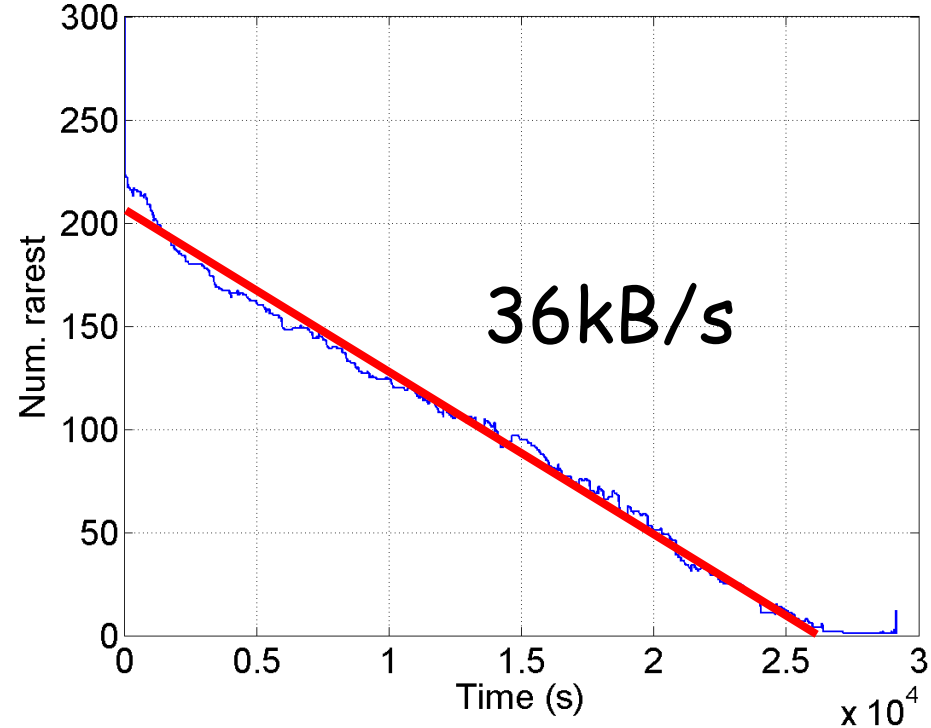
- ❑ Rare pieces
  - Pieces only on the initial seed
- ❑ Available pieces
  - Pieces with at least 2 copies in the torrent
- ❑ Poor peer availability for some torrents, but not all, with at most 1 seed
  - Torrents in transient state
    - Initial seed has not yet sent one copy of each piece (see next slide)
- ❑ Peer availability close to one for the others

# Deeper Look at Torrent 8

Replication of Pieces in the Peer Set, LS



Number of Rarest Pieces, LS



□ The initial seed has not yet sent one copy of each piece (transient state)

1 seed, 861 leechers, 863 pieces



# Deeper Look at Torrent 8

- There is 0 copy for the least replicated piece for most of the experiment
  - There are missing pieces
- Continuous increase of the mean number of copies
- Missing pieces served at the constant rate of 36kB/s
  - Likely it is the upload speed of the initial seed, but no guarantee

# Transient State

- ❑ Torrents with poor peer availability are in **transient state**
  - The initial seed has not yet sent one copy of each piece
  - Some pieces are rare (only present on the initial seed)
- ❑ Rare pieces served at the upload speed of the seed (constant rate)
- ❑ Other pieces (available pieces) served with a capacity of service increasing exponentially

# Peer Availability in Transient State

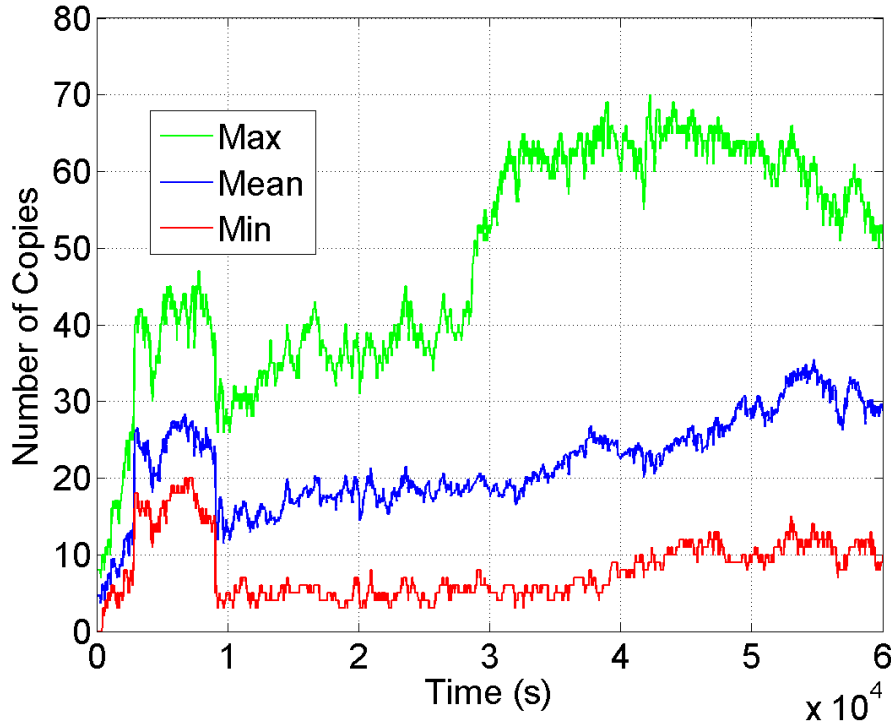
- Reason of the poor peer availability
  - Higher probability to have peers with the same subset of pieces as available pieces are replicated faster than rare pieces are injected in the torrent by the seed
  - Leecher with all the available pieces are not interested in any peer, except the initial seed

# Peer Availability in Transient State

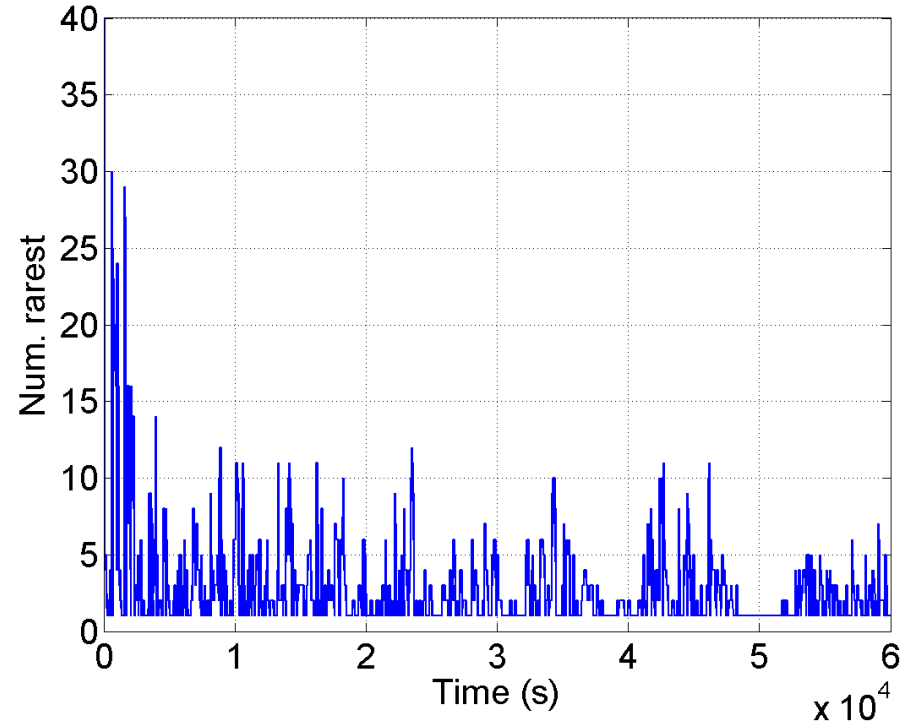
- This is a provisioning problem, not a piece selection problem
  - Cannot significantly improve on rarest first as the bottleneck is the upload speed of the initial seed
- Rarest first is an efficient piece selection strategy on real torrents
  - Network coding theoretically optimal in all cases, but more complex
  - Rarest first as efficient as network coding on real torrents (availability close to 1), but much simpler
    - Large peer set (80)

# Deeper Look at Torrent 7

Replication of Pieces in the Peer Set



Number of Rarest Pieces



□ The initial seed has already sent one copy of each piece (steady state)

1 seed, 713 leechers

# Deeper Look at Torrent 7

- ❑ Mean number of copies well bounded by min and max
- ❑ Closely follow the evolution of the peer set size
- ❑ Min curve closely follow mean but does not get closer
  - Is it a problem of rarest first?
- ❑ Consistent decrease in the number of rarest pieces
  - Each time a peer leave or join the peer set the rarest pieces set change

# Steady State

- Torrents with a high peer availability are in **steady state**
  - The initial seed has already served one copy of each piece
    - It is no more a bottleneck for the torrent
    - There is no rare piece

# Peer Availability in Steady State

## □ Rarest first algorithm

- Ensures a good replication of the pieces
- Prevents a return in transient state
  - Always replicate rare pieces

## □ Rarest first algorithm is enough to guarantee a high entropy for torrents in steady state



# Peer Availability: Conclusion

- Rarest first is enough to guarantee a high peer availability on the real torrents considered
  - The poor peer availability for torrents in transient state cannot be much improved using network coding because the bottleneck is the initial seed

# Peer Availability: Conclusion

- But, this is only a local view and a restricted number of torrents
  - Do not extrapolate those results to any case
    - Peer set of 80
    - Peers in the Internet: everybody can join everybody (except in case of NAT/firewall)
    - Medium to large contents
    - No guarantee that the peer availability is uniform in the torrent (even if it is likely)
  - This is a first step, but it already provides important results

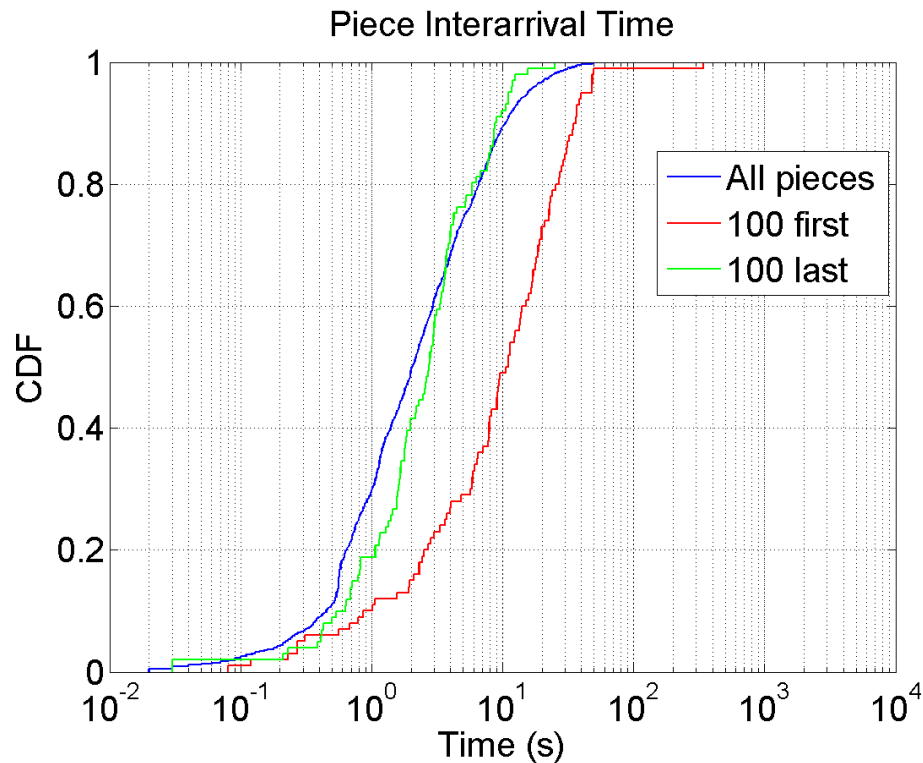
# Last Pieces Problem

□ Do we observe a last pieces problem?

# Last Pieces Problem

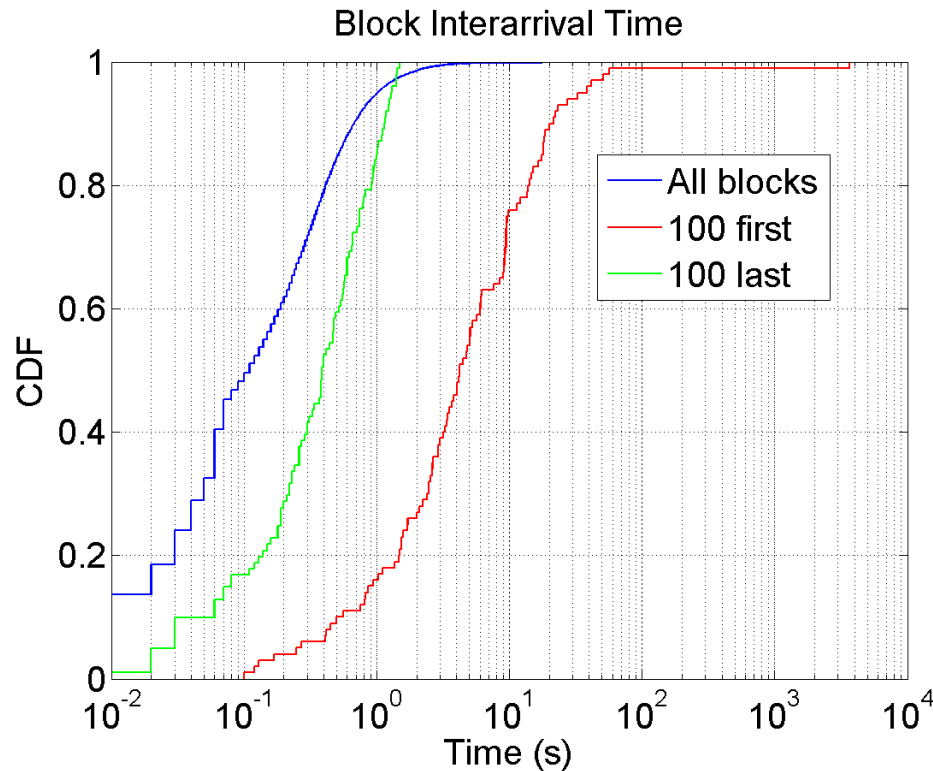
- ❑ Few pieces on some overloaded peers
  - Detected at the end of the download because fast peers are chosen first
  - End game mode does not help
- ❑ Is there a last pieces problem?
- ❑ Pieces are unit of replication, but blocks are unit of transmission
- ❑ Is there a last blocks problem?

# Last Pieces Problem: Torrent 10



- ❑ 1 seed, 1207 leechers, 1393 pieces
- ❑ No last pieces problem for torrents in steady state
- ❑ But, first pieces problem

# Last Blocks Problem: Torrent 10



- ❑ 1 seed, 1207 leechers, 1393 pieces
- ❑ No last blocks problem for torrents in steady state
- ❑ But, first blocks problem

# Last Pieces or Blocks Problem: Conclusion

- ❑ No last pieces or blocks problem
  - Appears rarely on torrents in transient state
    - Last pieces on the initial seed only
- ❑ First blocks problem
  - Slow startup phase
  - Area of improvement in particular for small contents

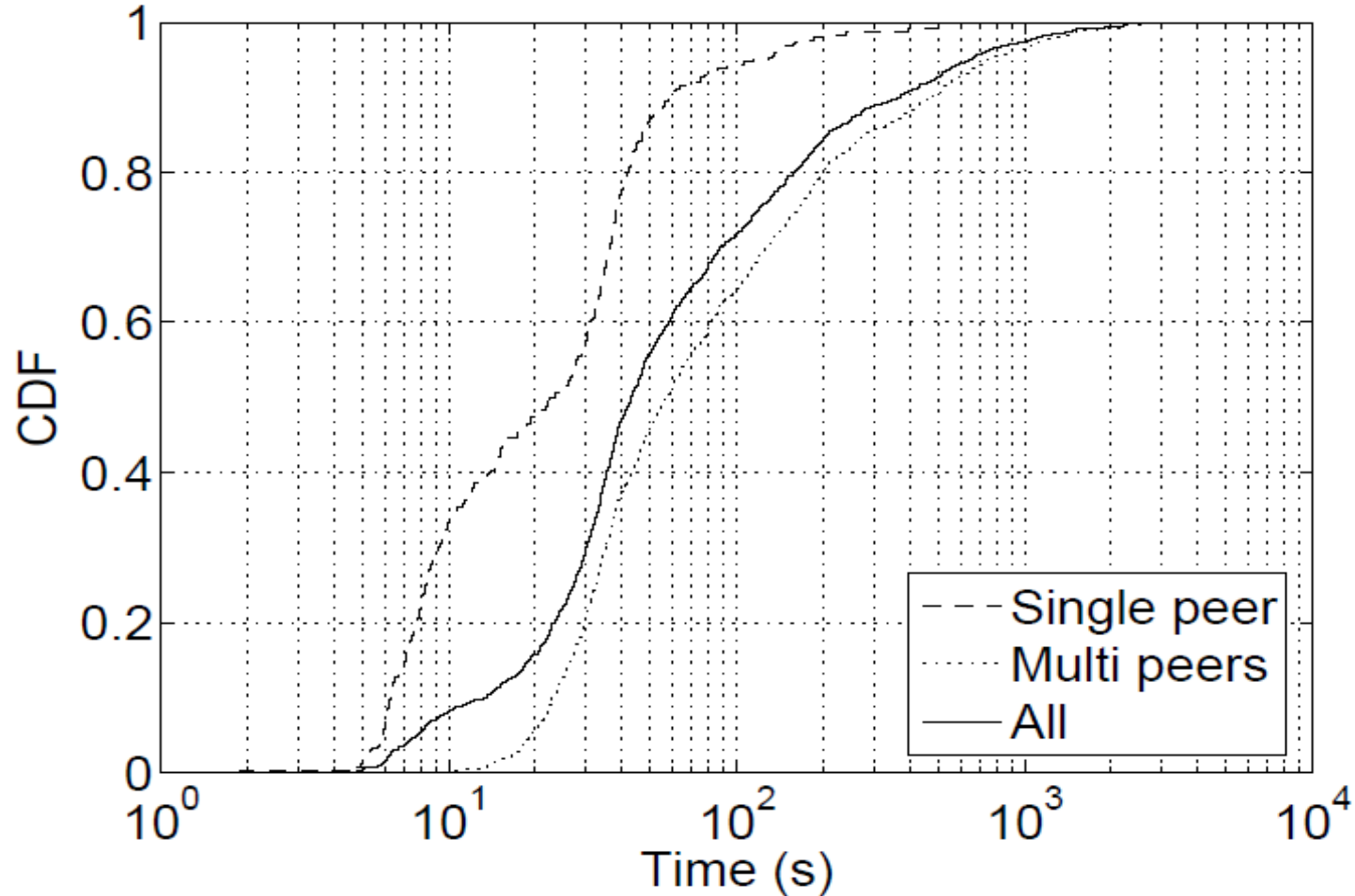
# Block Splitting

- ❑ What is the impact of block splitting on piece replication?



