ID2210 - Distributed Computing, Peer-to-Peer and GRIDS

Lecture 6

Content Distribution and BitTorrent

[Based on slides by Cosmin Arad]

Today

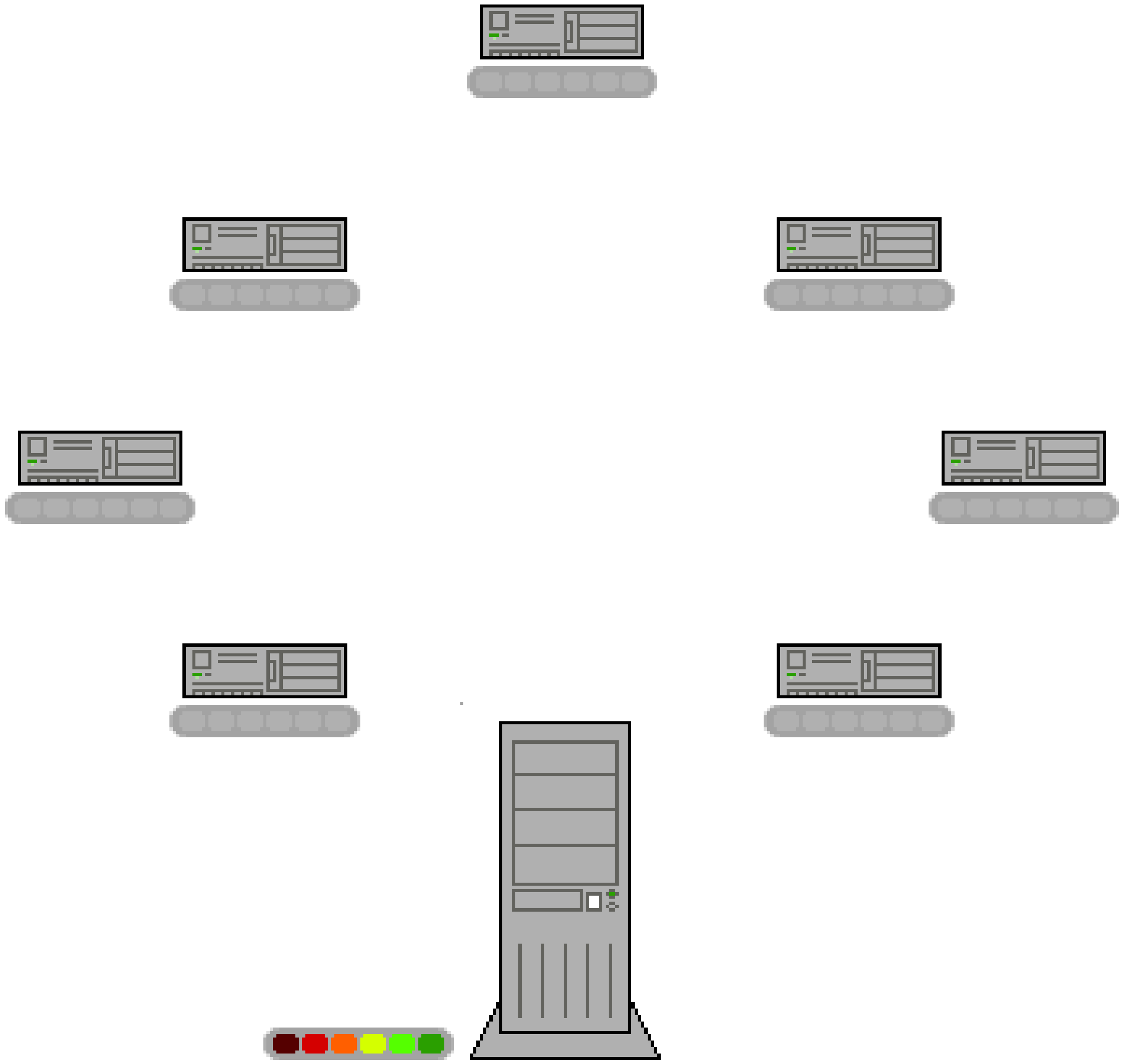
- The problem of content distribution