# Piece Intra Arrival Time [18]

Interarrival Time: First/Last Block for each Piece

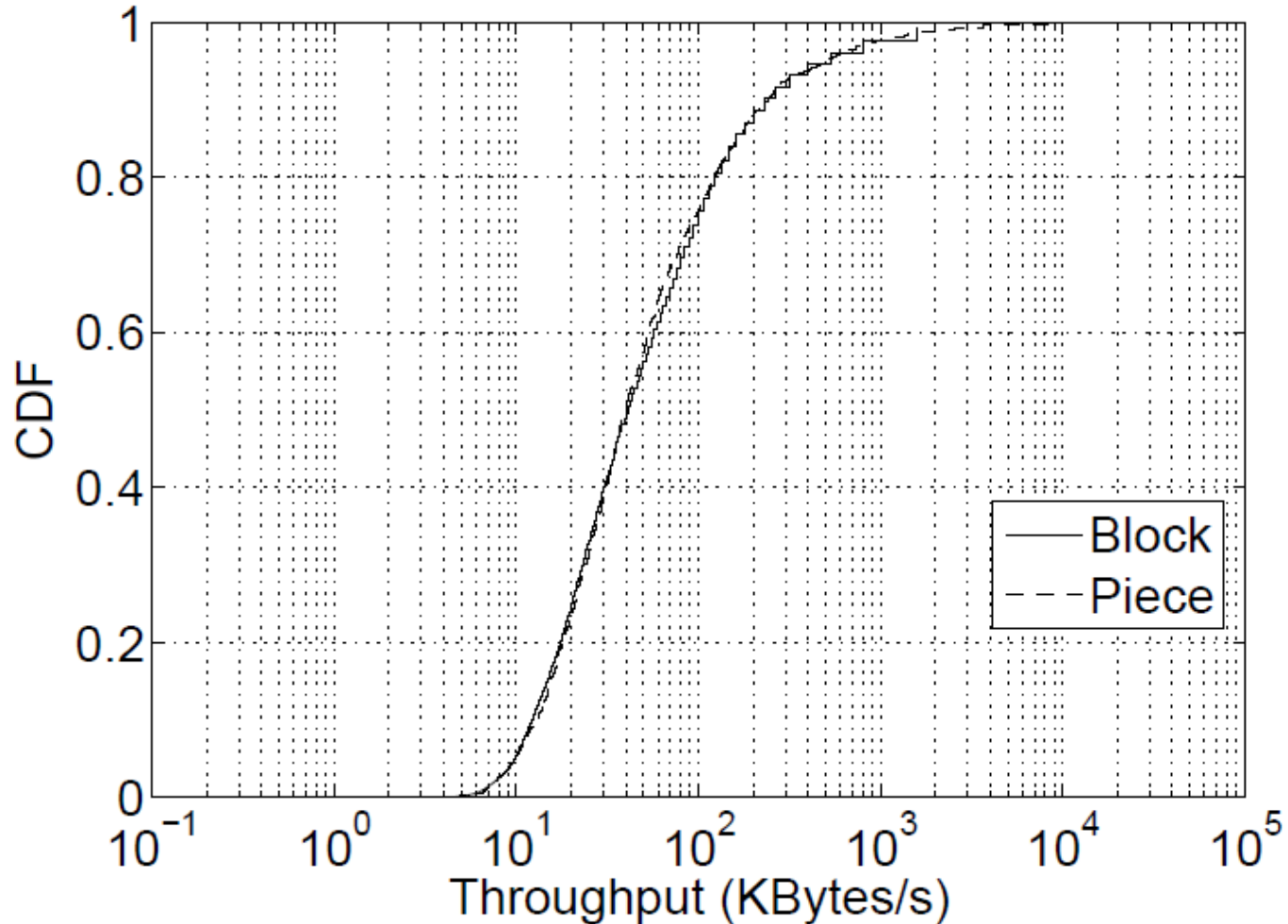


# Piece Intra Arrival Time [18]

- ❑ Some pieces have a large piece intra arrival time
  - Pieces downloaded from multiple peers
  - Strict priority mitigates the problem
- ❑ Only complete pieces can be retransmitted
- ❑ What is the impact of the piece intra arrival time on the piece download speed?

# Impact of Block Splitting [18]

Pieces and Blocks Download Throughput



# Impact of Block Splitting: Conclusion [18]

- ❑ Block splitting does not have an impact of piece download throughput
  - Strict priority mitigates successfully the impact of multi peer piece download

# Choke Algorithm

- Which fairness is achieved by the choke algorithm?

# Tit-for-Tat Fairness

- Choke algorithm fairness challenged in several studies
  - Does not guarantee strict byte reciprocation
  - Based on a short term throughput estimation

## □ Tit-for-tat Fairness

- Peer A can download data from peer B if:

(bytes downloaded from B - bytes uploaded to B) < threshold

# Tit-for-Tat Fairness

## □ Tit-for-tat fairness problems

- Does not take into account extra capacity
  - Seeds cannot evaluate the reciprocation of leechers
  - Leechers may have asymmetric capacity
- May lead to deadlock, as it is complex to find appropriate thresholds

## □ Need for another notion of fairness

# Peer-to-Peer Fairness

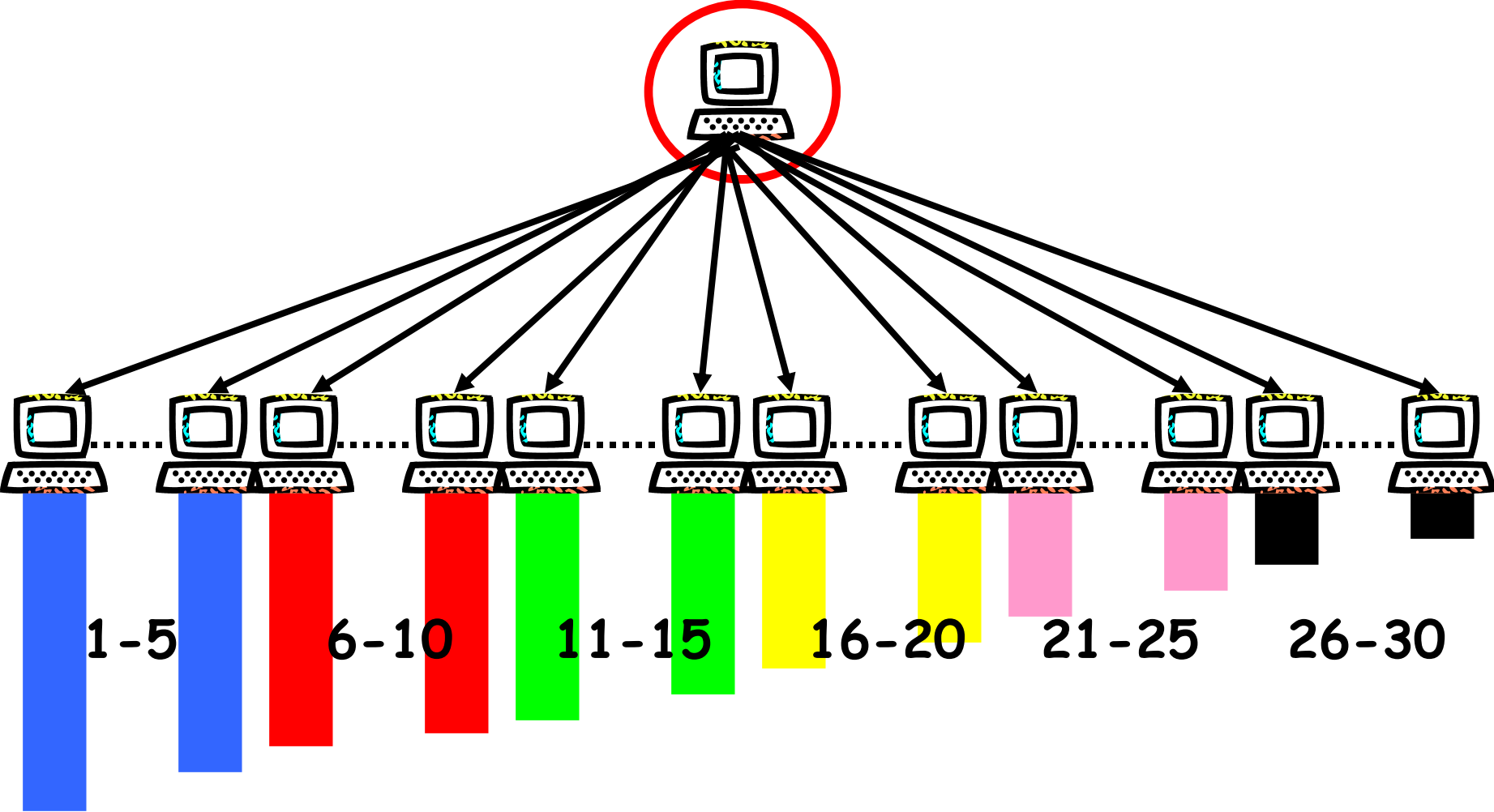
- Two criteria (inspired from BitTorrent)
  - Any **leecher**  $i$  with upload speed  $U_i$  should get a lower download speed than any other leecher  $j$  with an upload speed  $U_j > U_i$ 
    - A leecher must not receive a higher service than any other leecher that contributes more than himself
    - Do not steal capacity if it is used by someone else
    - No strict reciprocation
  - A **seed** must give the same service time to each leecher
    - Distribute evenly spare capacity



# Peer-to-Peer Fairness

- ❑ Excess capacity is used
- ❑ No need to maintain thresholds or enforce strict reciprocation
- ❑ Foster reciprocation and penalize free riders
  - Free riders cannot receive a higher capacity of service than contributing peers
  - In case of seeds, the larger the number of contributing leechers, the lower the amount of data received by free riders

# Fairness of the Choke Algorithm LS

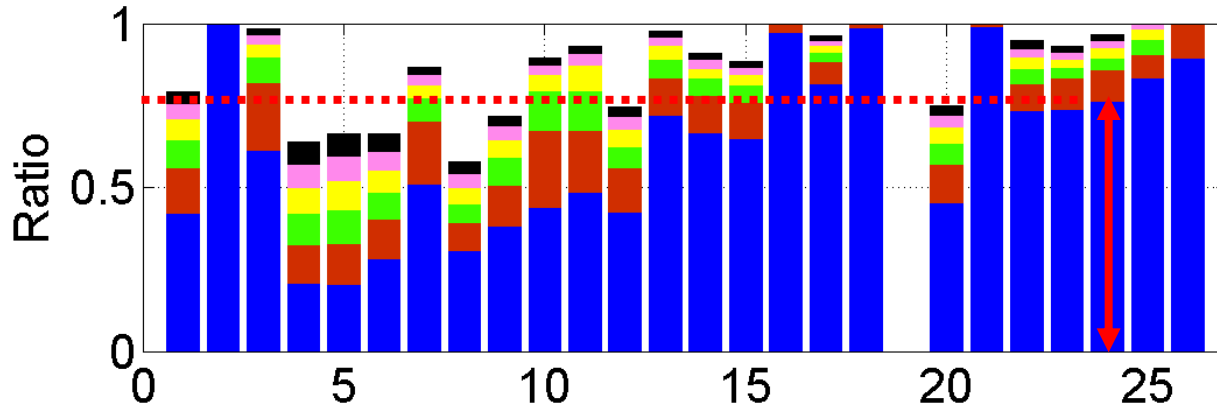


# Choke Algorithm: Leecher State

- ❑ We created 6 sets of 5 remote peers each
  - The blue (most left) corresponds to the five peers that receive the most
  - The black (most right) corresponds to the 26 to 30 peers that receive the most
- ❑ Set created based on upload speed (top subplot next slide)
- ❑ Same set kept for the bottom subplot of the next slide

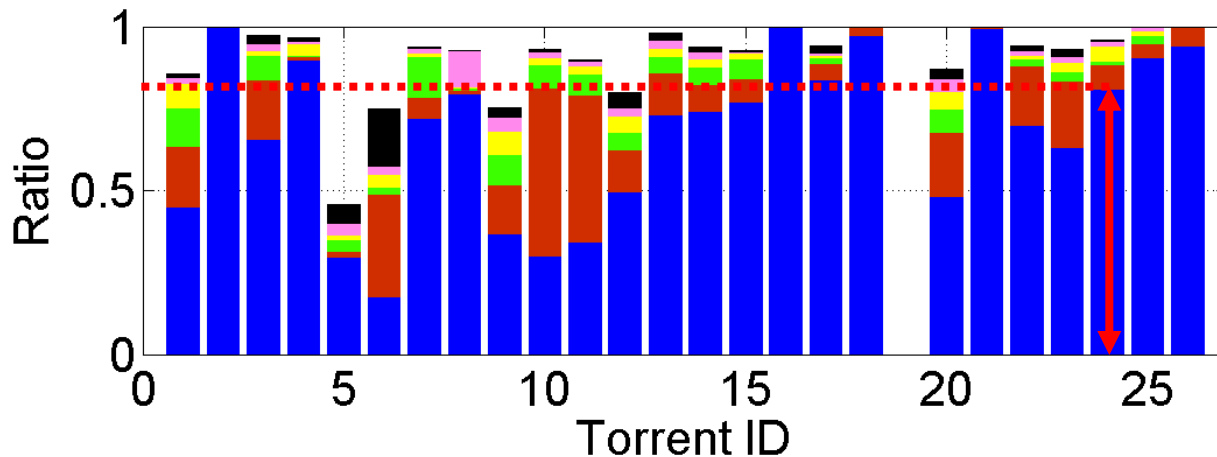
# Fairness of the Choke Algorithm LS

Contribution to the Amount of Uploaded Bytes, LS



□ Good reciprocation for torrents in steady state

Contribution to the Amount of Downloaded Bytes, L



□ Choke algorithm biased by poor peer availability for torrents in transient state



1-5

6-10

11-15

16-20

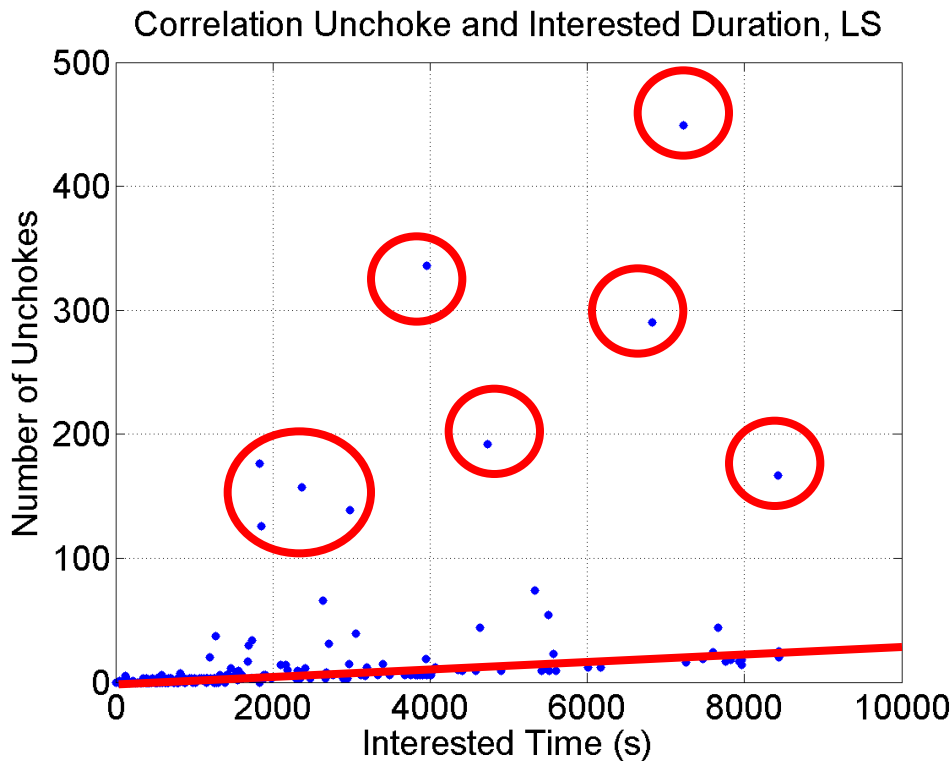
21-25

26-30

# Choke Algorithm: Leecher State

- ❑ Peers that received the most are also peers that gave the most
- ❑ For torrents in steady state, the five best uploaders receive most of the bytes (stable active peer set)
- ❑ Torrent 19
  - The local peer does not have any leecher in its peer set

# Choke Algorithm: Leecher State



- ❑ 1 seed, 713 leechers, torrent 7
- ❑ No correlation between interested time and unchoke duration
  - Peers unchoked based on their download rate

# Leecher State: Torrent 7

- ❑ Most peers OU, few peers RU a lot of time
- ❑ No correlation between RU and the remote peers interested time
- ❑ Correlation between OU and remote peers interested time
  - Because OU is random
- ❑ Choke algorithm in leecher state converge to an equilibrium

# Leecher State: Conclusion

## □ Choke algorithm

- P2P Fair
- Fosters reciprocation
- Select a small subset of peers to upload to
- Leads to an equilibrium

## □ Which kind of equilibrium, efficiency?

- We will answer soon

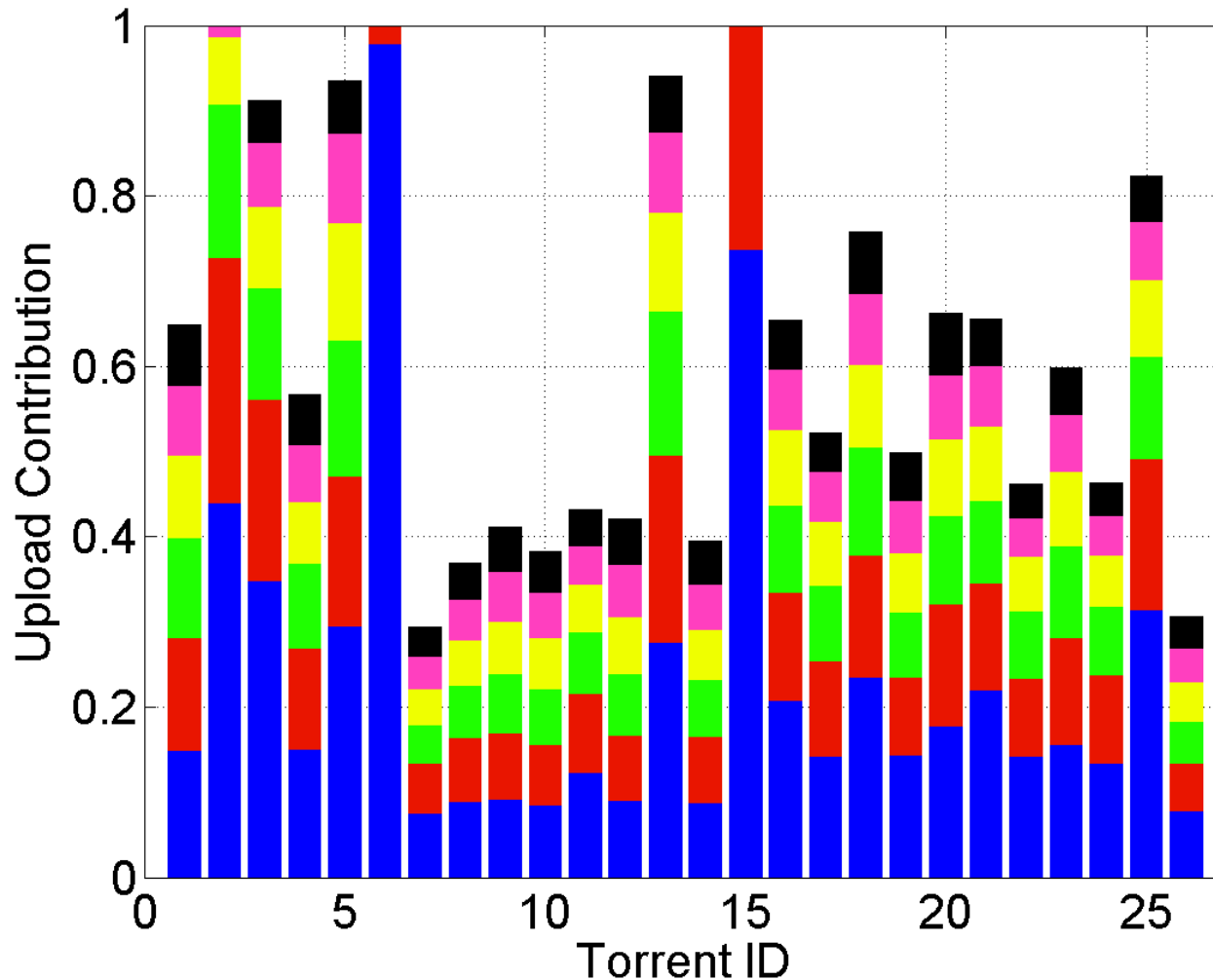


# Choke Algorithm: Seed State

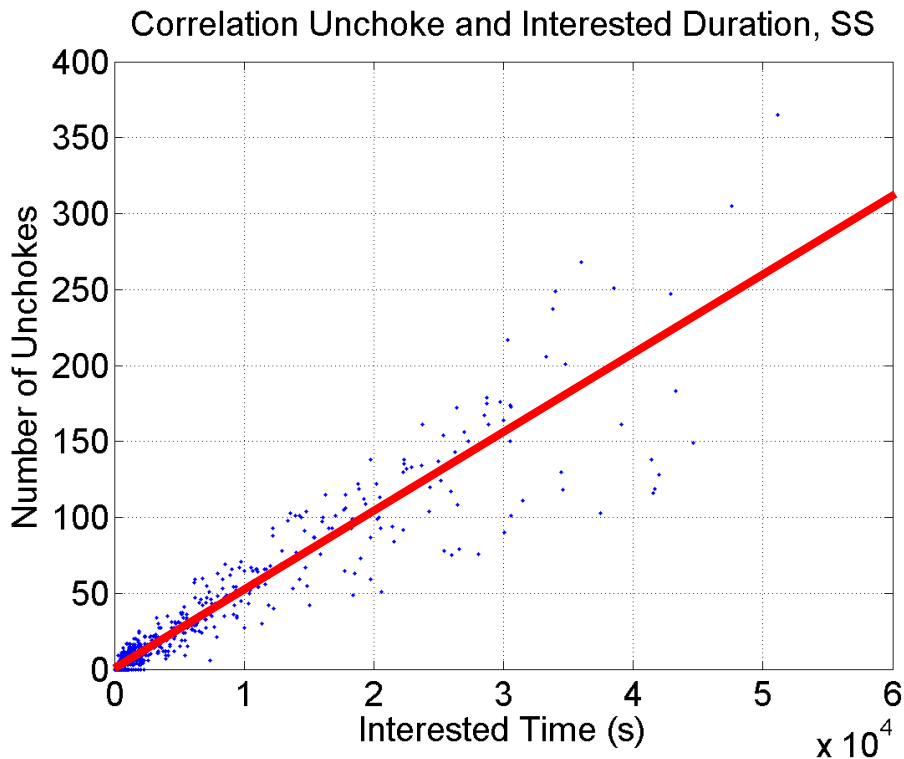
- We created 6 sets of 5 remote peers each
  - The blue corresponds to the five peers that receive the most
  - The black corresponds to the 26 to 30 peers that receive the most

# Choke Algorithm: Seed State

Contribution of Peers to the Amount of Uploaded Bytes, SS



# Choke Algorithm: Seed State



- ❑ 1 seed, 713 leechers, torrent 7
- ❑ Correlation between interested time and unchoke duration
  - Peers more likely to be unchoked when they are interested longer in the seed

# Seed State: Torrent 7

- ❑ Strong correlation between unchokes (SKU+SRU) and the remote peers interested time
- ❑ Choke algorithm in seed state shares evenly the download capacity of the seed among leechers

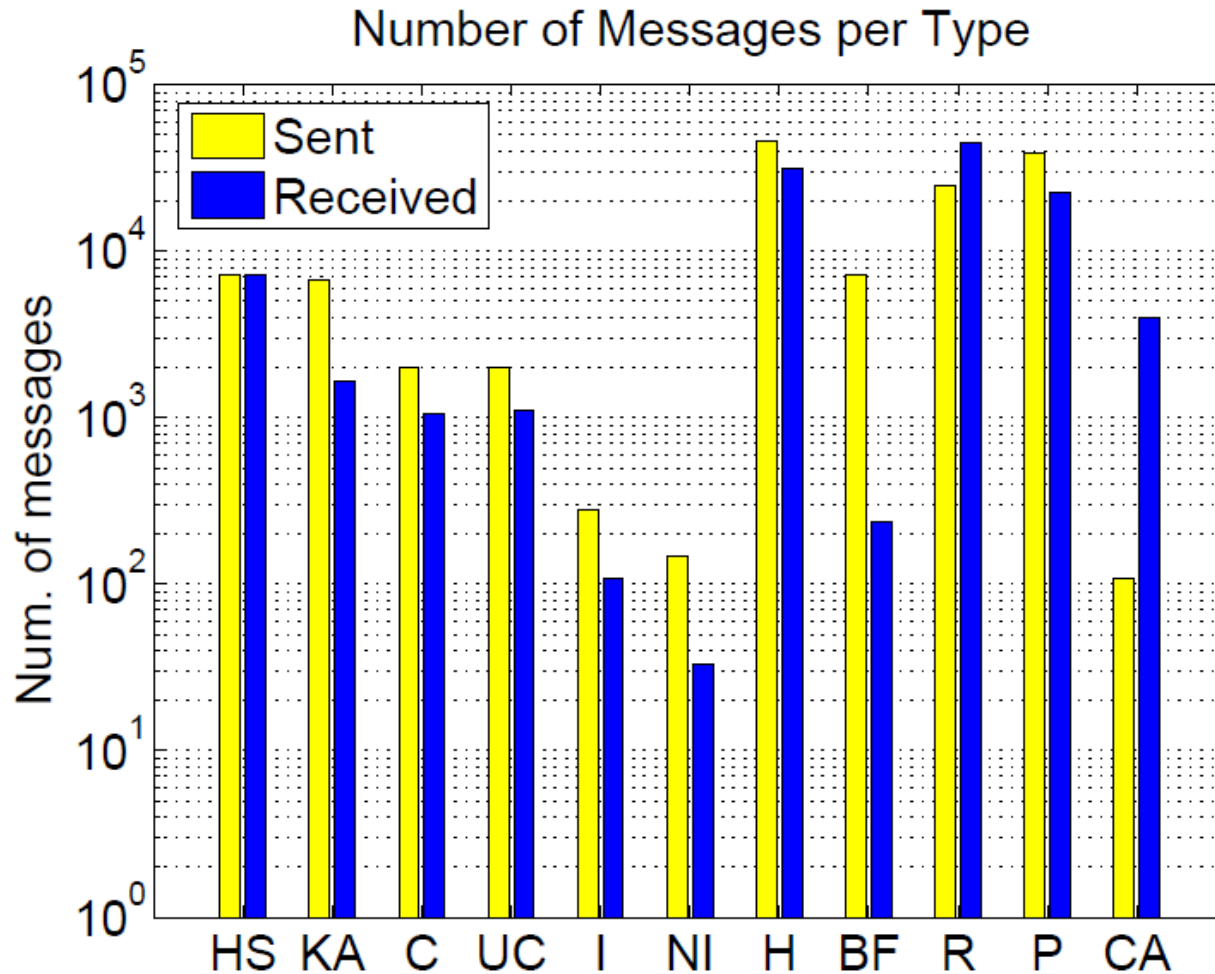
# Seed State: Conclusions

- ❑ Same service time to all the leechers
  - Depends on the time spent in the peer set
- ❑ Benefits
  - Entropy improved
    - Everybody receives the same amount of pieces
  - Free riders cannot receive more than contributing leechers
  - Better resilience of the torrent in the startup phase

# Protocol Overhead

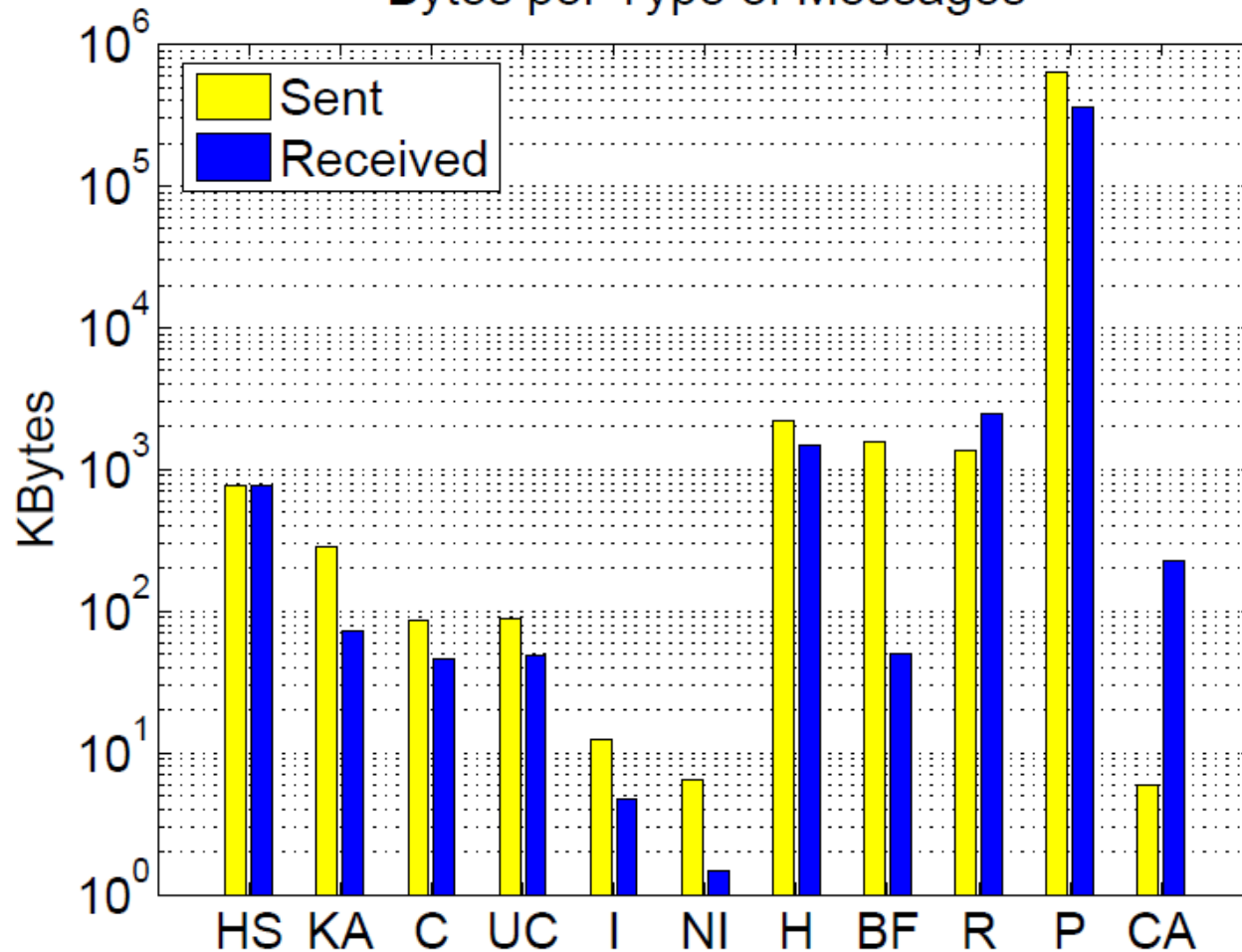
- ❑ What is the overhead of the BitTorrent protocol?

# Protocol Overhead: Messages [18]



# Protocol Overhead: Bytes [18]

Bytes per Type of Messages



□ Count the TCP/IP header (40 bytes)



# Protocol Overhead [18]

- ❑ We count as overhead the 40 bytes of the TCP/IP header for each message exchanged plus the BitTorrent message overhead
- ❑ We count as payload the bytes received or sent in a PIECE message without the PIECE message overhead
- ❑ Upload overhead
  - Ratio of all the sent messages overhead over the total amount of bytes sent (overhead + payload)
- ❑ Download overhead
  - Ratio of all the received messages overhead over the total amount of bytes received (overhead + payload)

# Protocol Overhead: Conclusions

## [18]

- 2% of overhead for most of the experiments
- Messages that account for most of the overhead
  - HAVE, REQUEST, BITFIELD

# Protocol Overhead: Conclusions

## [18]

### □ Download overhead

- Peers that stay long in a torrent in seed state
  - Do not receive any payload data anymore
  - Continue to receive messages (HAVE, REQUEST, BITFIELD)
- Increases moderately
- Unavoidable if a peer acts as a seed, which does not receive anything by definition

# Protocol Overhead: Conclusions

## [18]

### □ Upload overhead

- Peers that stay few time in a torrent in seed state or that have a small upload speed
  - Bytes uploaded in seed state reduce the upload overhead
  - But, most of the upload overhead is created in leecher state: all the HAVE and REQUEST messages sent in leecher state

# BT Algorithms Conclusions

- ❑ Rarest first guarantees a high peer availability
  - No need for more complex solution in the monitored torrents
  - Transient state is a seed provisioning issue
  - No last pieces problem
- ❑ Choke algorithm is fair and fosters reciprocation
  - Robust to free riders
- ❑ Rarest first and choke algorithms are enough on real torrents
  - Simple and efficient on real torrents

# BT Algorithms Conclusions

- ❑ Keep in mind the limitations of this study
  - Single client instrumentation
  - Limited torrent set
- ❑ Keep in mind the context
  - Torrents in the Internet: good connectivity
  - Medium to large contents

# Properties of the Choke Algorithm

- We have seen that the choke algorithm in leecher state leads to an equilibrium
  - Questions are what is the efficiency of this equilibrium and what is the reason of this equilibrium?
- We have just taken the point of view of a single peer
  - Do our results still hold if we take a global point of view?
- We now focus on the choke algorithm

# Study 2 [35]



# Methodology: Experiments

- Instrumentation of around 40 peers on PlanetLab
  - 1 single initial seed always connected for the duration of the experiment
  - 40 leechers join at the same time (flash crowd) and leave as soon as they have the content
  - All peers (seed + leechers) use the instrumented client of study 1
  - Content: 113MB, 453 pieces (256kB each)

# Methodology: Experiments

- Run three types of experiments
  - $U_{\max}$  is the maximum upload speed
  - Two-class
    - 20 leechers with  $U_{\max} = 20\text{kB/s}$
    - 20 leechers with  $U_{\max} = 200\text{kB/s}$
  - Three-class
    - 13 **slow** leechers with  $U_{\max} = 20\text{kB/s}$
    - 14 **medium** leechers with  $U_{\max} = 50\text{kB/s}$
    - 13 **fast** leechers with  $U_{\max} = 200\text{kB/s}$
  - Uniform
    - Uniform max upload distribution with a  $5\text{kB/s}$  step starting from  $20\text{kB/s}$

# Methodology: Experiments

- Seed upload limit
  - Three types of experiments
    - 200 kB/s, 100 kB/s, and 20 kB/s
- No download limitation for leechers

# Experiments Rational

- Does the choking algorithm
  - Converge to an equilibrium?
    - Speed and stability
  - Provide effective sharing incentives?
    - How much do I gain if I contribute
  - Reach optimal efficiency?
    - How far is it from a 100% upload utilization

**What is the impact of the initial seed upload capacity on those properties ?**

# Experiments Rational

## □ Clustering

- Peers with same upload speed should interact
- It may seem clear
- But
  - A unchokes B, because B has been uploading fast to A
  - B continues uploading fast to A only if A starts uploading fast to B
  - Relationship initiated via an OU unchoke, but OUs are performed at random (no guarantee that A and B will ever meet)
- To create and preserve clustering
  - OUs should initiate interactions between peers with similar upload speeds.
  - Such interactions should persist, despite potential disruptions (OU by others, network bandwidth fluctuations)

# Metrics

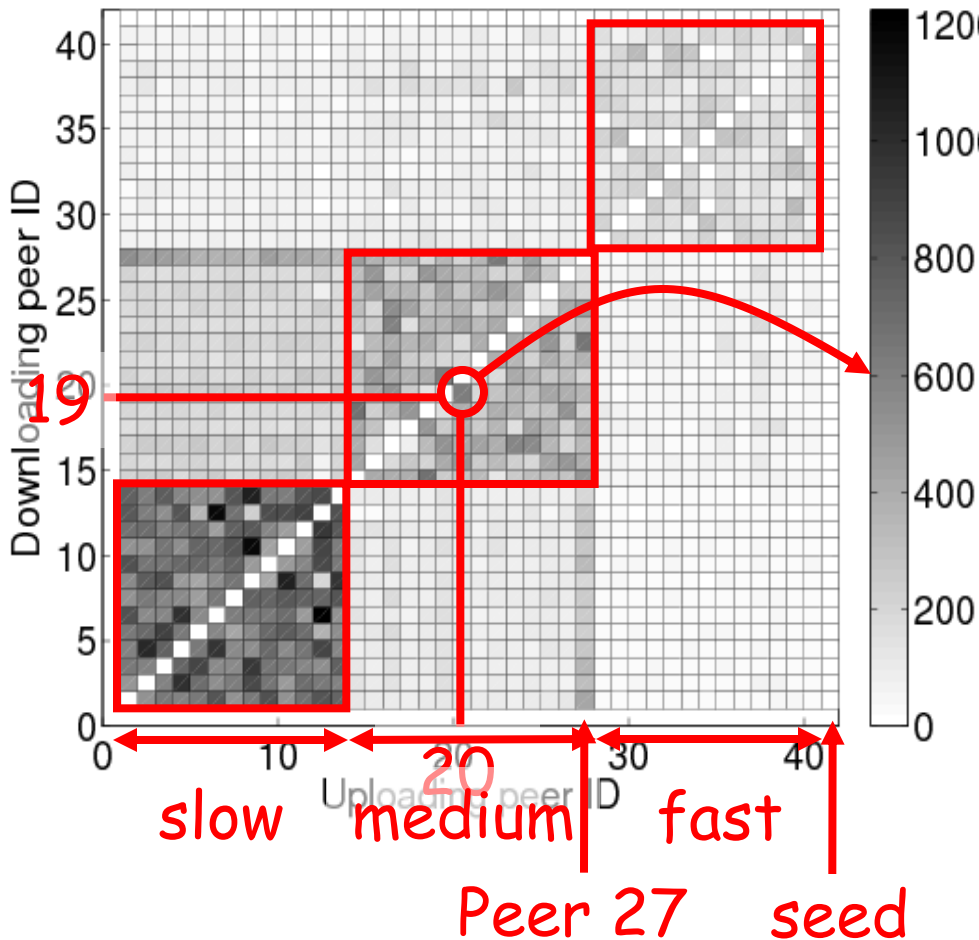
- ❑ Clustering index (cluster creation)
  - Convergence?
- ❑ Completion time (sharing incentives)
  - How does a peer's upload speed affects its download speed?
- ❑ Upload utilization (efficiency)
  - What is the peers' upload utilization?

# Fast Seed

- What are the choke algorithm properties with a fast seed?
  - Clustering
  - Sharing incentive
  - Upload utilization

# Peer Clustering: Fast Seed

Regular Unchoke Duration (All Runs)



- Three-class scenario, averaged over all 13 runs
- Seed max upload speed: 200kB/s
- We see clusters per class
- Two artifacts
  - Slow class squares are darker since peers take longer to complete
  - Peer 27 slower than other peers in its class (problem with a PlanetLab node): Reciprocates mainly with the slow leechers



# Clustering Index

□ Clustering index of a peer  $P$  for class  $C$

$$I_C(P) = \frac{\sum_{i=\text{peers in } C} \text{duration of regular unchokes of } P \text{ to } i}{\sum_{i=\text{all peers}} \text{duration of regular unchokes of } P \text{ to } i}$$

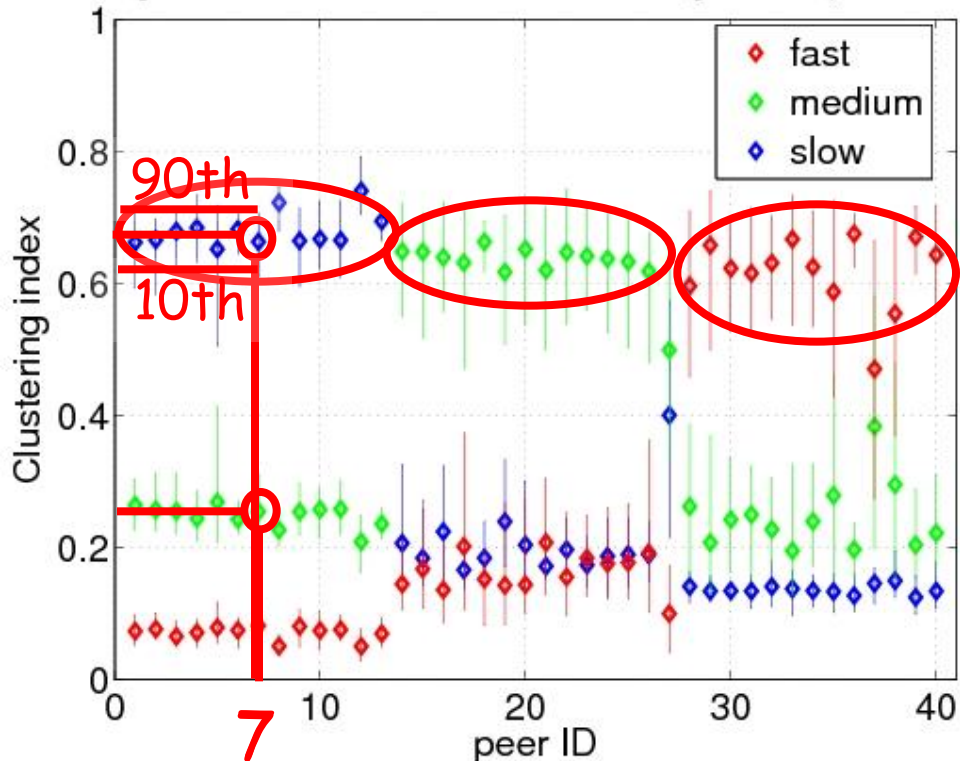
□  $I_C(P)=1$  if  $P$  unchoked only peers in class  $C$

□  $I_C(P)=0$  if  $P$  unchoked only peers not in class  $C$

□  $I_C(P)=0.3$  if  $P$  unchoked peers uniformly at random (with 3 classes)