- A popular solution: BitTorrent
- Underlying incentive scheme
- How BitTorrent works in detail
- Discussion on BitTorrent extensions

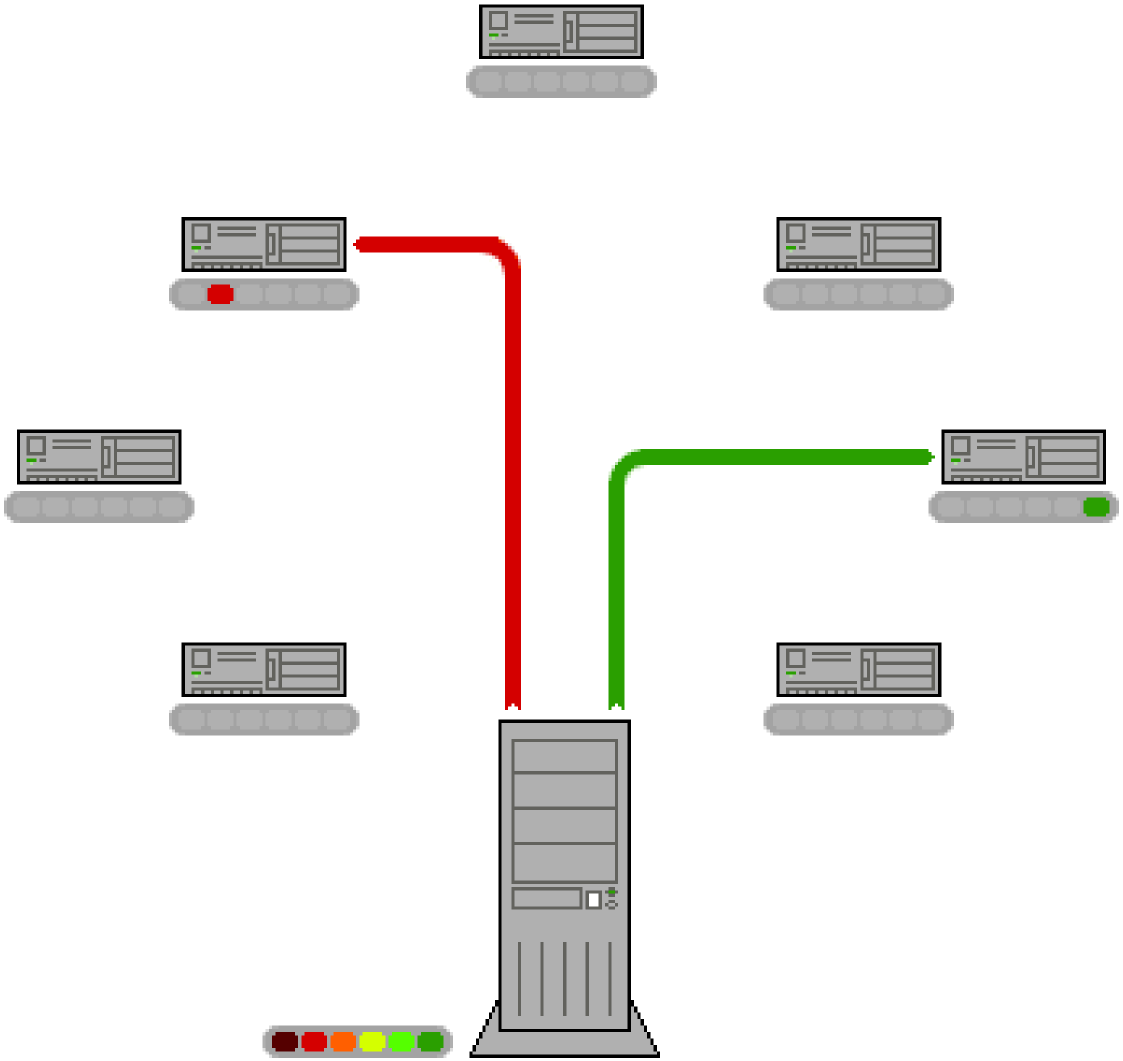
The problem

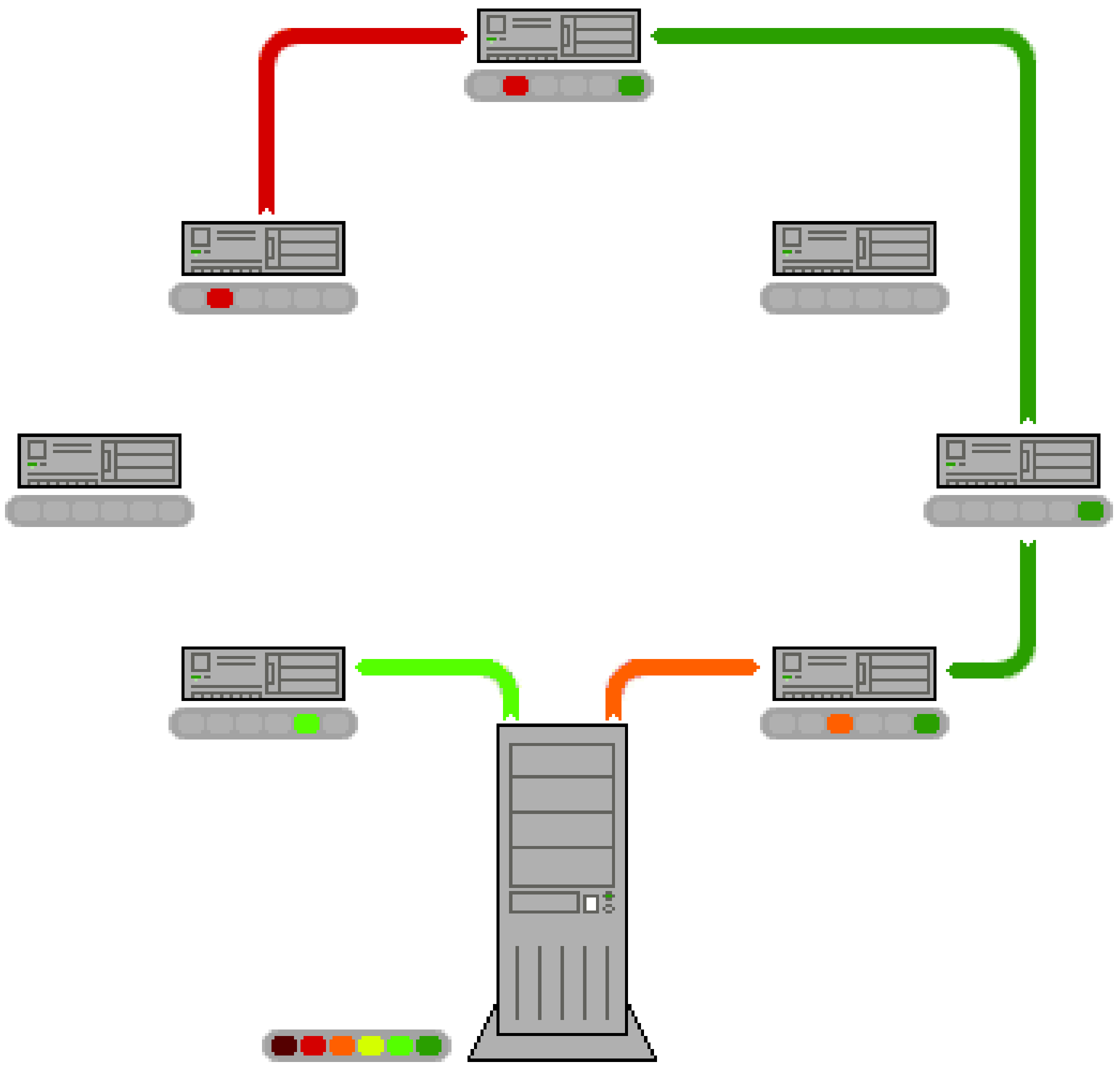