# Peer Clustering: Fast Seed

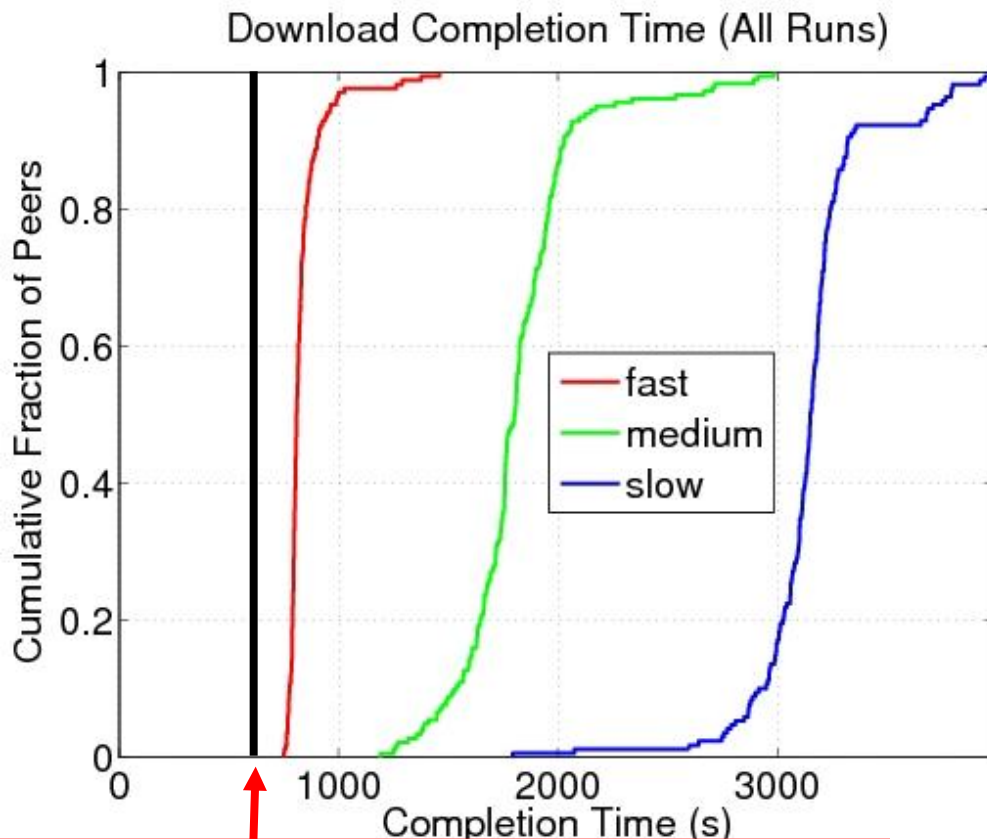
Regular Unchoke Duration Clustering Index (All Runs)



- ❑ Three-class scenario, averaged over all 13 runs
- ❑ Seed max upload speed: 200kB/s
- ❑ Each peer has a high clustering index to peers in its class
  - ❑ Peers of a specific class prefer to unchoke peers in the same class

Clusters of peers in the same class

# Sharing Incentive: Fast Seed



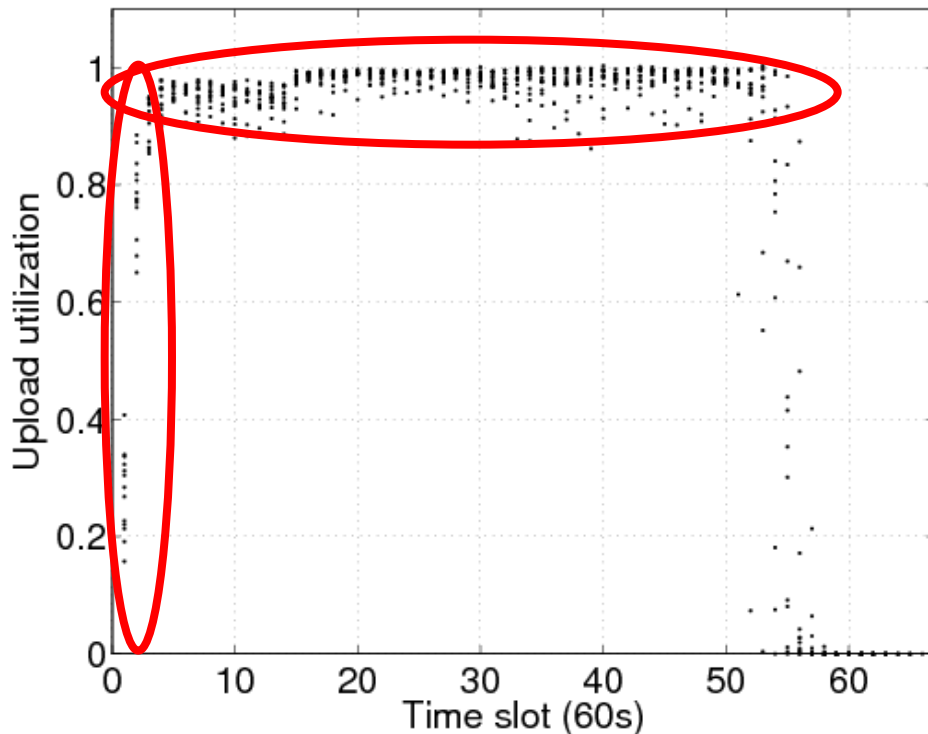
- ❑ Three-class scenario, for all 13 runs
- ❑ Seed max upload speed: 200kB/s
- ❑ Fast peers complete close to earliest possible completion time
- ❑ The more you contribute the faster you complete

Earliest possible completion

Effective sharing incentive

# Upload Utilization: Fast Seed

Global Upload Utilization (All Runs)



- ❑ Three-class scenario, for all 13 runs
- ❑ Seed max upload speed: 200kB/s
- ❑ Each dot is the average upload utilization over all peers for a single run
- ❑ **Upload utilization close to 1**
  - Room for improvement at the beginning

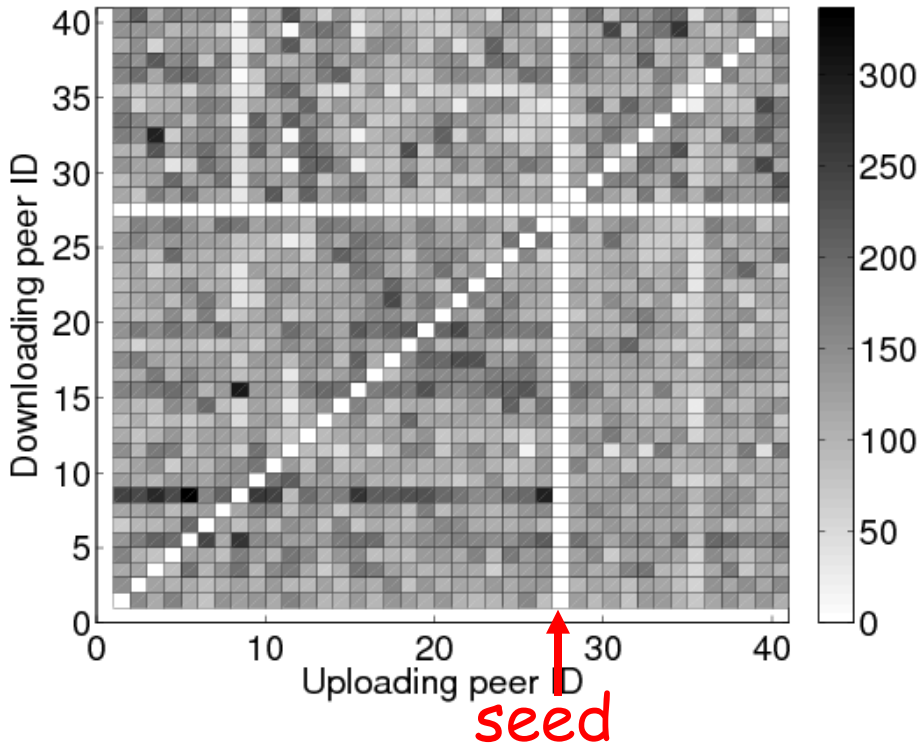
**High upload utilization**

# Slow Seed

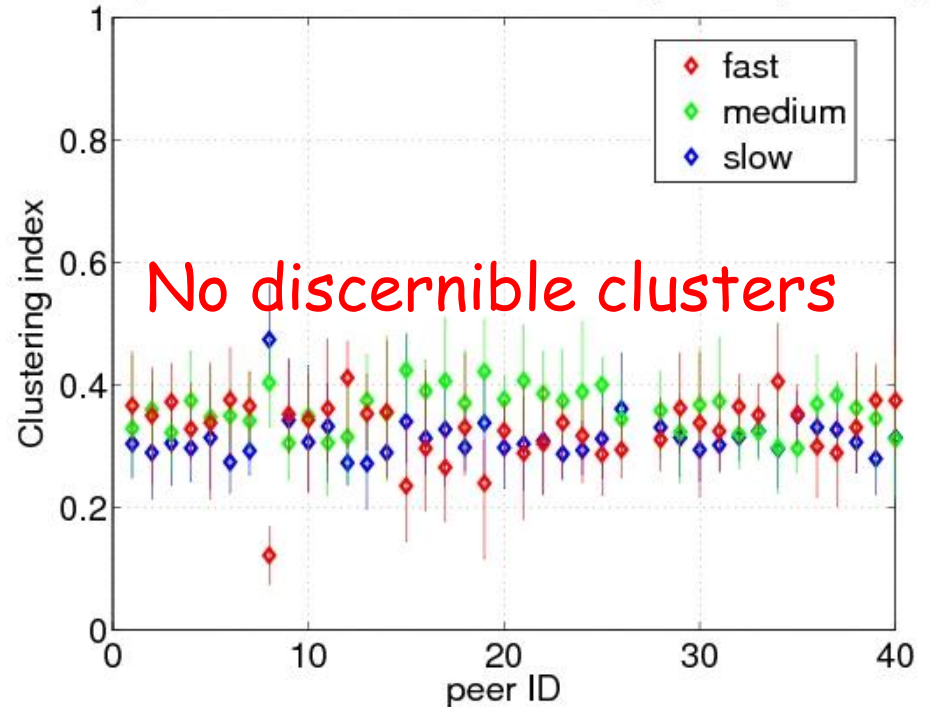
- What are the choke algorithm properties with a slow seed?
  - Clustering
  - Sharing incentive
  - Upload utilization

# Peer Clustering: Slow Seed

Regular Unchoke Duration (All Runs)



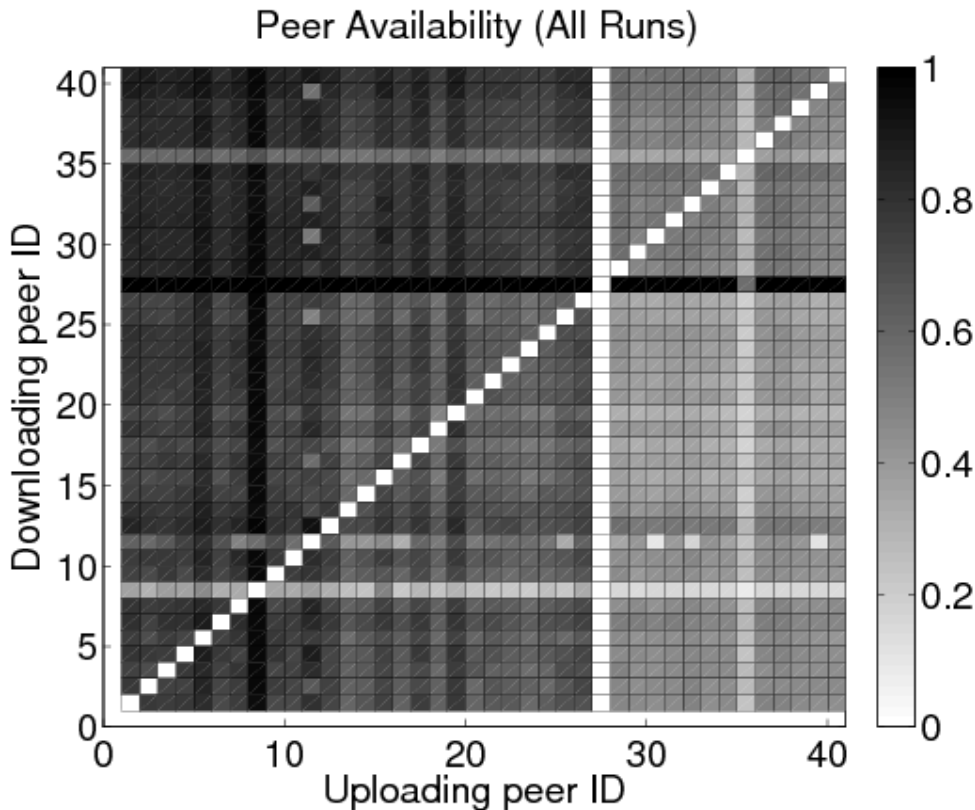
Regular Unchoke Duration Clustering Index (All Runs)



- ❑ Three-class scenario, averaged over all 8 runs
- ❑ Seed max upload speed: 100kB/s

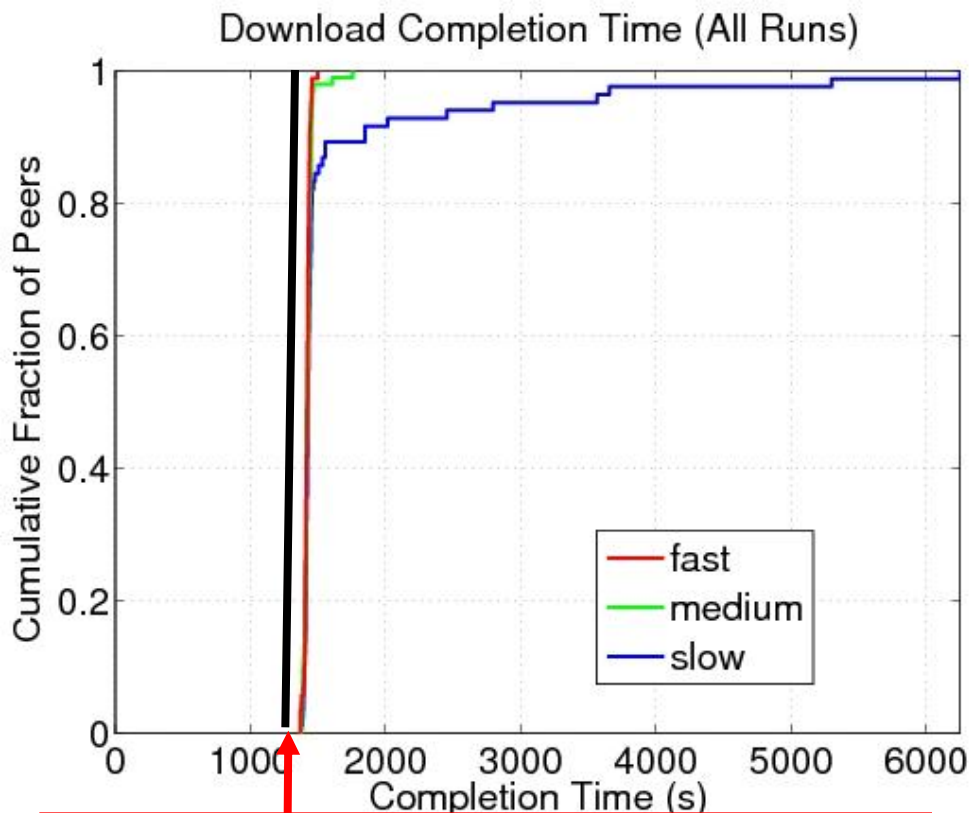
**Fast peers break all clusters**

# Peer Clustering: Slow Seed



- Fast peers have poor peer availability to other peers
  - Fast peers reciprocate with slower peers (when fast peers are not interested in any other fast peer), thus they break clusters for slower peers

# Sharing Incentive: Slow Seed



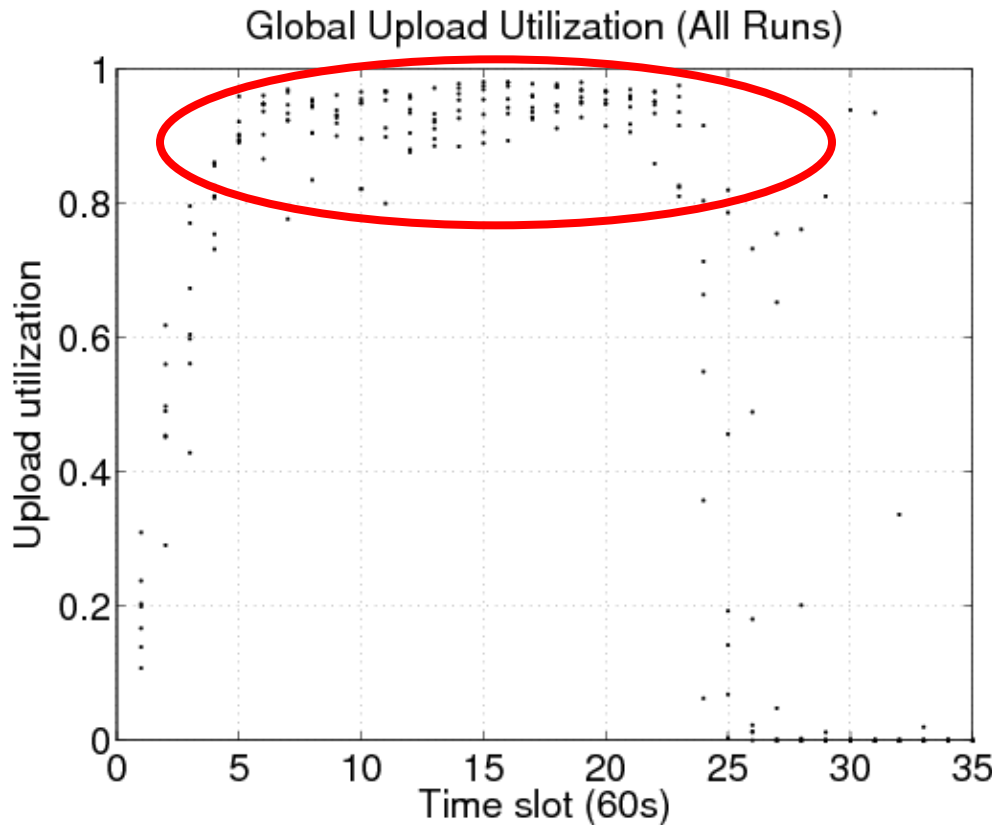
Earliest possible completion time

- ❑ Three-class scenario, for all 8 runs
- ❑ Seed max upload speed: 100kB/s
- ❑ Most peers complete close to the earliest completion time
  - Choking algorithm does not provide effective sharing incentive when the seed is underprovisioned
  - Earliest completion time longer than with a fast seed

No effective sharing incentives



# Upload Utilization: Slow Seed



- ❑ Three-class scenario, for all 8 runs
- ❑ Seed max upload speed: 100kB/s
- ❑ Each dot is the average upload utilization over all peers for a single run
- ❑ **Still fairly high upload utilization**
- ❑ With an initial seed at 20kB/s the upload utilization falls to 0.2

Upload utilization depends on seed upload speed

# Summary

- ❑ Seed provisioning is critical to the choking algorithm's effectiveness
- ❑ Well-provisioned initial seed
  - Cluster formation, effective sharing incentive, good upload utilization
- ❑ Underprovisioned initial seed
  - No clustering, ineffective sharing incentives, upload utilization can still be high
  
- ❑ What is the practical impact of these results?