- The distribution of a large piece of static content, from a limited source, to a very large number of users, as fast as possible.
- Providing the necessary upload bandwidth at the source is expensive
- Solutions?

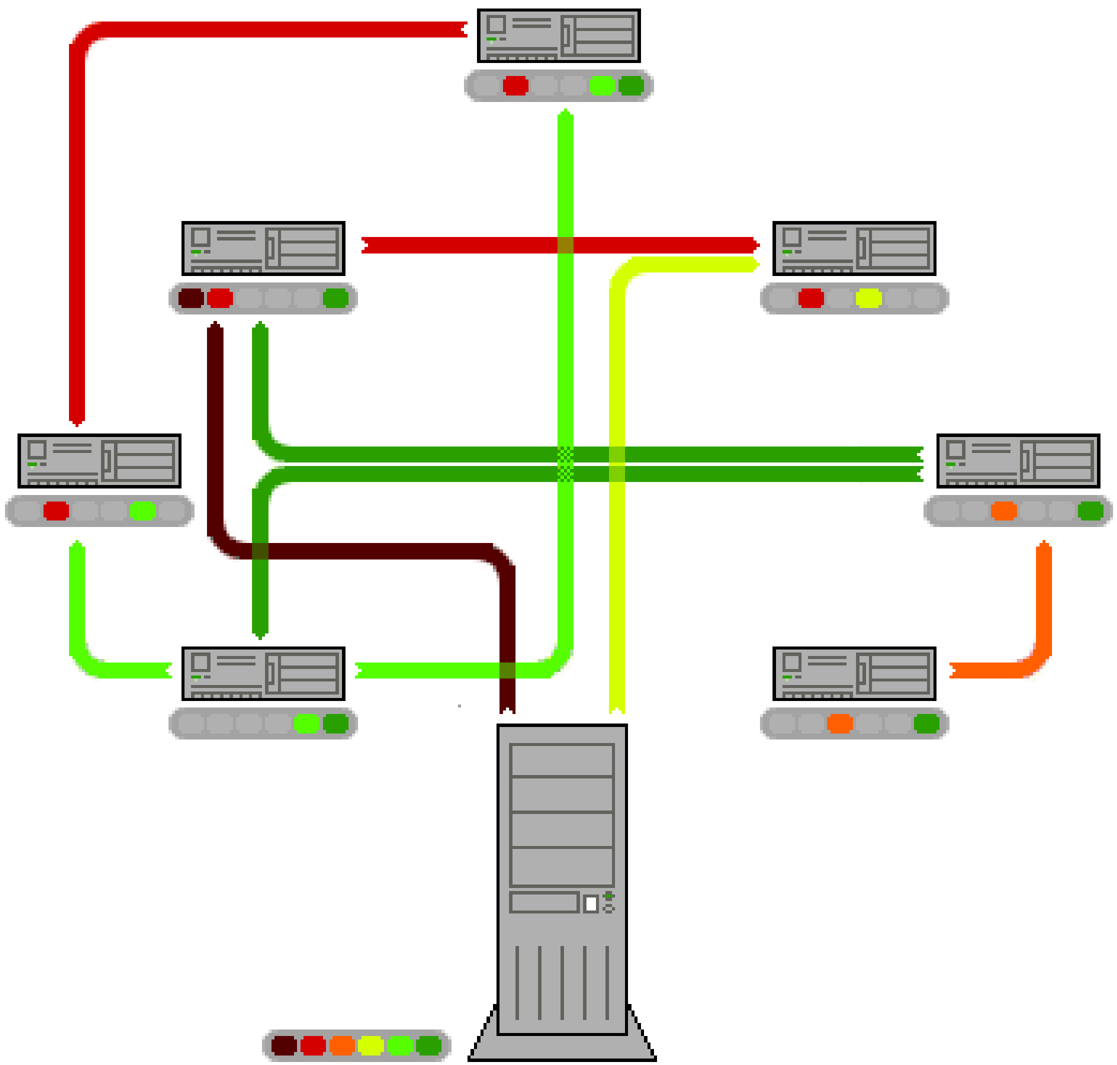
The solution idea

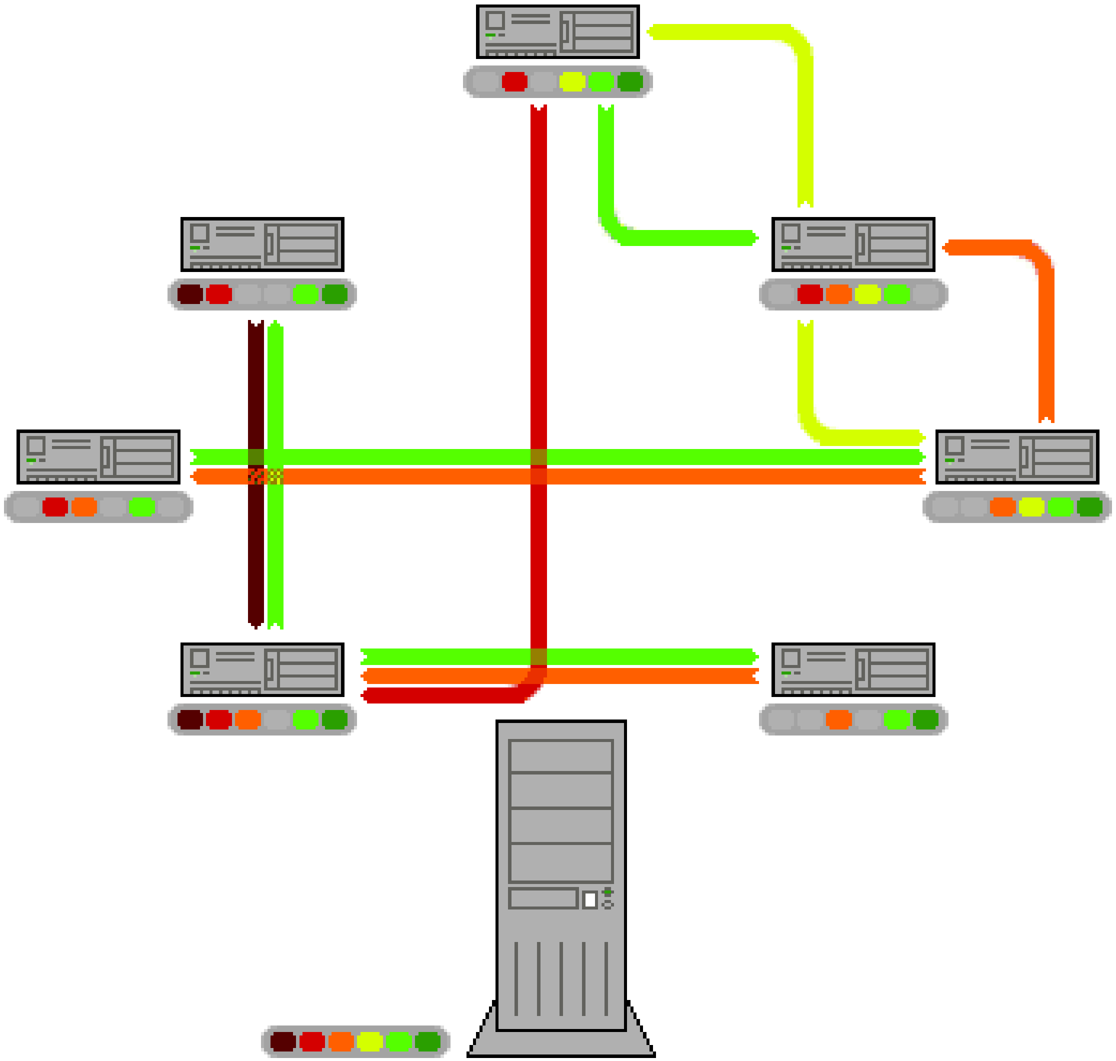
- Use the upload capacity of the downloaders
- Create opportunities for data exchange between downloaders.