# Seed provisioning

- ❑ It has been known that the initial seed upload speed is critical to the service capacity of torrents in their starting phase
- ❑ We have shown that it is also critical to the robustness of the torrent to free riders

**The seed should be at least as fast as the fastest leechers to support a robust torrent during the startup phase**

# Seed provisioning

- The seed should be as fast as the fastest peers
  - Rule of thumb
  - How to know the fastest peers?
  - Depends on how many fast peers there are
- [40] provides a model that confirms that RU leads to the formation of clusters

# Outline

- Overview
- Content Replication
- BitTorrent
  - Overview
  - Algorithm details
  - Evaluation
    - Torrent Scale
    - Algorithms
- Advanced subjects
- Security
- Localization

# Defeating BitTorrent

# Defeating BitTorrent: Myth or Reality

- Several studies [36-39] claim that BitTorrent can be defeated, i.e., free riding is possible
- No strong result up to now
  - Those studies either consider the FU choke algorithm in seed state
    - The RR choke algorithm render those attacks impossible
  - Or they use tricks to improve the upload speed of free riders
    - A contributing peer will always receive a higher service rate than a free rider, thus the sharing incentive still exist

# Defeating BitTorrent: Myth or Reality

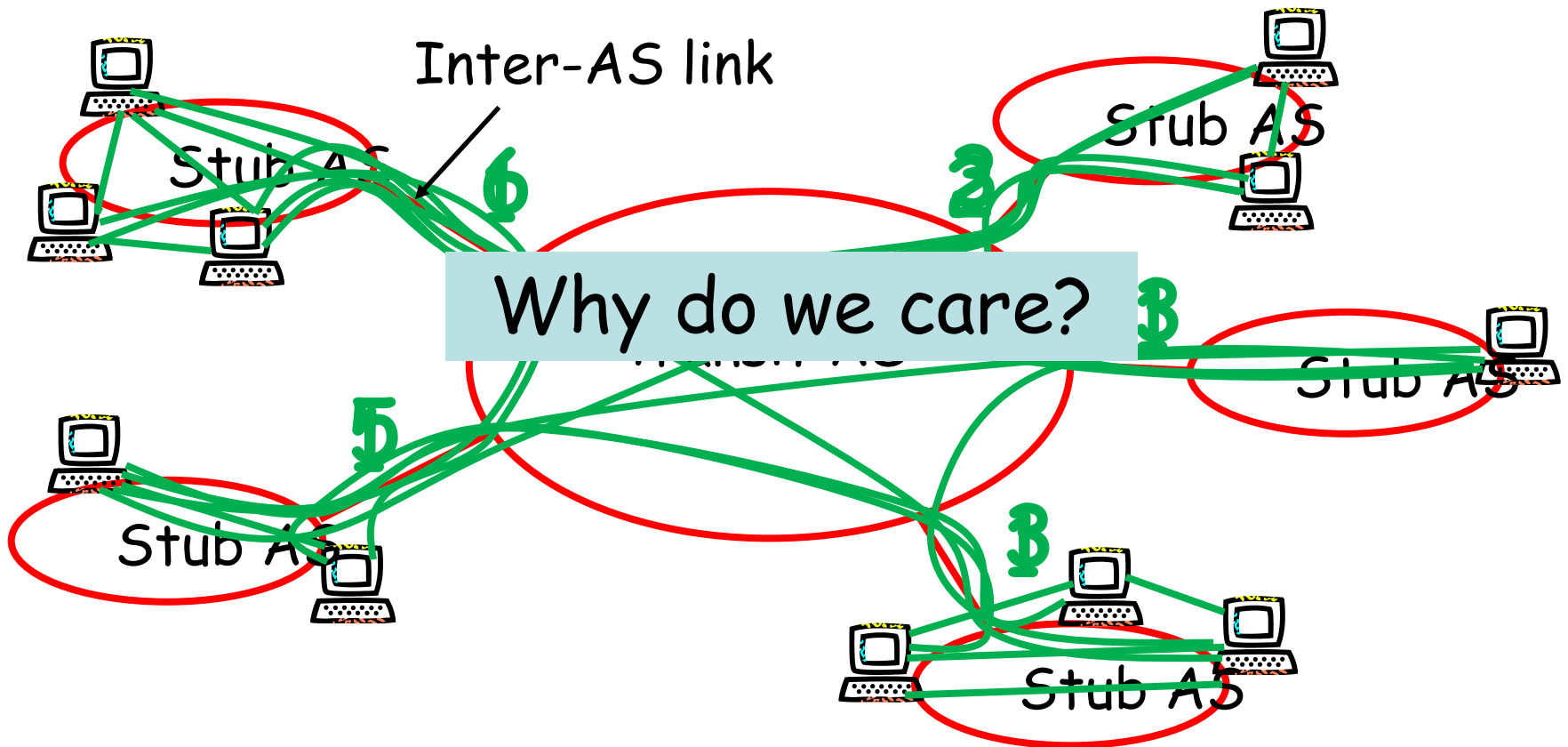
- ❑ To really defeat BT, one would need to find a way for a free rider to receive a higher service rate than a contributing peer
  - Nobody has found such an attack up to now



# BitTorrent Locality [43]

# What is BitTorrent Locality?

□ 11 TCP flows: random



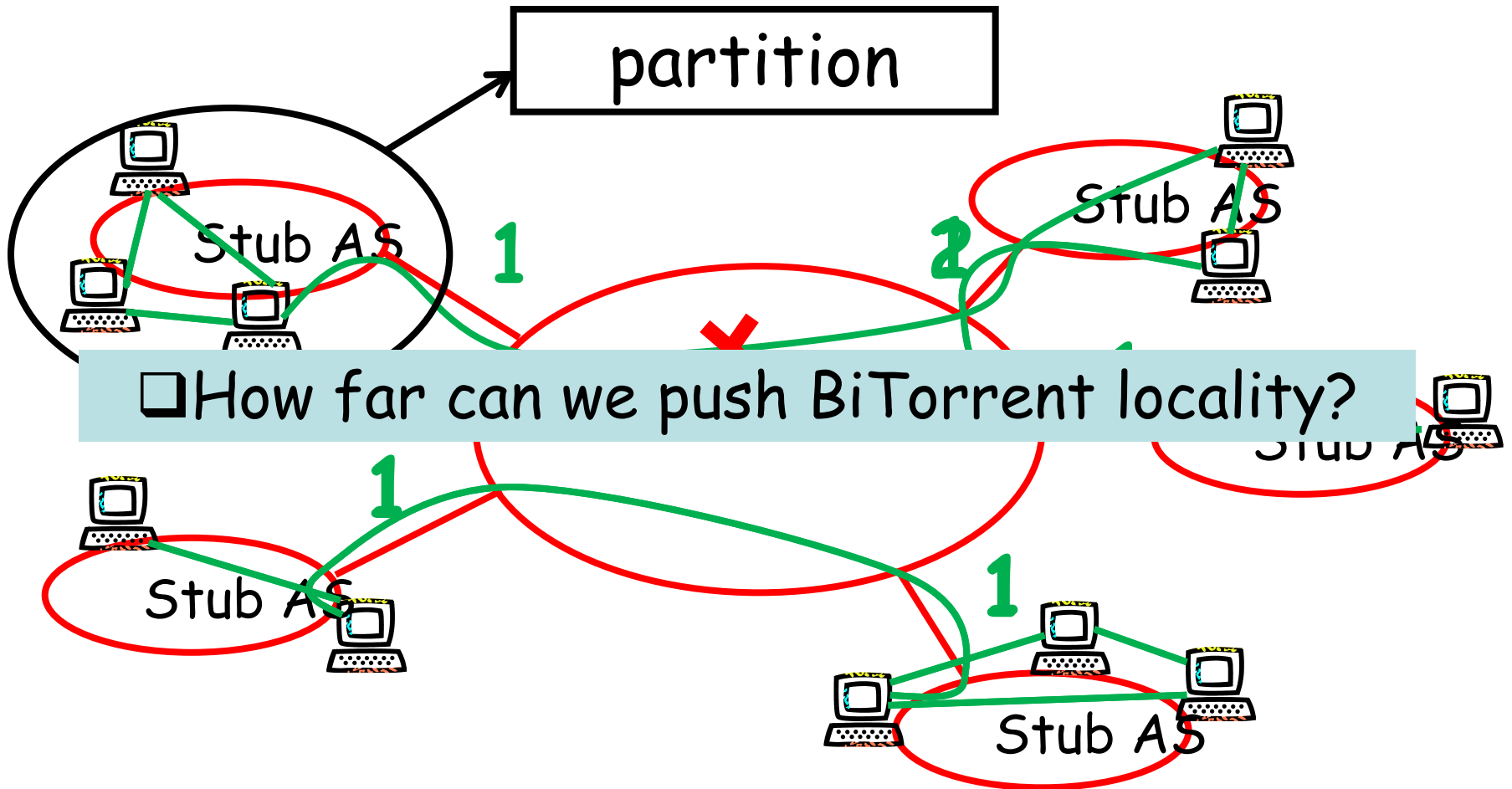
# Why Do We Care?

- ❑ Content provider save money by using P2P content distribution
  - No need for dedicated infrastructure
- ❑ ISPs suffer from P2P content distribution
  - Do not take into account ISPs topology and in particular peering links
  - Peering links are expensive and loaded ones
  - ISPs block P2P traffic

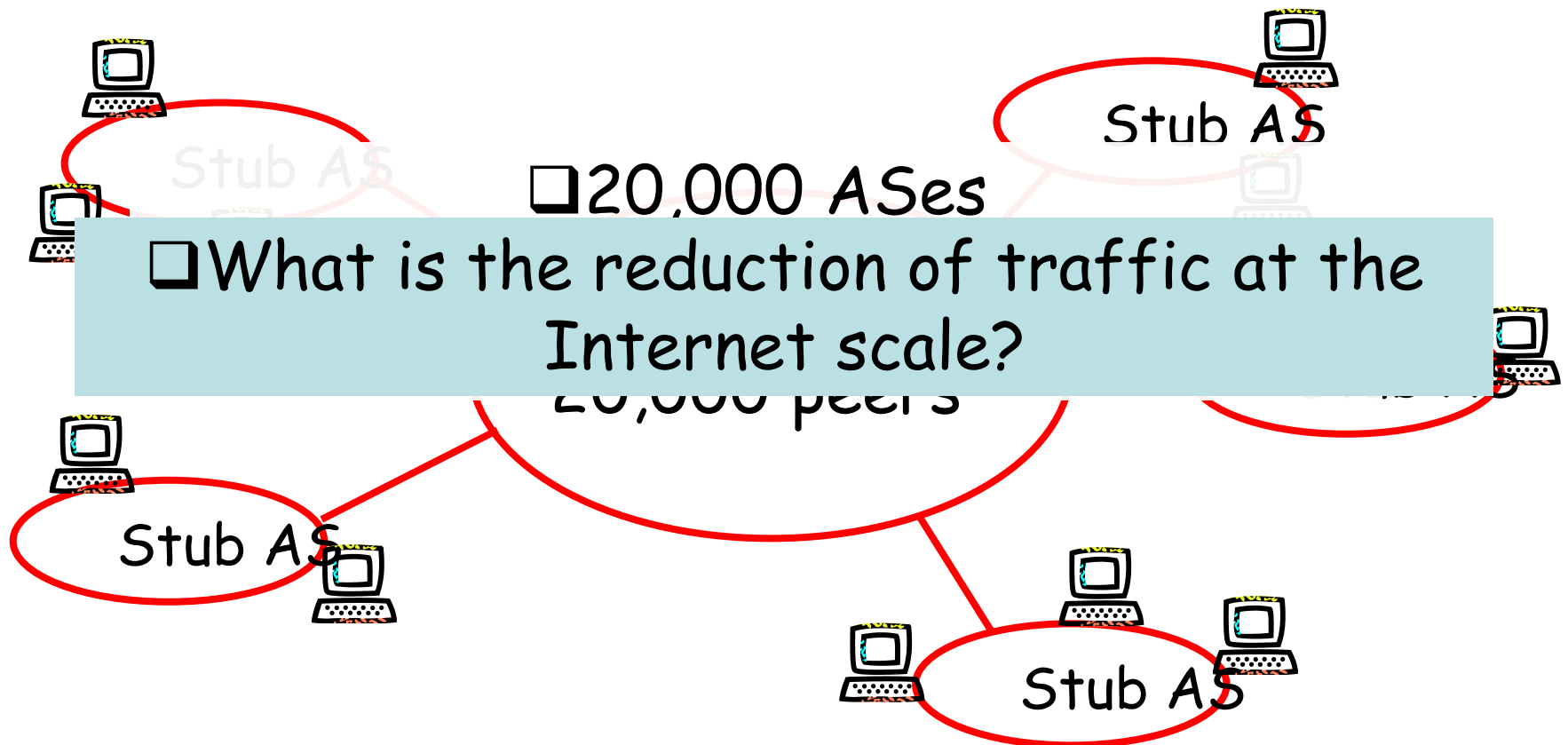
# Why Do We Care?

- Content providers need ISPs adoption of P2P
  - Do not create a higher load than classical client-server or CDN distribution
    - Do not overload peering links
  - No specific support can be assumed from the ISPs
    - Do not ask for specific servers
    - Do not ask for collaboration (disclosing structure of their network)

# What Are the Issues?



# What Are the Issues?



How far can we push  
BitTorrent locality?

# Methodology

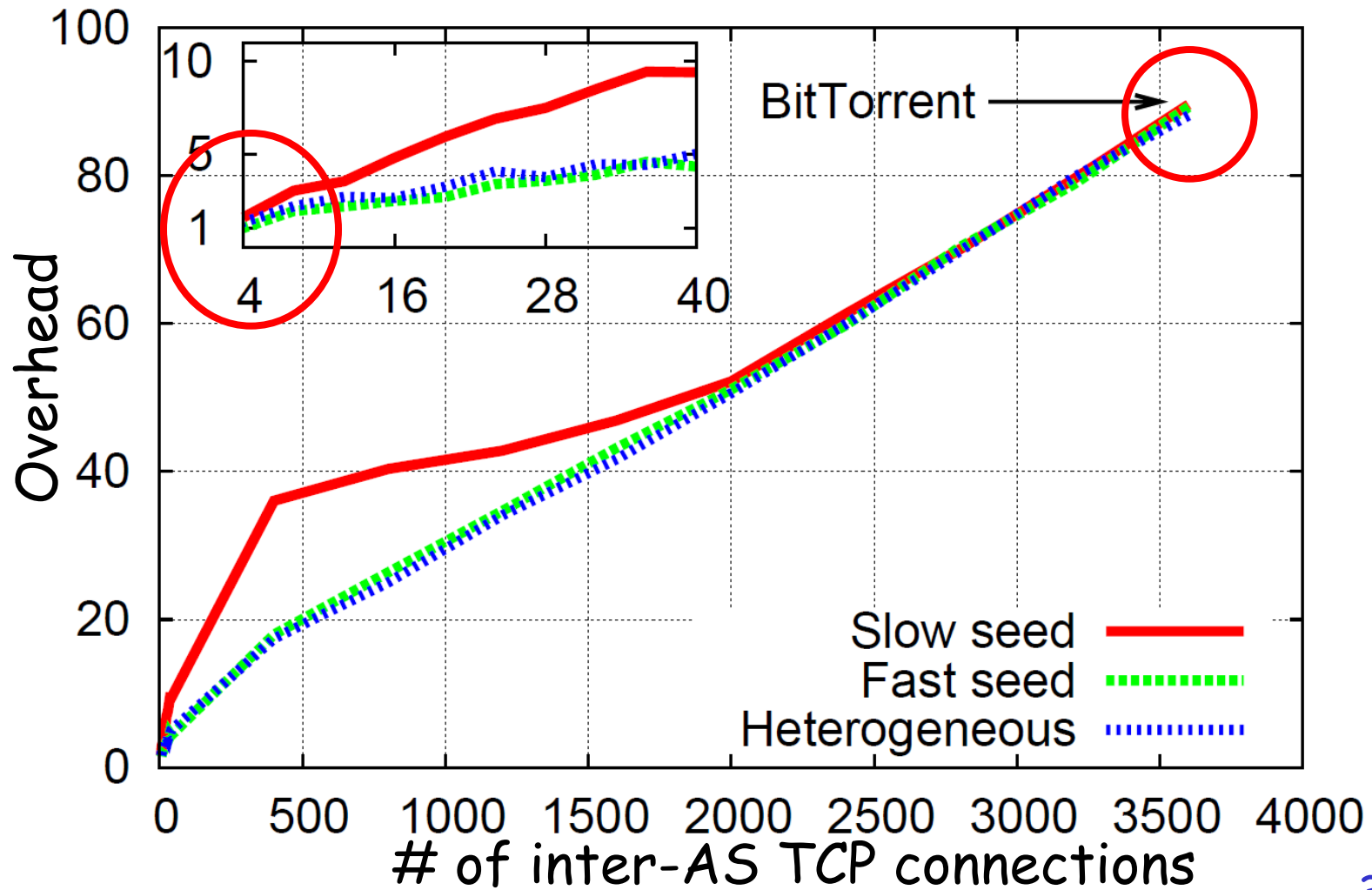
- ❑ Experiments on a cluster (grid5000)
  - 1000 peers, 10 ASes, 100MB content
- ❑ 3 scenarios
  - Slow seed: all peers upload at 20kB/s
  - Fast seed: all peers upload at 20kB/s, the seed at 100kB/s
  - Heterogeneous: 1/3 upload at 20kB/s, 1/3 at 50kB/s, 1/3 at 100kB/s, the seed at 100kB/s



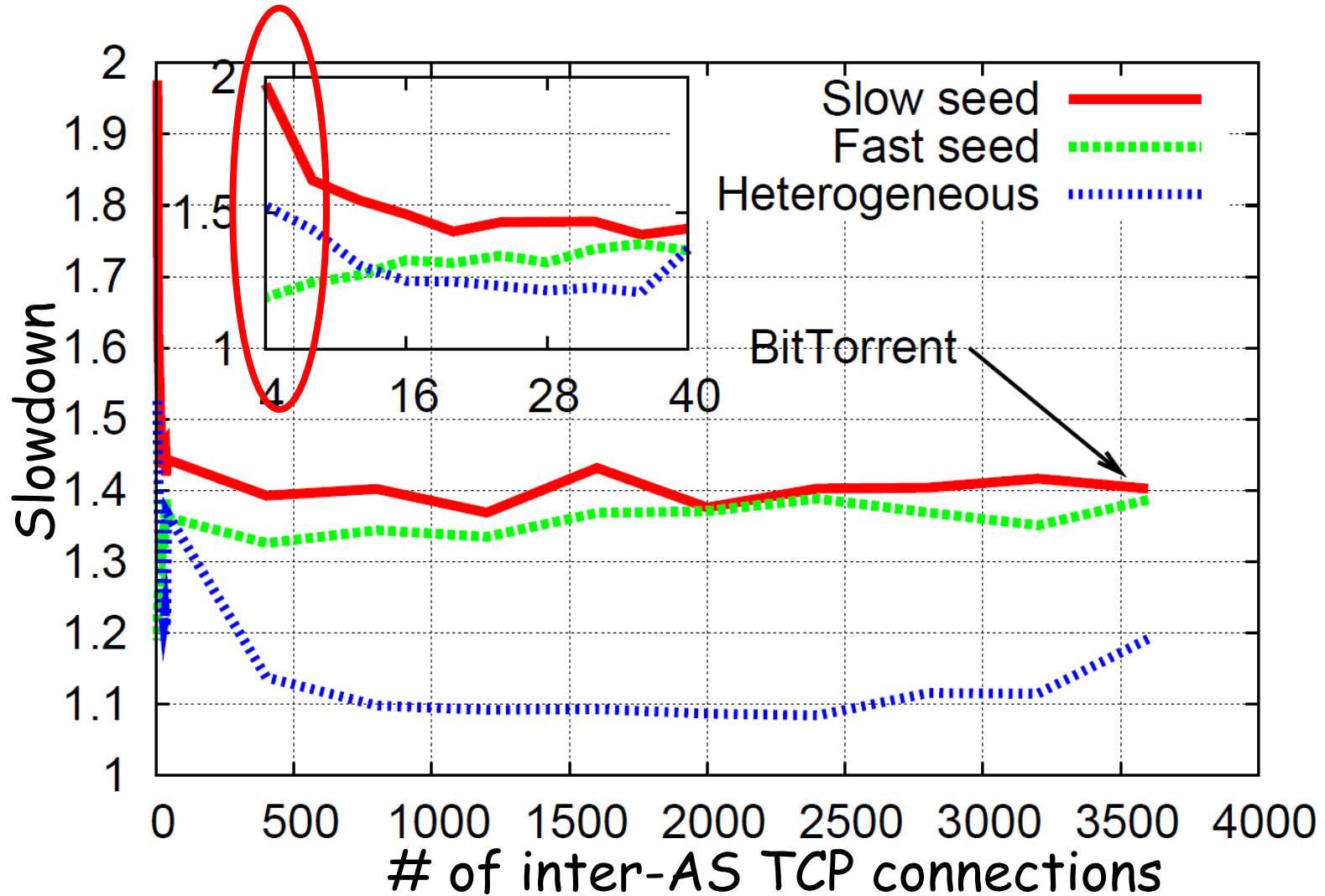
# Locality Policy

- For each AS, we limit the number of TCP connections between peers within the AS and peers outside the AS
  - Locality of  $X$  means there are  $X$  TCP connection between peers inside the AS and peers outside

# Overhead on Inter-AS Links



# Peers Slowdown



# Summary

- ❑ High overhead reduction is possible without performance losses
- ❑ Validated with
  - Various torrent sizes (from 100 to 10,000 peers)
  - Various AS sizes (from 10 to 5000 peers per AS)
  - Inter-AS bottlenecks
  - Churn

What is the traffic reduction  
at the Internet scale?