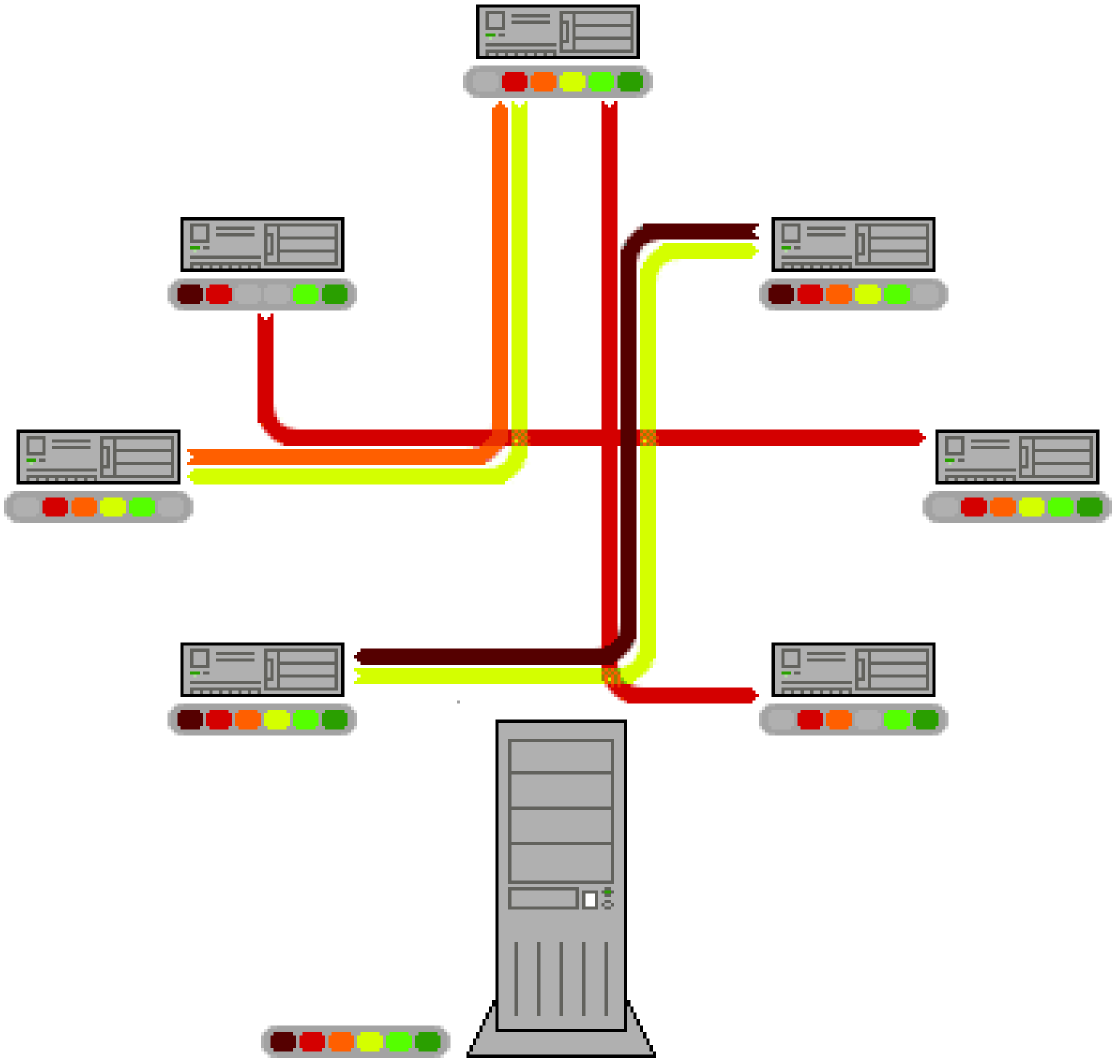