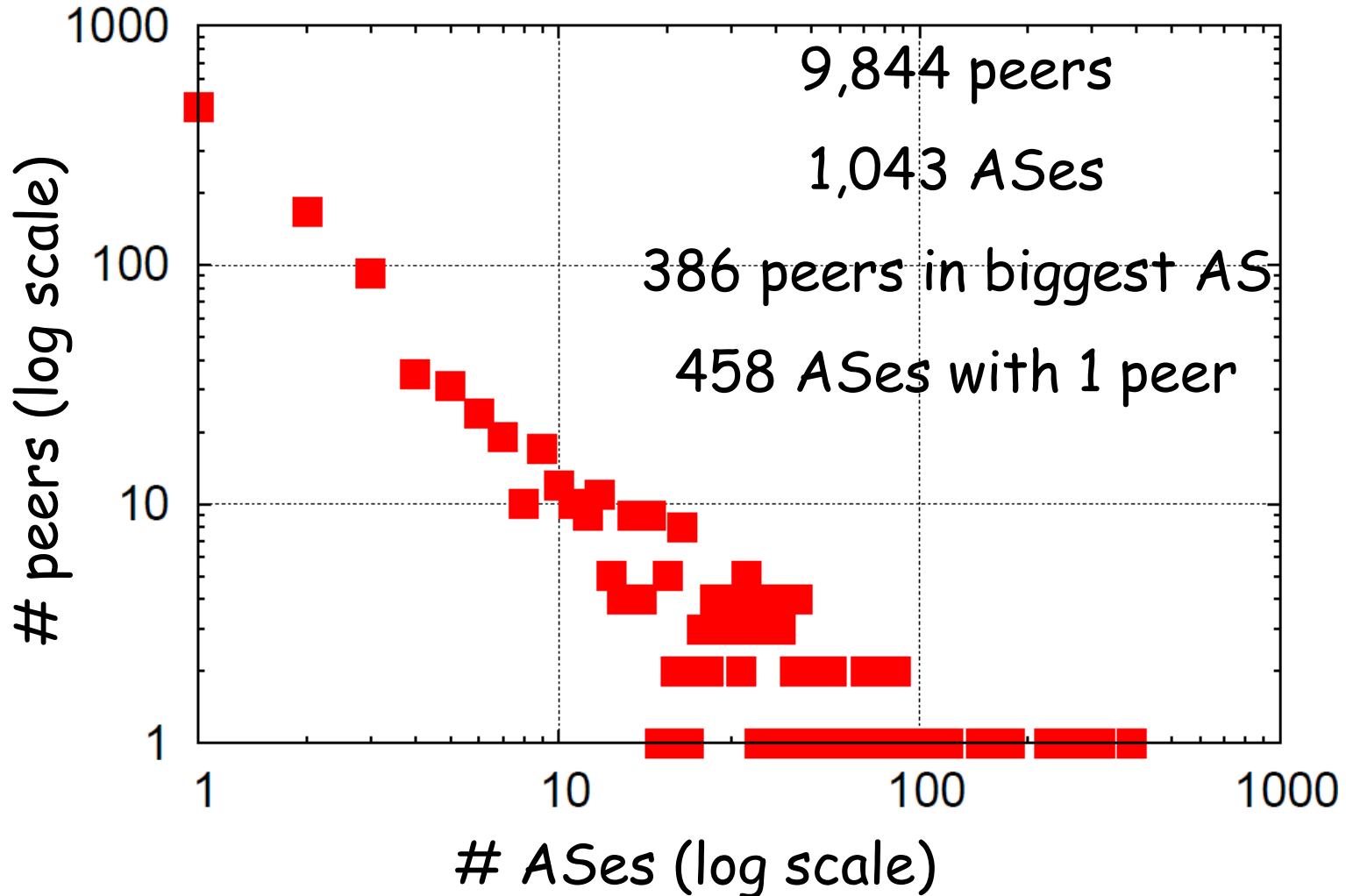
# Methodology

- On mininova we collected on 12 hours
  - 210k **active** torrents
  - 6M unique peers
  - Peers spread among 9.6k ASes
  - Largest torrents crawled within a few seconds
  - 110k torrents and 6.6k ASes cannot benefit from locality

# Methodology

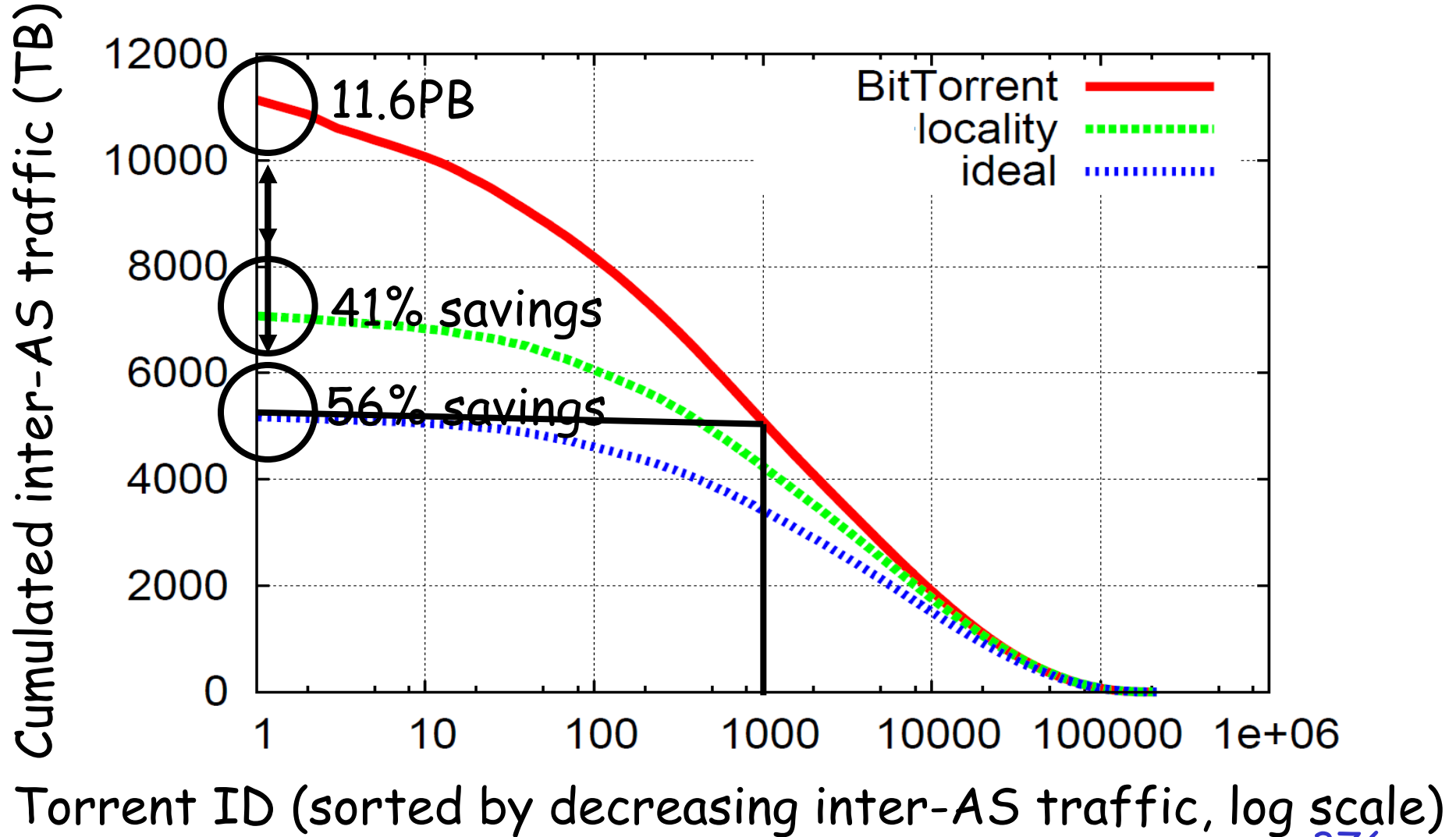
- Run real controlled experiments using real AS distribution for representative torrents
  - For each torrent, locality savings mainly depend on the number of peers per AS
- We used the average experimental savings to compute savings on all torrents

# AS Distribution of a Typical Torrent





# Benefits at the Internet Scale



# Conclusion

- How far can we push BitTorrent locality?
  - Up to few inter-AS TCP connections without performance penalty
- What is the traffic reduction at the Internet scale?
  - Up to 41% of all inter-AS traffic: 4.6PB

BitTorrent locality can achieve dramatic inter-AS savings at the Internet scale without performance penalty

# Important Studies

- [45] (T. Karagiannis et al.) Seminal work that introduces the notion of P2P locality and shows that it makes sense (you can reduce traffic)
- [46] (H. Xie et al. ) Proposition of an infrastructure called P4P to enable P2P locality

# Important Studies

- [47] (D. R. Choffnes et al.)  
Implementation of a Vuze plugin called Ono to implement locality with client support only
- [43] (Le Blond et al.) Experimental evaluation of the impact of locality of inter-AS traffic reduction and BitTorrent users download completion time

# Outline

- ❑ Overview
- ❑ Content Replication
- ❑ BitTorrent
- ❑ Security
  - Foundations
  - Privacy attacks
- ❑ Localization

# Foundations

- ❑ You MUST read [23] [24]
  - Small papers (2 and 4 pages only)
  - Best examples on how to use mathematics the simplest and best way
- ❑ Highly recommended read [51]
  - Easy to read
  - Book to understand the difference between theory and practice
- ❑ Current work
  - Tor, Freenet, Publius, OceanStore

# Shared Secret

# How to Share a Secret [23]

## □ Problem

- 11 scientists are working on a super secret project
  - They don't trust each other
  - The project is in a digital safe
- To open the digital safe, at least 6 out of the 11 scientists must be present

□ Apply to any problem with a group of suspicious individuals with conflicting interests that must cooperate



# (k,n) Threshold Scheme

- Formal definition of the previous example: (k,n) threshold scheme
- Let  $D$  be some secret data
  - Let's divide  $D$  into  $n$  pieces  $D_1, \dots, D_n$  such that
    - Knowledge on any  $k$  or more  $D_i$  pieces makes  $D$  easily computable
    - Knowledge of any  $k-1$  or fewer  $D_i$  pieces leaves  $D$  completely undetermined
- Very useful when  $D$  is a decryption key

# Trivial Solution

## □ Let's take a simple problem

- 11 scientists are working on a secret project
- The project is encrypted
- To decrypt the project at least 6 scientists have to be present

## □ Trivial solution

- Encrypt the content  $N$  times
- Each encryption key is split into 6 fragments

# Trivial Solution

- How many times the content must be encrypted?
  - Any set of 6 scientists is associated to a decryption key
    - Each scientist of a given set will have a fragment of a sixth of the key
  - The number of keys is the combination of 6 scientists out of 11
    - $\binom{11}{6} = \frac{11!}{6!(11-6)!} = 462$

# Trivial Solution

- How many fragment each scientist must carry
  - Any set of 6 scientists must be able to reconstruct a key, that is, to decrypt the content
  - Each scientist needs a different fragment to reconstruct each key with 5 other scientists chosen among 10 (that is 11 minus himself)
    - $\binom{10}{5} = \frac{10!}{5!(10-5)!} = 252$

# Trivial Solution

- Trivial solution is impractical
  - For only 11 scientists and 6 out of 11 able to decrypt the content
    - 462 keys, i.e., 462 encryptions of the content
    - 252 key fragments per scientist

# Shamir's (k,n) Threshold Scheme

## □ Basic idea

- A polynomial of degree  $k-1$  is uniquely defined by  $k$  points
  - 2 points for a line, 3 points for a parabola, 4 points for a cubic curve, etc.
- With  $k-1$  points only there is an infinity of  $k$  polynomial that can cross those points
  - So you need at least  $k$  points to find the polynomial equation  $g(x)$  using Lagrange interpolation
  - The secret is  $g(0)$

# Implementation on Galois Field

- All the arithmetic is modular arithmetic on Galois field (finite field)
  - Mandatory to provide perfect secrecy, that is  $k-1$  pieces do not give any information on the secret under a  $(k,n)$  threshold scheme
  - The set of integers modulo a prime number  $p$  forms a field in which interpolation is possible

# Implementation (k,n) Threshold Scheme

## □ Create the n fragments

- Let  $D$  be your secret ( $D$  is an integer without loss of generality)
- Choose a prime  $p > \max(D, n)$
- $g(x)$  is a random polynomial of degree  $k-1$  so that  $g(x) = \sum_{i=0}^{k-1} a_i x^i$ 
  - $a_0 = D$ , and  $a_i, i \in \{1, \dots, k-1\}$  are chosen with a uniform distribution on  $[0, p[$

$$g(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{k-1} x^{k-1}$$

Secret  $D$

Random



# Implementation (k,n) Threshold Scheme

## □ Create the n fragments

- Compute
  - $D_1 = g(1) \bmod p, D_2 = g(2) \bmod p, \dots, D_n = g(n) \bmod p$
- Distribute the tuples  $(i, D_i)$

# Implementation (k,n) Threshold Scheme

- Retrieve  $D$  based on  $k$  fragments  $(x_i, D_i)$ 
  - Use Lagrange polynomial interpolation to reconstruct the polynomial
    - $g(0) = D$  is the secret

$$g(0) = \sum_{i=0}^k D_i \left( \prod_{j=0, j \neq i}^k \frac{-x_j}{x_i - x_j} \right)$$

# Example for $(k=3, n=5)$

□ The secret is  $D=148$

□ Let's take

- $p=997$  (prime),  $a_1=59$  (random),  $a_2=340$ (random)
- $g(x)=148 + 59x + 340x^2$

□ We compute 5 fragments

- $D_1 = g(1) \bmod 997 = 547$
- $D_2 = g(2) \bmod 997 = 1626 \bmod 997 = 629$
- $D_3 = g(3) \bmod 997 = 3385 \bmod 997 = 394$
- $D_4 = g(4) \bmod 997 = 5824 \bmod 997 = 839$
- $D_5 = g(5) \bmod 997 = 8943 \bmod 997 = 967$

# Example for (k=3,n=5)

- We give to each user a fragment among
  - (1,547), (2,629), (3,394), (4,839), (5,967)
- Assume users with fragments 1,3,4 want to reconstruct the secret
  - They compute  $g(0)$

$$g(0) = 547 \left( \frac{-3}{1-3} \frac{-4}{1-4} \right) + 394 \left( \frac{-1}{3-1} \frac{-4}{3-4} \right) + 839 \left( \frac{1}{4-1} \frac{3}{4-3} \right)$$

$$g(0) = 547 * 2 - 394 * 2 + 839 = 1145$$

$$g(0) \bmod 997 = 148$$

# Properties of Shamir's Scheme

- ❑ The size of each fragment does not exceed the size of the secret (if  $p$  is the same size order as the secret)
- ❑ New fragments can be generated at any time without affecting existing ones
- ❑ All fragments can be changed without changing the secret by generating a new polynomial

# Properties of Shamir's Scheme

- ❑ Possibility of hierarchical schemes by giving a different number of fragments depending on roles (e.g., president 3 fragments, executives 1 fragment)
- ❑ No unproven assumptions (unlike cryptographic or hash protocols)

# Untraceable Message

# Chaum-net [24]

□ Chaum-net = mix-net

- Basis for onion routing

□ Problem

- Alice wants to send a message  $M$  to Bob
  - Assume an unsecure communication network
  - Nobody knows who is the sender (even Bob)
  - Nobody knows who is the receiver (except Alice)
  - Nobody, except Bob, is able to get  $M$



# Notations

- Assume a public key cryptosystem (e.g., RSA)
  - $M$  is a message
  - $K$  is a public key,  $K^{-1}$  is the corresponding private key
  - $K(K^{-1}(M)) = K^{-1}(K(M)) = M$

# Sealed Message

- A message  $M$  is sealed with public key  $K$  if only the holder of  $K^{-1}$  can retrieve  $M$
- $K(M)$  is not sealed because anyone can verify the guess  $K(N) = K(M)$ 
  - Keep in mind that  $M$  might be easy to guess due to its semantic
  - The attacker might know  $M$  and just want to find who is sending it

# Sealed Message

## □ Solution

- Create a large random string  $R$  (e.g., 256 bits large)
- Append  $R$  to the message  $M$ :  $R,M$
- The sealed message is  $K(R,M)$ 
  - As  $R$  is a large random string, not practical to guess  $R,M$
- Once Bob get  $K(R,M)$ 
  - Compute  $K^{-1}(K(R,M)) = R,M$
  - Remove  $R$  (easy if  $R$  is fixed length)

# Mix

## □ A mix is a machine

- Might be a dedicated machine, a router, an end-user in an overlay

## □ Mix purpose

- Hide correspondences between incoming and outgoing messages
  - Not possible to map a source and an outgoing message (apart for the mix)
  - No possible to map a receiver and an incoming message (apart for the mix)

# Trust in Mix

- But, the mix can make the correspondence between incoming and outgoing messages
  - If the mix compromised
    - Possible to know the sender and receiver for each message
    - But, impossible to find what is the message

# Trust in Mix

## □ Use a cascade of mixes

- A single mix in the cascade is enough to hide correspondences between incoming and outgoing messages
- Work with a partially trusted set of mixes
  - As long as one mix in the cascade can be trusted
  - Or, as long as all untrusted mixes in the cascade do not cooperate

# Cascade of Mixes

- ❑ No guarantee that it works
  - Increasing the number of mixes in the cascade
    - Increases the confidence
    - But, increases the end-to-end delay
- ❑ Tor uses at least 3 mixes selected at random (see [50] for details)
  - Called a Circuit
    - Periodically select new random mixes to form a new circuit

# Goal of Chaum-net

Send sealed messages from Alice to Bob through a cascade of mixes



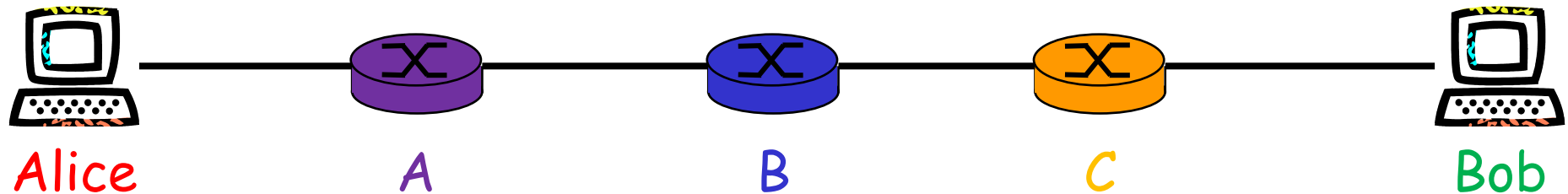
# How It Works?

- Assume an overlay of end-users
  - Each end-user has a couple of private and public keys
    - We note the public key  $K_A$  for end-user  $A$
  - The public keys and the address of owners are publicly available
    - $(K_A, IP_A)$  for each end-user  $A$
    - In a central repository, using a distributed storage, etc.

# How It Works?

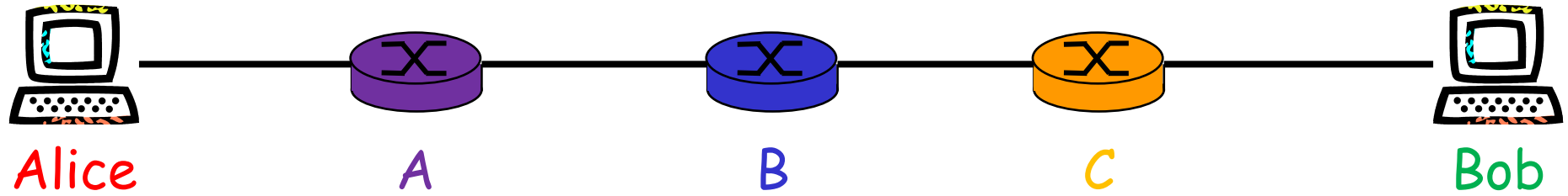
- Alice wants to send the message  $M$  to Bob
  - Any other end-user may act as a mix
  - Alice selects at random a few end-users
    - Get their public key and address
    - Typically select 3 mixes

# How to Send the Message?



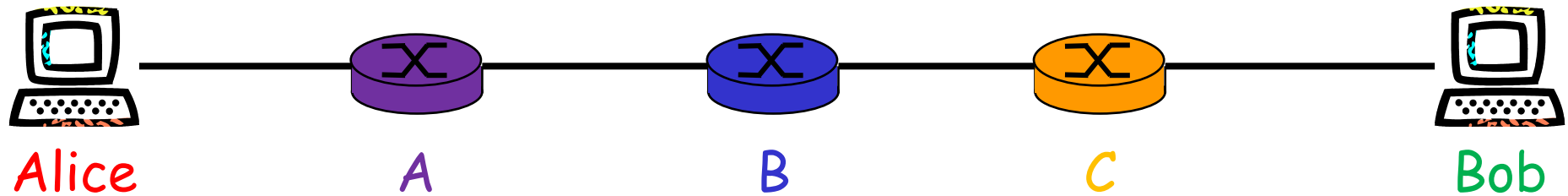
- The path is  $A \rightarrow B \rightarrow C \rightarrow \text{Bob}$ 
  - Create layered (onion) sealed messages from Bob to A

# How to Send the Message?



$$\overbrace{K_{\text{Bob}}(R_0, M)}^{\text{Bob}}$$

# How to Send the Message?

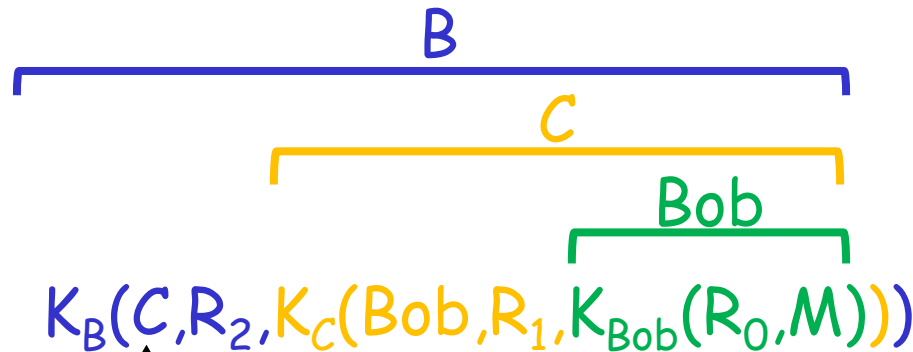
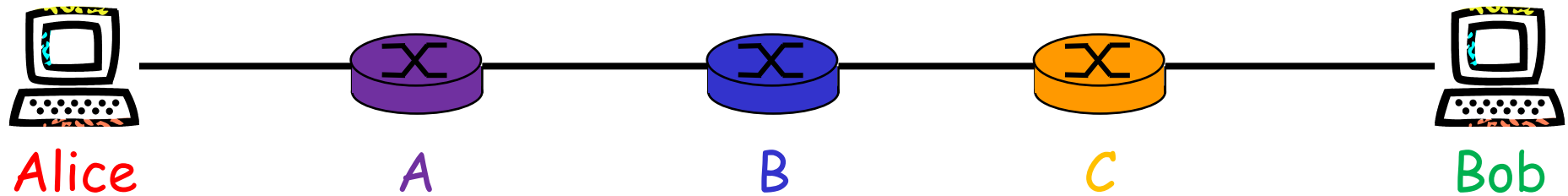


$$K_C(\text{Bob}, R_1, K_{\text{Bob}}(R_0, M))$$

The equation is annotated with brackets. A yellow bracket labeled 'C' spans the entire expression. A green bracket labeled 'Bob' spans the inner expression  $K_{\text{Bob}}(R_0, M)$ .

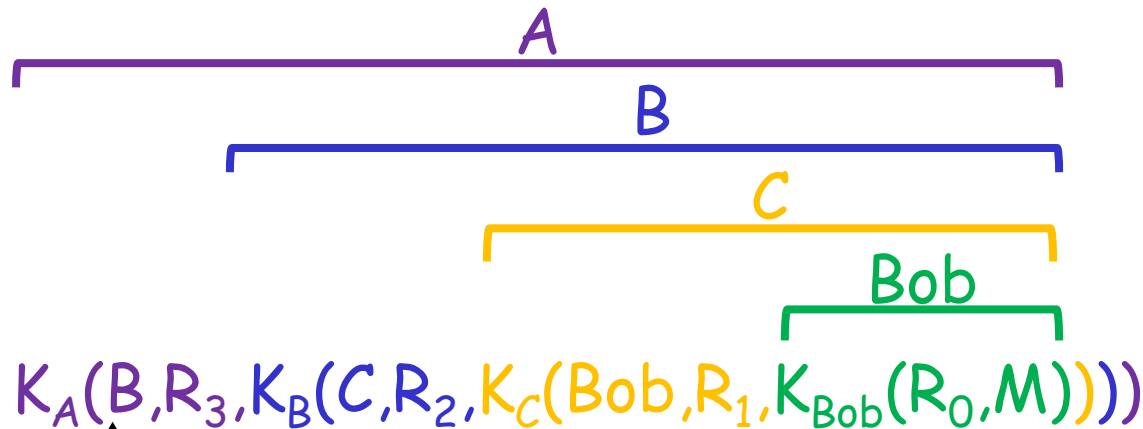
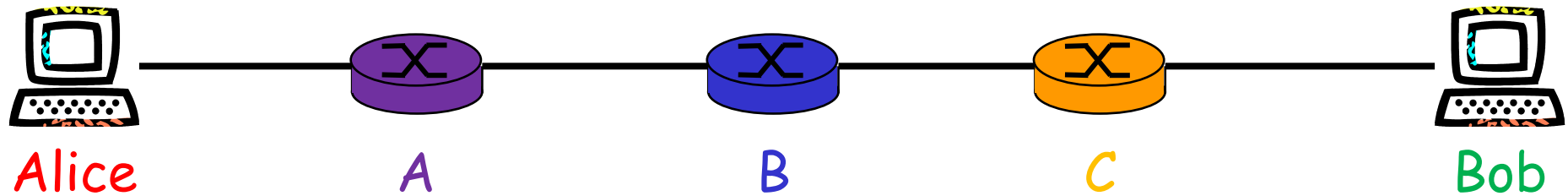
Next hop address

# How to Send the Message?



Next hop address

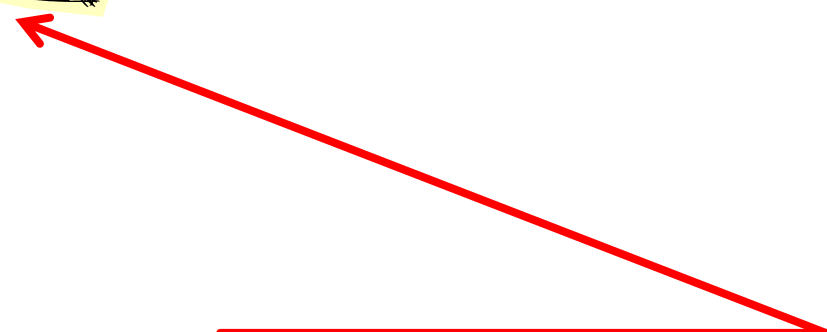
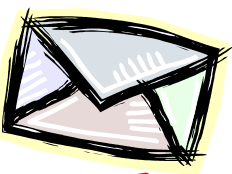
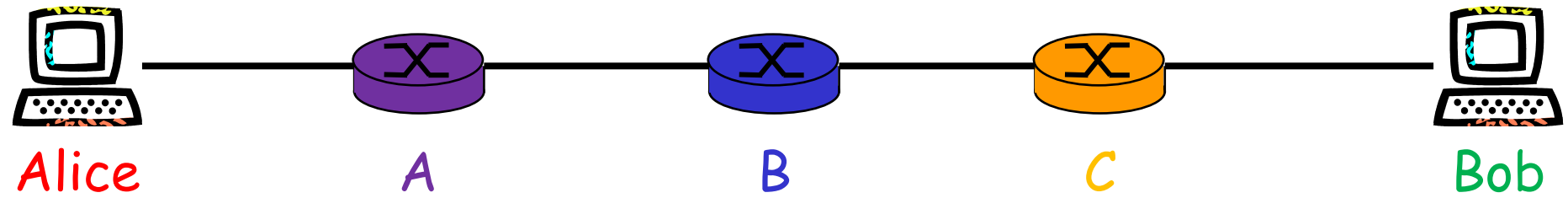
# How to Send the Message?



$$K_A(B, R_3, K_B(C, R_2, K_C(\text{Bob}, R_1, K_{\text{Bob}}(R_0, M))))$$

Next hop address

# How to Relay the Message?

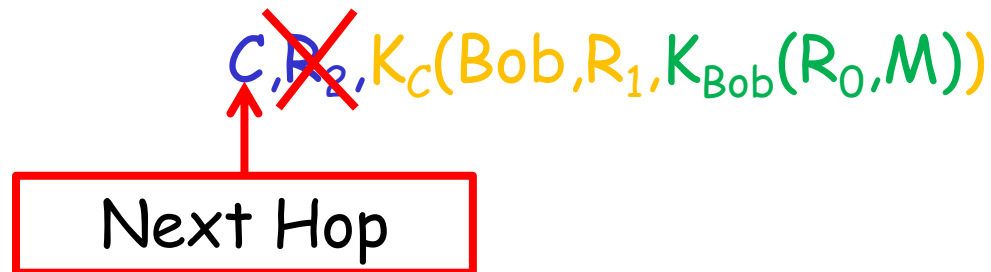
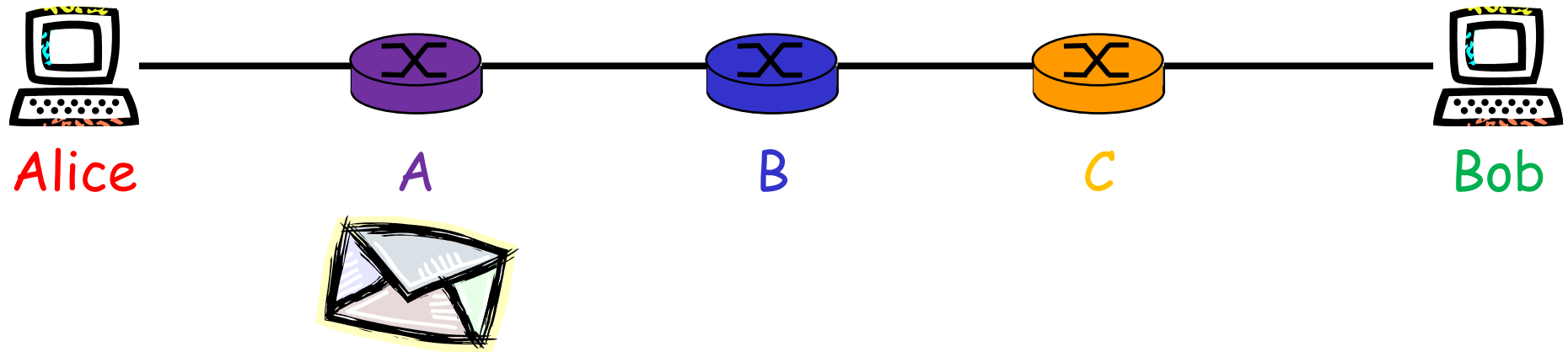


~~$B, R_3, K_B(C, R_2, K_C(\text{Bob}, R_1, K_{\text{Bob}}(R_0, M)))$~~

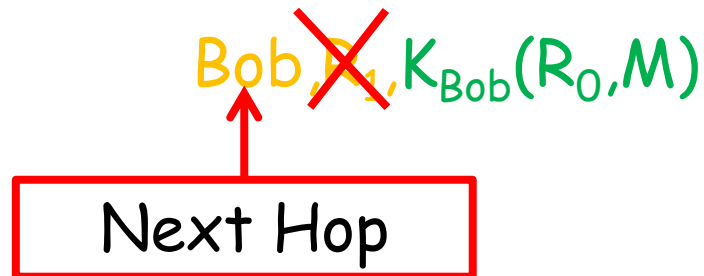
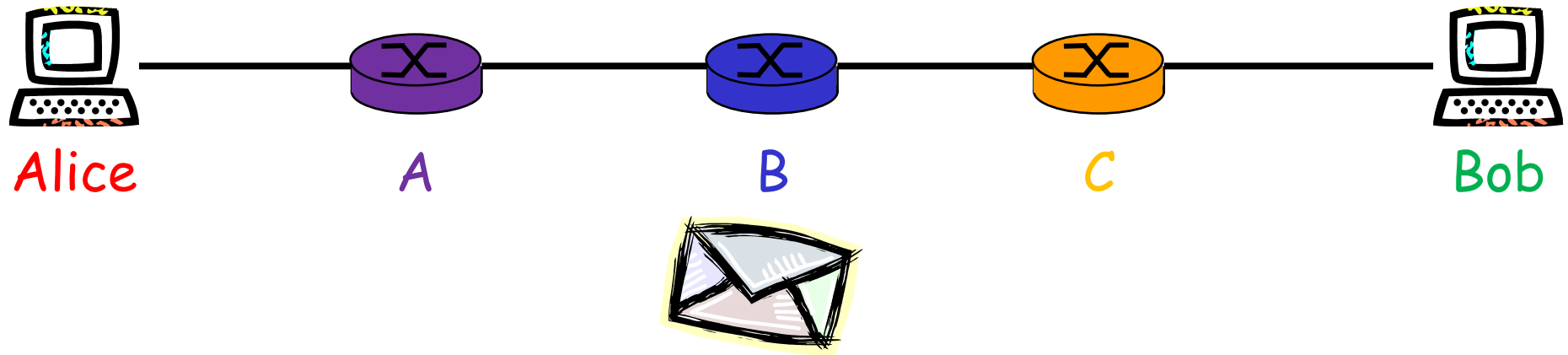
Next Hop



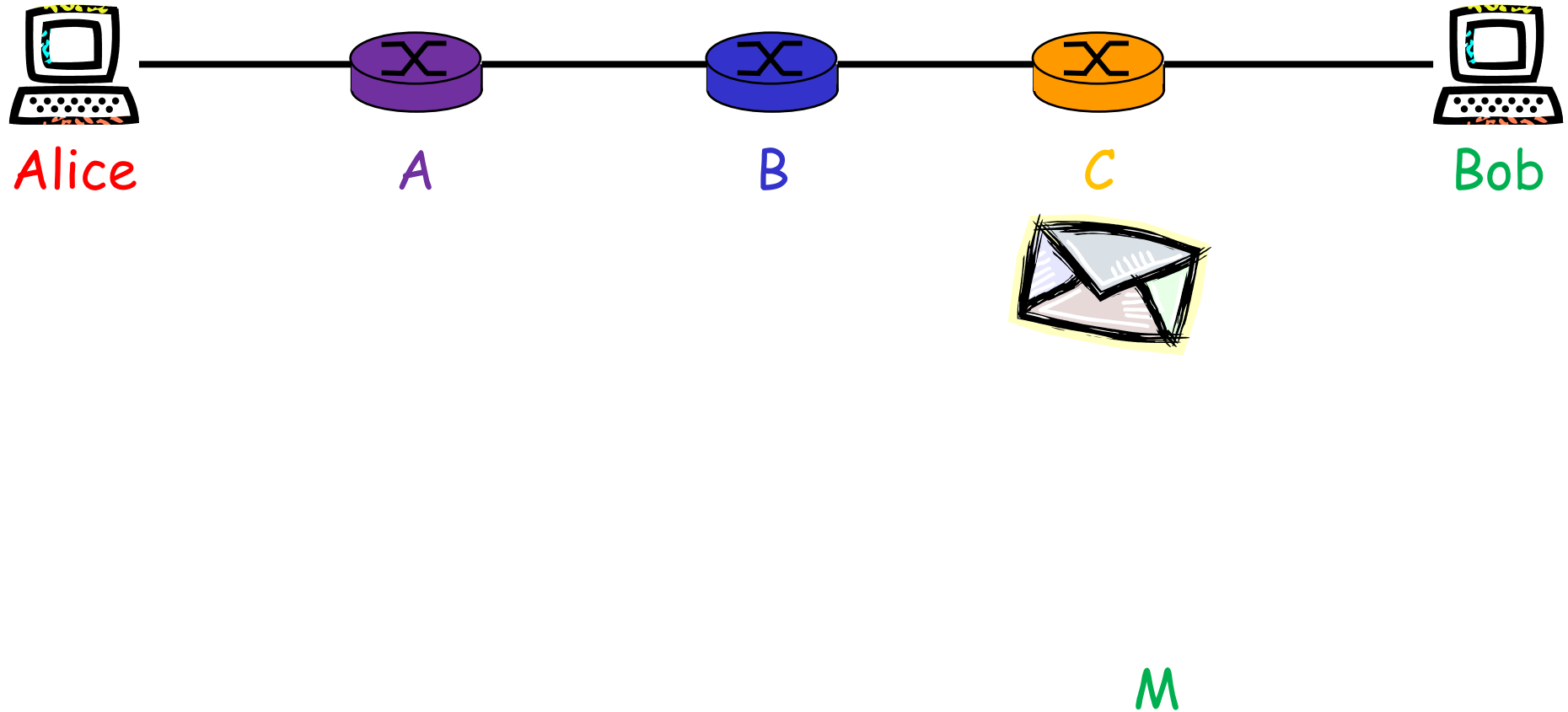
# How to Relay the Message?



# How to Relay the Message?



# How to Relay the Message?



# From Theory To Practice

## □ Attacks still possible

### ▪ Timing

- Correlate when packets are received and sent at each mix
- Can be solved using batches

### ▪ Active end-users

- Possible to know who is sending and who is receiving
- Can be solved with padding, but highly costly

### ▪ Read [24] and [50]

# Outline

- ❑ Overview
- ❑ Content Replication
- ❑ BitTorrent
- ❑ Security
  - Foundations
  - Privacy attacks
- ❑ Localization

# Privacy: Who (Really) Cares?

- Privacy has an ambivalent status
  - Law/states **fails** to protect privacy of users in the internet
    - It is on a per-country basis, but the internet is worldwide
  - Users **spread** personal information all over the internet
    - Anonymizing **IP addresses** is ineffective
    - Extremely complex to preserve privacy

# Why Should I Care?

- ❑ “I do nothing illegal”
  - Where?
  - No problem to publish all your browsing history of the past 2 years?
- ❑ “I do not leave any personal information”
  - Are you using Google?
- ❑ “I don't have any immoral activities”
  - Morality is vastly different from countries to countries
    - Facebook breast-feeding vs. racism

# Privacy: Why Is It So Complex?

- ❑ Privacy is no more a protocol or system issue only
- ❑ Protocols and systems **interact** in many complex ways
  - Might be **closed** systems (facebook, google, skype, etc.)
  - Might have many **implementation flavors** (BitTorrent, HTML, etc.)



# Spying the World From Your Laptop [52]

# Why BitTorrent?

- ❑ BitTorrent **widely popular**
  - Several 10M of users at any moment in time
  - Several 100M of users cumulated over months
- ❑ BitTorrent **most efficient** P2P protocol
  - The only one candidate for legal P2P delivery

What is the privacy implication of BitTorrent usage?

# BitTorrent Overview

[Search Torrents](#) | [Browse Torrents](#) | [Recent Torrents](#) | [TV shows](#) | [Music](#) | [Top 100](#)

Pirate Search

Audio  Video  Applications  Games  Other All

## Details for this torrent

### Alice.In.Wonderland.2010.720p.BluRay.x264-CBGB

Type:	<a href="#">Video &gt; Highres - Movies</a>	Quality:	+16 / -4 (+12)
Files:	<a href="#">2</a>	Uploaded:	2010-05-13 01:51:19 GMT
Size:	4.37 GiB (4694255747 Bytes)	By:	<a href="#">GoodFilms</a> 
Info:	<a href="#">IMDB</a>	Seeders:	2411
		Leechers:	11203
		Comments:	42

[Download](#)

Enjoy Movies, TV Shows, Music and Games on your browser!

[DOWNLOAD THIS TORRENT](#) ([MAGNET LINK](#))

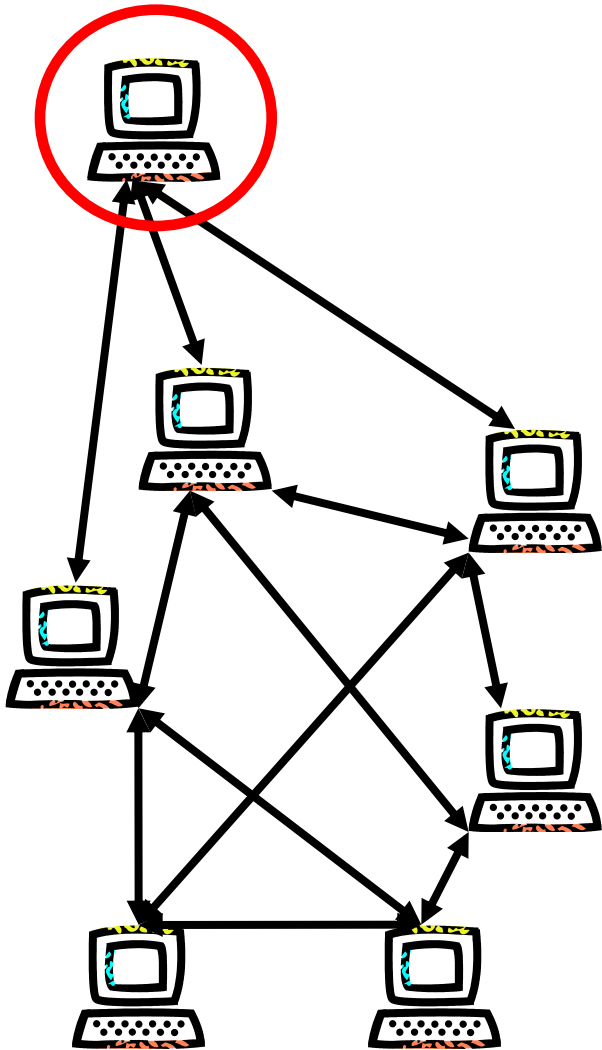
HeXagon.cc <http://hd.heXagon.cc>

```
runtime : 109 min
video : 4256 kbps
audio : 1509 kbps DTS English
resolution : 1280x720
size : 4479 MB
subs : English
```

<http://www.imdb.com/title/tt1014759/>

<http://thepiratebay.org/torrent/5193427/> More 720p Movie Torrents

# BitTorrent Overview



- Who inserts contents?
- Who is downloading what?