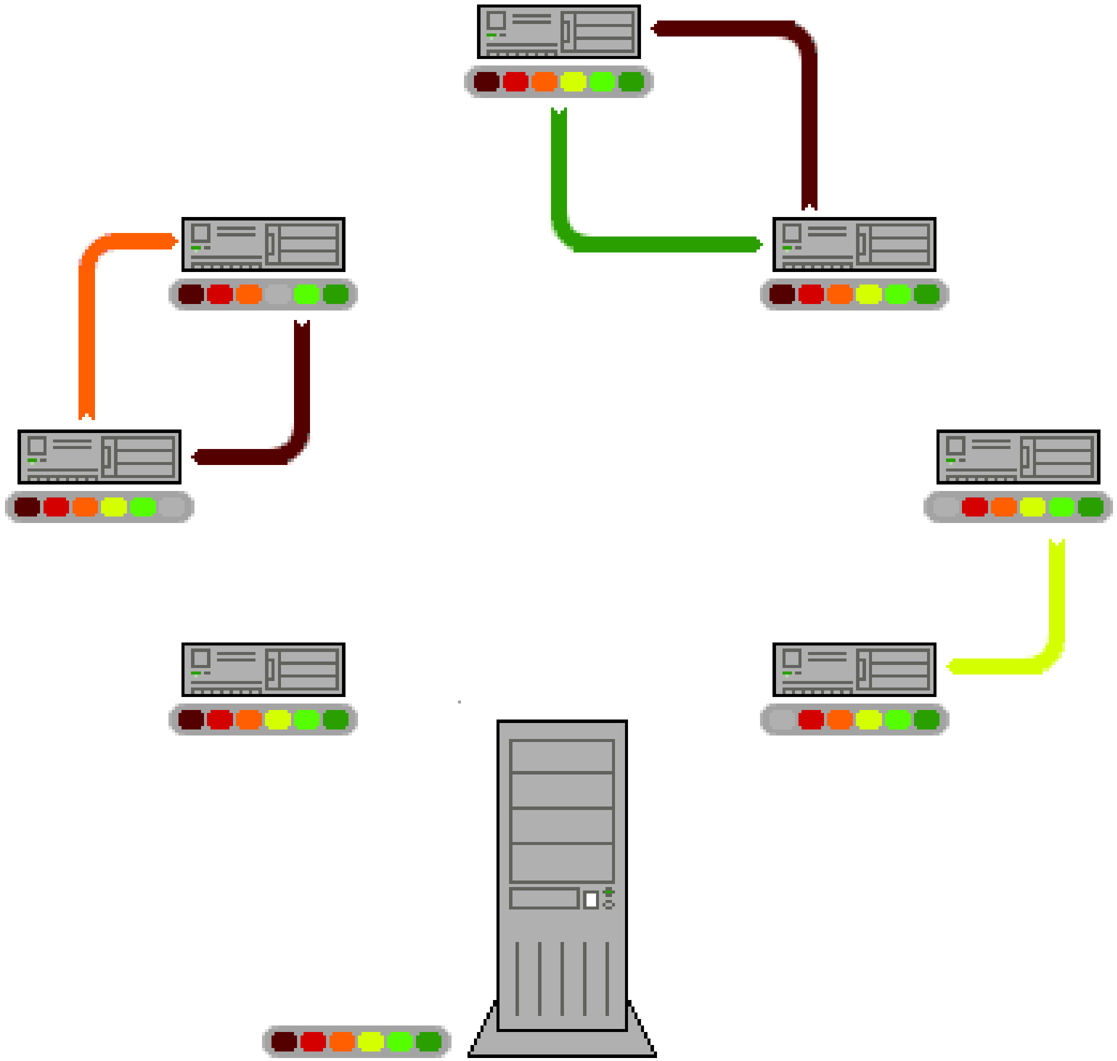


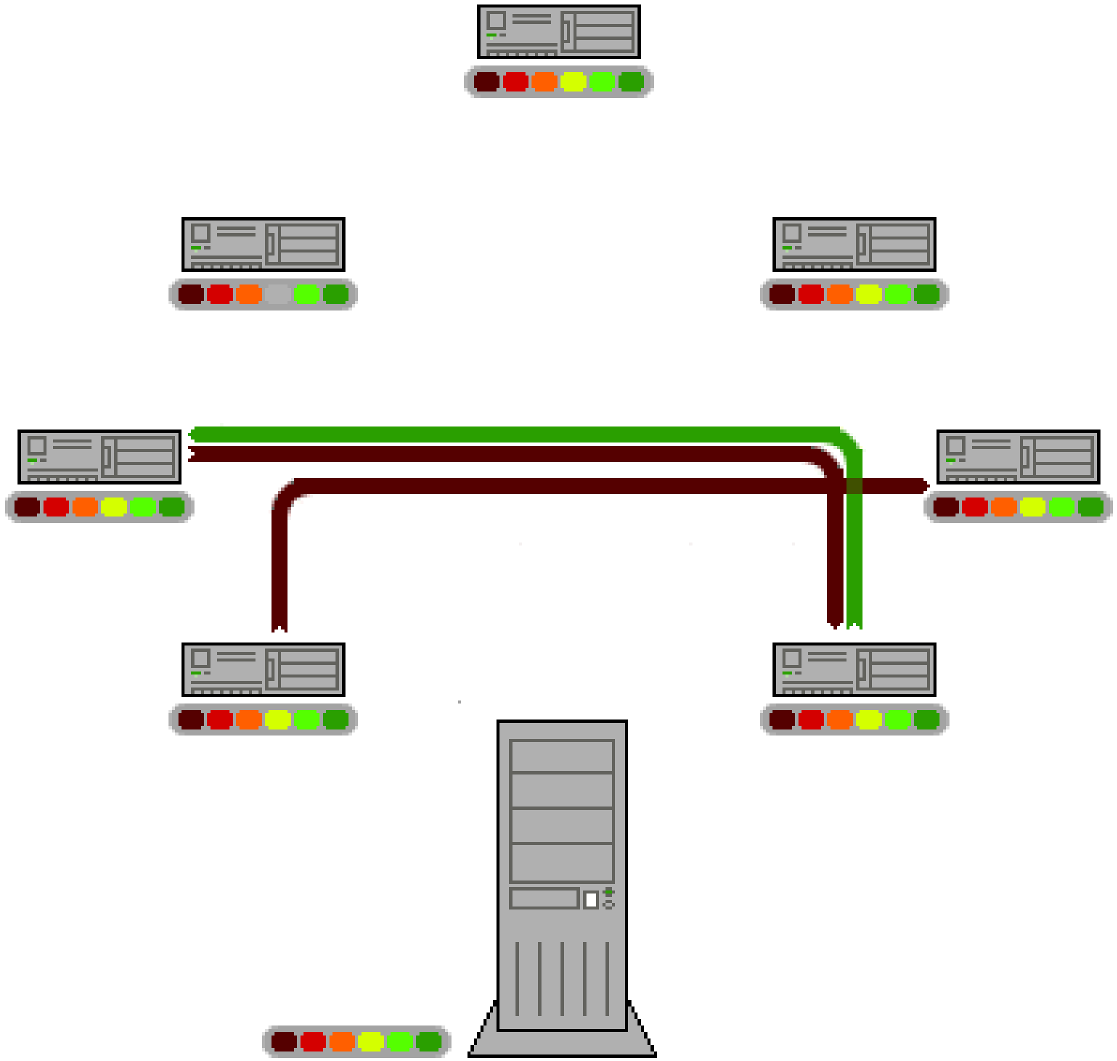