# Why Is It Hard?

- ❑ No way to get this information directly
  - Very good engineering of the implementations
  - Many **blacklisting** policies
- ❑ Need to correlate many different sources of information
  - Deep understanding of protocols and implementations
    - Experiments and measurements

# Why Is It Hard?

## □ Design goal

- Data collection without dedicated infrastructure and without being blacklisted

## □ High volume of data

- 148M IP addresses \* 1.2M contents
  - 2000M downloads
  - 3TB of storage on a NAS

Challenge in collecting and analyzing the data

# Who inserts contents?

# State of the Art

## □ BitTorrent

- Introduction in 2000
- Half of the Internet traffic in 2004

## □ Nobody ever looked at content providers


- Believed to be impossible
  - Initial private phase for torrents
  - Content providers lies on their status



# Method: First Minute

- ❑ Join torrents within its first minute
  - After announcement on TPB web site
  - If we are alone with another peer
    - It is the initial seed
  
- ❑ Fails for most interesting torrents

# Method: Correlation




[Search Torrents](#) | [Browse Torrents](#) | [Recent Torrents](#) | [TV shows](#) | [Music](#) | [Top 100](#)

  
  
 Audio  Video  Applications  Games  Other 

## Details for this torrent

**Alice.In.Wonderland.2010.720p.BluRay.x264-CBGB**

Type:	<a href="#">Video &gt; Highres - Movies</a>	Quality:	+16 / -4 (+12)
Files:	2	Uploaded:	2010-05-13 01:51:19 GMT
Size:	4.37 GiB (4694255747 Bytes)	By:	<b>GoodFilms</b> 
Info:	<a href="#">IMDB</a>	Seeders:	2411
		Leechers:	11203
		Comments:	42

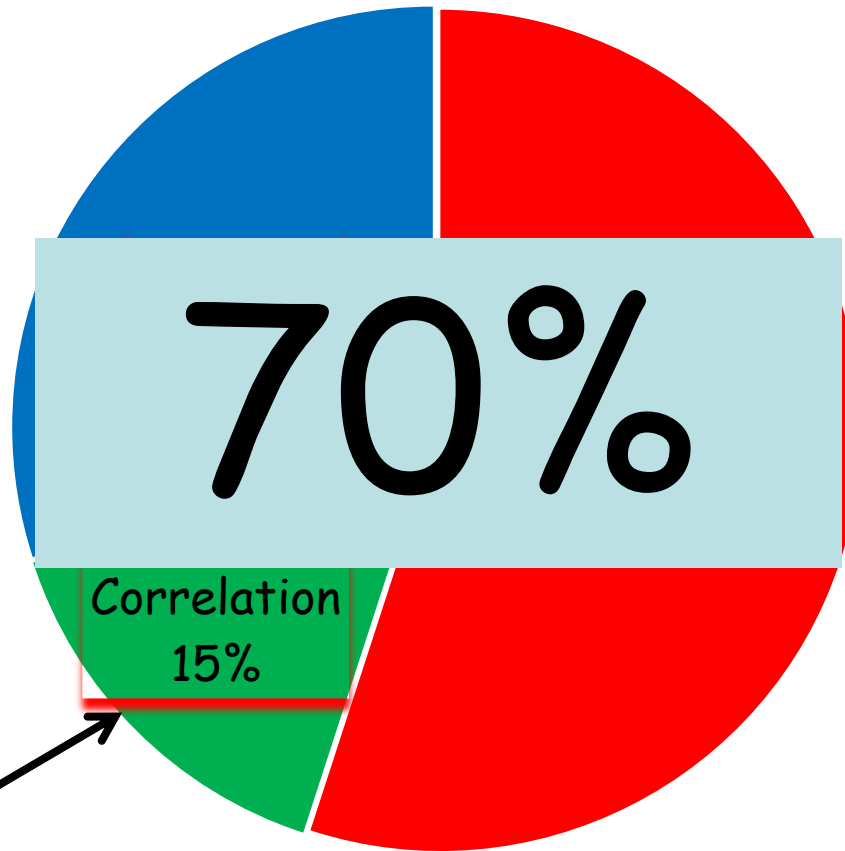
Enjoy Movies, TV Shows, Music and Games on your browser!

[DOWNLOAD THIS TORRENT](#)  [MAGNET LINK](#)

```
HeXagon.cc http://hd.heXagon.cc
runtime : 109 min
video : 4256 kbps
audio : 1509 kbps DTS English
resolution : 1280x720
size : 4479 MB
subs : English

http://www.imdb.com/title/tt1014759/
http://thepiratebay.org/torrent/5193427/ More 720p Movie Torrents
```

# Success of the Method



Most interesting torrents

# Who is downloading what?

# Method

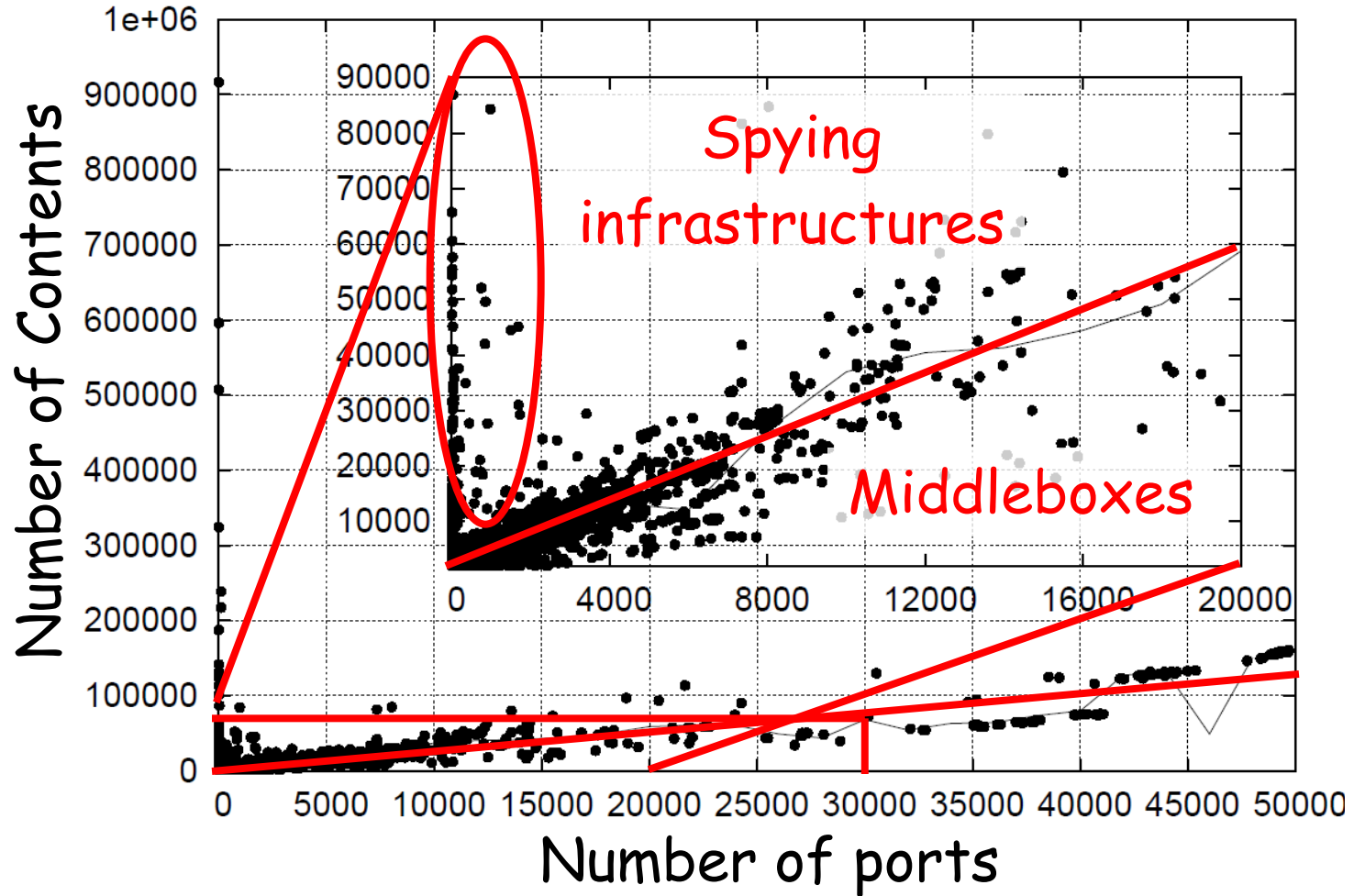
- ❑ Retrieved continuously IP addresses of most BitTorrent peers
  - For 103 days, every two hours
    - 700 000 torrents per snapshot
    - 5M to 10M IP addresses
  - 148M IP addresses in 1.2M torrents downloading 2000M of contents
- ❑ Analysis of such a large amount of data is complex

# How To Identify Heavy Downloaders?

- Lets take top 10,000 IP addresses
  - Subscribed to at least 1636 contents

Do we find the heavy downloaders?

# Who Are These Peers?



# Outline

- Overview
- Content Replication
- BitTorrent
- Security
- Localization



# Localization

## □ To read

- Chord[25]
- Impact of DHT routing geometry [29]

## □ Interesting read (because well known)

- Kademlia[27]: used in BitTorrent and Emule
- Pastry[26]
- CAN[28]

Thank you for attending this  
course

# Sources of this Presentation

Note: References in bold are highly recommended reads.

- ❑ [1] Cache Logic <http://www.cachelogic.com/research/>
- ❑ [2] Keith W. Ross and Dan Rubenstein "P2P Systems". Infocom 2004 tutorial. <http://cis.poly.edu/~ross/tutorials/P2PtutorialInfocom.pdf>
- ❑ [3] S. Sen and Jia Wang "Analysing peer-to-peer traffic across large networks". ACM SIGCOMM'02
- ❑ [4] T. Karagiannis, A. Broido, M. Faloutsos, Kc Claffy "Transport Layer Identification of P2P Traffic". ACM IMC'04
- ❑ [5] X. Yang and G. de Veciana "Service Capacity of Peer to Peer Networks". IEEE Infocom'04
- ❑ [6] D. Qiu and R. Srikant "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks". ACM SIGCOMM'04
- ❑ [7] J. H. Saltzer, D. P. Reed, and D. D. Clark "End-to-end arguments in system design". *ACM Transactions on Computer Systems* 2, 4 (November 1984) pages 277-288
- ❑ [8] P. Rodriguez, E. W. Biersack "Dynamic Parallel Access to Replicated Content in the Internet". *IEEE/ACM Transactions on Networking*, August 2002 (Also in *IEEE/Infocom 2000*)
- ❑ [9] E. W. Biersack, P. Rodriguez, P. Felber "Performance Analysis of Peer-to-Peer Networks for File Distribution". Research Report RR-04-108. April 2004.
- ❑ [10] P. A. Felber and E. W. Biersack. Self-scaling Networks for Content Distributions. In Ozalp Babaoglu et al., editors, *Self-Star Properties in Complex Information Systems*, volume 3460 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.
- ❑ [11] E. K. Lua et al. "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes", IEEE Communications survey and tutorial, March 2004.
- ❑ [12] Jian Liang, Rakesh Kumar, Keith Ross, "The KaZaA Overlay: A Measurement Study", *Computer Networks* (Special Issue on Overlays), to appear.
- ❑ [13] Y. Kulbak, D. Bickson "The eMule Protocol Specification" January 2005
- ❑ [14] C. Gkantsidis, P. Rodriguez "Network Coding for Large Scale Content Distribution"

# Sources of this Presentation

Note: References in bold are highly recommended reads.

- ❑ [15] Dejan Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh" *SOSP'03*, October 2003.
- ❑ [16] T. Klingberg, R. Manfredi, "Gnutella Protocol Development v0.6", June 2002
- ❑ [17] T. Klingberg, "Partial File Sharing Protocol", August 2002
- ❑ [18] A. Legout, G. Urvoy-Keller, and P. Michiardi. "Understanding BitTorrent: An Experimental Perspective". *Technical Report (inria-00000156, version 3 - 9 November 2005)*, INRIA, Sophia Antipolis, November 2005.
- ❑ [18] BitTorrent Protocol Specification v1.0. <http://wiki.theory.org/BitTorrentSpecification>
- ❑ [19] **Bram Cohen, "Incentives Build Robustness in BitTorrent", May 2003**
- ❑ [20] J.A Pouwelse et al., "The BitTorrent P2P File-Sharing System: Measurements and Analysis", IPTPS 2005
- ❑ [21] M. Izal et al., "Dissecting BitTorrent: Five Months in a Torrent's Lifetime", PAM 2004
- ❑ [22] L. Guo et al., "Measurements, Analysis, and Modeling of BitTorrent-like Systems" IMC 2005
- ❑ [23] **Shamir "How to Share a Secret" Communications of the ACM, 1979**
- ❑ [24] **D. L. Chaum "Untraceable Electronic Mail, Return Addresses and Digital Pseudonyms" Communications of the ACM, 1981**
- ❑ [25] **Stoica et al. "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications" ACM SIGCOMM'01**
- ❑ [26] Rowstron and Druschel "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems"
- ❑ [27] Maymounkov and Mazières "Kademlia: A peer-to-peer Information System Based on the XOR Metric"
- ❑ [28] Ratnasamy et al. "A Scalable Content-Addressable Network", SIGCOMM'01
- ❑ [29] **Gummadi et al. "The Impact of DHT Routing Geometry on Resilience Proximity"**

# Sources of this Presentation

Note: References in bold are highly recommended reads.

- ❑ [30] C. Fragouli, J.-Y. Le Boudec and J. Widmer "Network Coding: An Instant Primer". *ACM Sigcomm Computer Communication Review*, Vol. 36, Nr. 1, pp. 63-68, 2006.
- ❑ [31] C. Gkantsidis, J. Miller, P. Rodriguez "Comprehensive view of a Live Network Coding P2P system". *ACM SIGCOMM/USENIX IMC'06*, Brasil. Oct 2006.
- ❑ [32] C. Gkantsidis, P. Rodriguez "Cooperative Security for Network Coding File Distribution". *IEEE/INFOCOM'06*, Barcelona, April 2006.
- ❑ [33] A. R. Bharambe, C. Herley, and V. N. Padmanabhan "Analyzing and Improving a BitTorrent Network's Performance Mechanisms". In *Proc. of Infocom'06*, Barcelona, Spain, April 2006.
- ❑ [34] **A. Legout, G. Urvoy-Keller, and P. Michiardi "Rarest First and Choke Algorithms Are Enough". In *Proc. of ACM SIGCOMM/USENIX IMC'2006*, Rio de Janeiro, Brazil, October 2006.**
- ❑ [35] **A. Legout, N. Liogkas, E. Kohler, and L. Zhang "Clustering and Sharing Incentives in BitTorrent Systems". *Technical Report (inria-00112066, version 1 - 21 November 2006)*, INRIA, Sophia Antipolis, November 2006.**
- ❑ [36] S. Jun and M. Ahamad."Incentives in BitTorrent Induce Free Riding". In *Proc. of the Workshop on Economics of Peer-to-Peer Systems (P2PEcon'05)*, Philadelphia, PA, August 2005.
- ❑ [37] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. "Exploiting Bittorrent For Fun (But Not Profit)". In *Proc. of IPTPS'06*, Santa Barbara, CA, February 2006.
- ❑ [38] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. "Free Riding in BitTorrent is Cheap." In *Proc. of HotNets-V*, Irvine, CA, November 2006.
- ❑ [39] J. Shneidman, D. Parkes, and L. Massoulié. "Faithfulness in Internet Algorithms". In *Proc. of the Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS'04)*, Portland, OR, September 2004.
- ❑ [40] B Fan, DM chiu and JCS Lui, "The Delicate Tradeoff of BitTorrent-like File Sharing Protocol Design", *IEEE ICNP 2006*

# Sources of this Presentation

Note: References in bold are highly recommended reads.

- ❑ [41] Kenjiro Cho, Kensuke Fukuda, Hiroshi Esaki, Akira Kato "Observing Slow Crustal Movement in Residential User Traffic". CoNext'2008, December 2008.
- ❑ [42] C. Zhang, P. Dughel, D. Wu, K.W. Ross, "Unraveling the BitTorrent Ecosystem". To appear in IEEE Transactions on Parallel and Distributed Systems.
- ❑ [43] Stevens Le Blond, Arnaud Legout, Walid Dabbous. "Pushing BitTorrent Locality to the Limit". Computer Networks, October 2010, ISSN 1389-1286, DOI: 10.1016/j.comnet.2010.09.014.
- ❑ [44] Anwar Al Hamra, Nikitas Liogkas, Arnaud Legout, Chadi Barakat. "Swarming Overlay Construction Strategies". In *Proc. of ICCCN'2009*, August 2--6, 2009, San Francisco, CA, USA.
- ❑ [45] T. Karagiannis, P. Rodriguez, and K. Papagiannaki. "Should internet service providers fear peer-assisted content distribution?" In *Proc. of IMC'05*, Berkeley, CA, USA, October 2005.
- ❑ [46] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz. "P4p: Provider portal for applications." In *Proc. of ACM SIGCOMM*, Seattle, WA, USA, August 2008.
- ❑ [47] D. R. Choffnes and F. E. Bustamante. "Taming the torrent: A practical approach to reducing cross-isp traffic in p2p systems." In *Proc. of ACM SIGCOMM*, Seattle, WA, USA, August 2008.
- ❑ [48] The BitTorrent Protocol Specification, BEP 3, [http://www.bittorrent.org/beps/bep\\_0003.html](http://www.bittorrent.org/beps/bep_0003.html)
- ❑ [49] Peer ID Conventions , BEP 20, [http://www.bittorrent.org/beps/bep\\_0020.html](http://www.bittorrent.org/beps/bep_0020.html)
- ❑ [50] R. Dingledine, N. Mathewson, P. Syverson. "Tor: The Second-Generation Onion Router." In *proc. of Usenix Security'2004*, San Diego, CA, USA, August 2004.
- ❑ [51] Niels Ferguson, Bruce Schneier, Tadayoshi Kohno. "Cryptography Engineering." 2010, Wiley.
- ❑ [52] Stevens Le Blond, Arnaud Legout, Fabrice Lefessant, Walid Dabbous, Mohamed Ali Kaafar. "Spying the World from your Laptop - Identifying and Profiling Content Providers and Big Downloaders in BitTorrent." In *Proc. of LEET'10*, April 27, 2010, San Jose, CA, USA.

# Internet Timeline

From 1962 to 1991

Disclaimer: I found this timeline long ago, but I don't remember where. If you did it and want to be credited for it, send me an email.

# Internet Timeline

- 1962 **Kleinrock** thesis describes underlying principles of packet-switching technology
- 1966 **ARPANET** project
  - Larry Roberts of MIT's Lincoln Lab is hired to manage the ARPANET project.
  - ARPA computer network, a packet-switched network with minicomputers acting as gateways for each node using a standard interface.
- 1967 **Packet switching**
  - Donald Davies, of the National Physical Laboratory in Middlesex, England, coins the term *packet switching* to describe the lab's experimental data transmission.



# Internet Timeline

- **1968 Interface message processors**
  - Bolt Beranek and Newman, Inc. (BBN) wins a DARPA contract to develop the packet switches called interface message processors (IMPs).
- **1969 DARPA deploys the IMPs**
  - First transmission between UCLA and Stanford: "lo"
- **1970 Initial ARPANET host-to-host protocol**
  - Network Working Group (NWG), formed at UCLA by Steve Crocker, deploys the initial ARPANET host-to-host protocol, called the Network Control Protocol (NCP). The primary function of the NCP is to establish connections, break connections, switch connections, and control flow over the ARPANET, which grows at the rate of one new node per month.

# Internet Timeline

- ❑ **1972 First e-mail program**
  - **Ray Tomlinson** at BBN writes the first e-mail program to send messages across the ARPANET. In sending the first message to himself to test it out, he uses the @ sign—the first time it appears in an e-mail address.
- ❑ **1972 First public demonstration of the new network technology**
  - Robert Kahn at BBN, who is responsible for the ARPANET's system design, organizes the first public demonstration of the new network technology at the International Conference on Computer Communications in Washington, D.C., linking 40 machines and a Terminal Interface Processor to the ARPANET.
- ❑ **1973 Paper describes basic design of the Internet and TCP**
  - **Robert Kahn and Vinton Cerf**, "A Protocol for Packet Network Interconnection" in *IEEE Transactions on Communications*.
- ❑ **1974 F.F. Kuo "ALOHA System", January 1974**

# Internet Timeline

- ❑ 1976 TCP/IP incorporated in Berkeley Unix
- ❑ 1977 **Demonstration of independent networks to communicate**
  - Cerf and Kahn organize a demonstration of the ability of three independent networks to communicate with each other using TCP protocol.
- ❑ 1981 **TCP/IP standard adopted**
  - Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.
  - Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- ❑ 1982 **ARPANET hosts convert to new TCP/IP protocols**
  - All hosts connected to ARPANET are required to convert to the new TCP/IP protocols by January 1, 1983. The interconnected TCP/IP networks are generally known as the Internet.

# Internet Timeline

- **1983 UNIX scientific workstation introduced**
  - Sun Microsystems introduces its UNIX scientific workstation. TCP/IP, now known as the Internet protocol suite, is included, initiating broad diffusion of the Internet into the scientific and engineering research communities
- **1983 The Internet**
  - ARPANET, and all networks attached to it, officially adopts the TCP/IP networking protocol. From now on, all networks that use TCP/IP are collectively known as the Internet. The number of Internet sites and users grow exponentially
- **1984 Advent of Domain Name Service.** Developed by **Paul Mockapetris** and **Craig Partridge**
- **1984 J. H. Saltzer, D. P. Reed, and D. D. Clark** "End-to-end arguments in system design" *ACM Transactions on Computer Systems*, November 1984

# Internet Timeline

- ❑ 1984 **John Nagle** "Congestion Control in IP/TCP Internetworks" October 1984
- ❑ October 1986 First Congestion Collapse
  - From 32 Kbps to 40 bps
- ❑ 1988 **Van Jacobson** "Congestion Avoidance and Control" SIGCOMM'88, August 1988
- ❑ 1991 **World Wide Web software developed**
  - CERN releases the World Wide Web software developed earlier by **Tim Berners-Lee**. Specifications for HTML (hypertext markup language), URL (uniform resource locator), and HTTP (hypertext transfer protocol) launch a new era for content distribution.