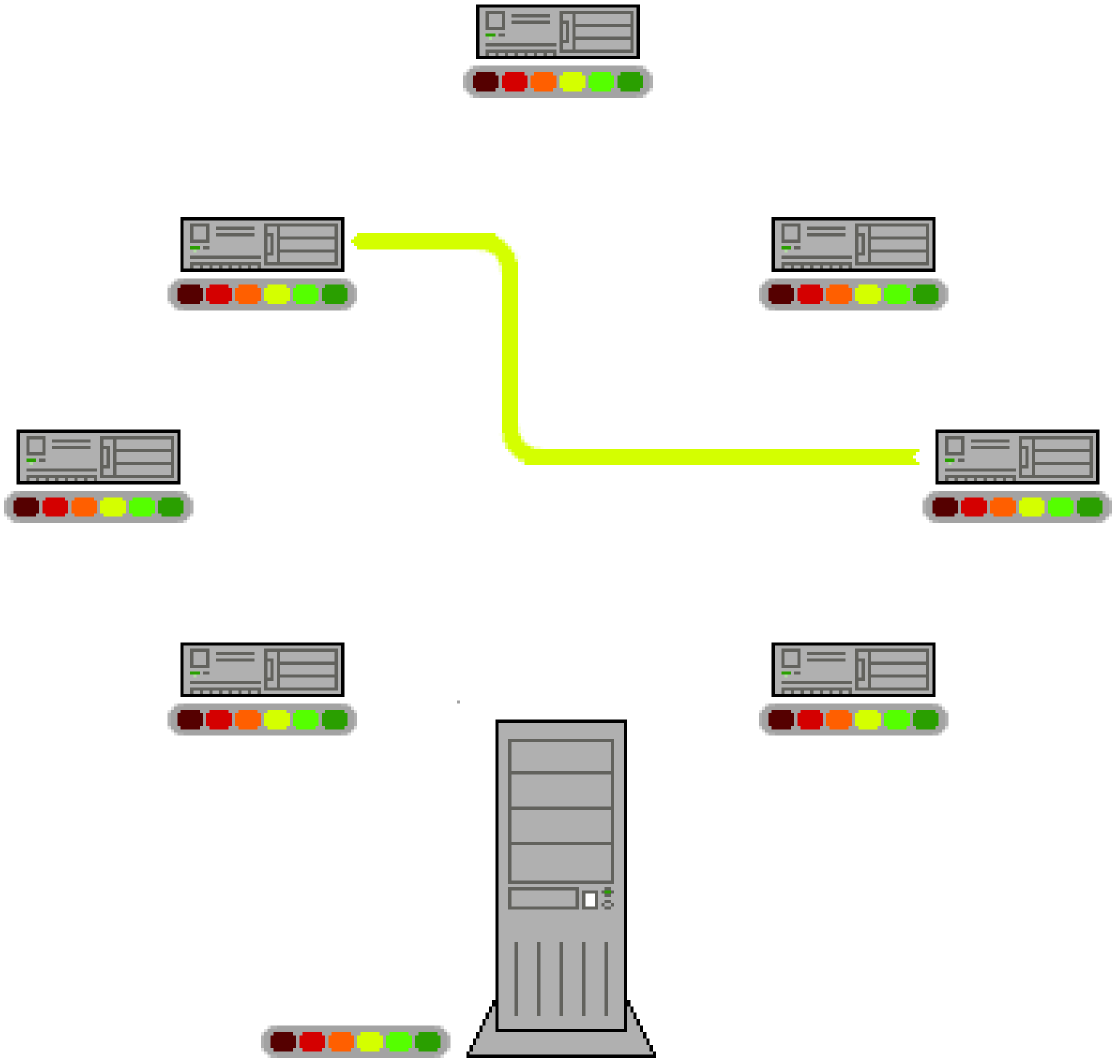


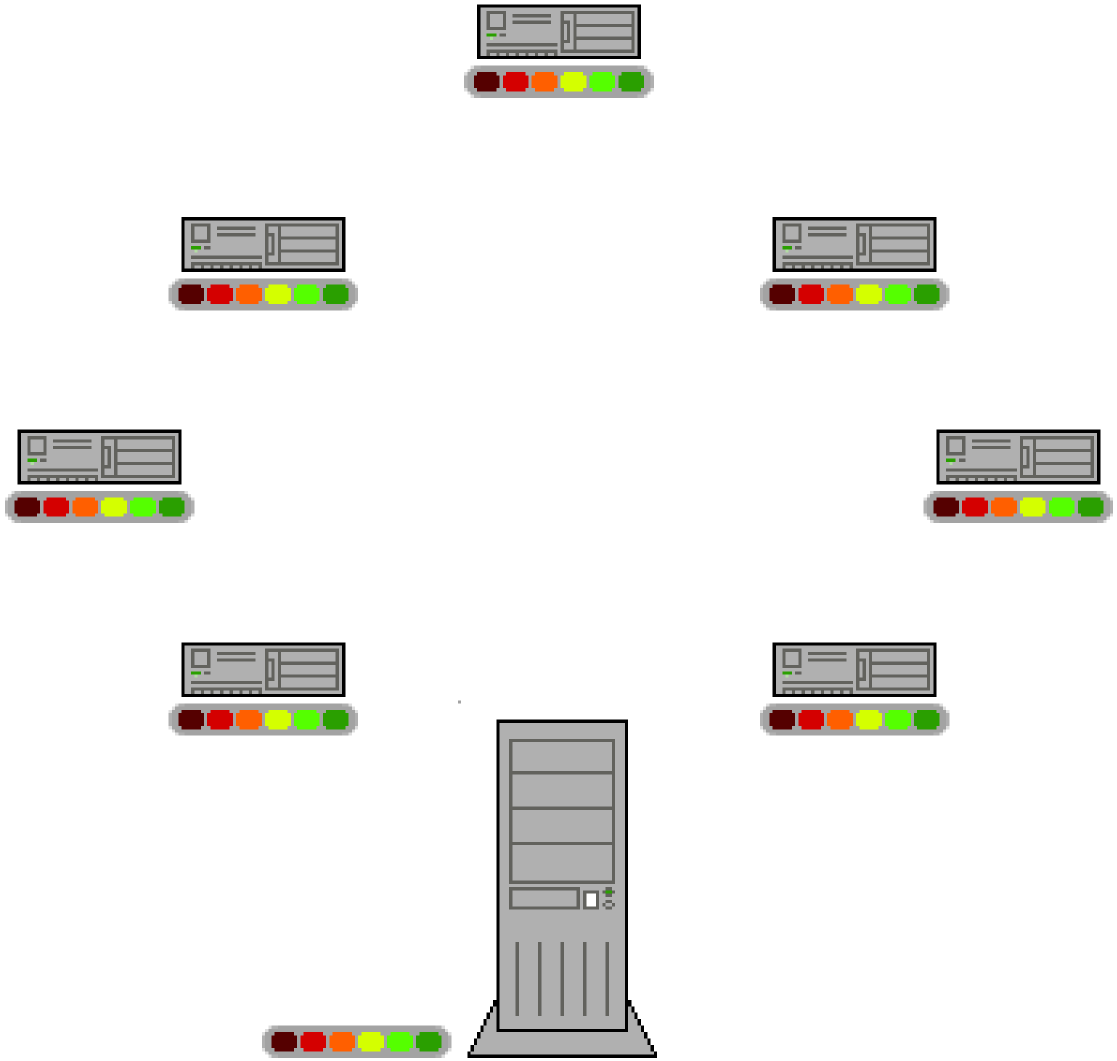












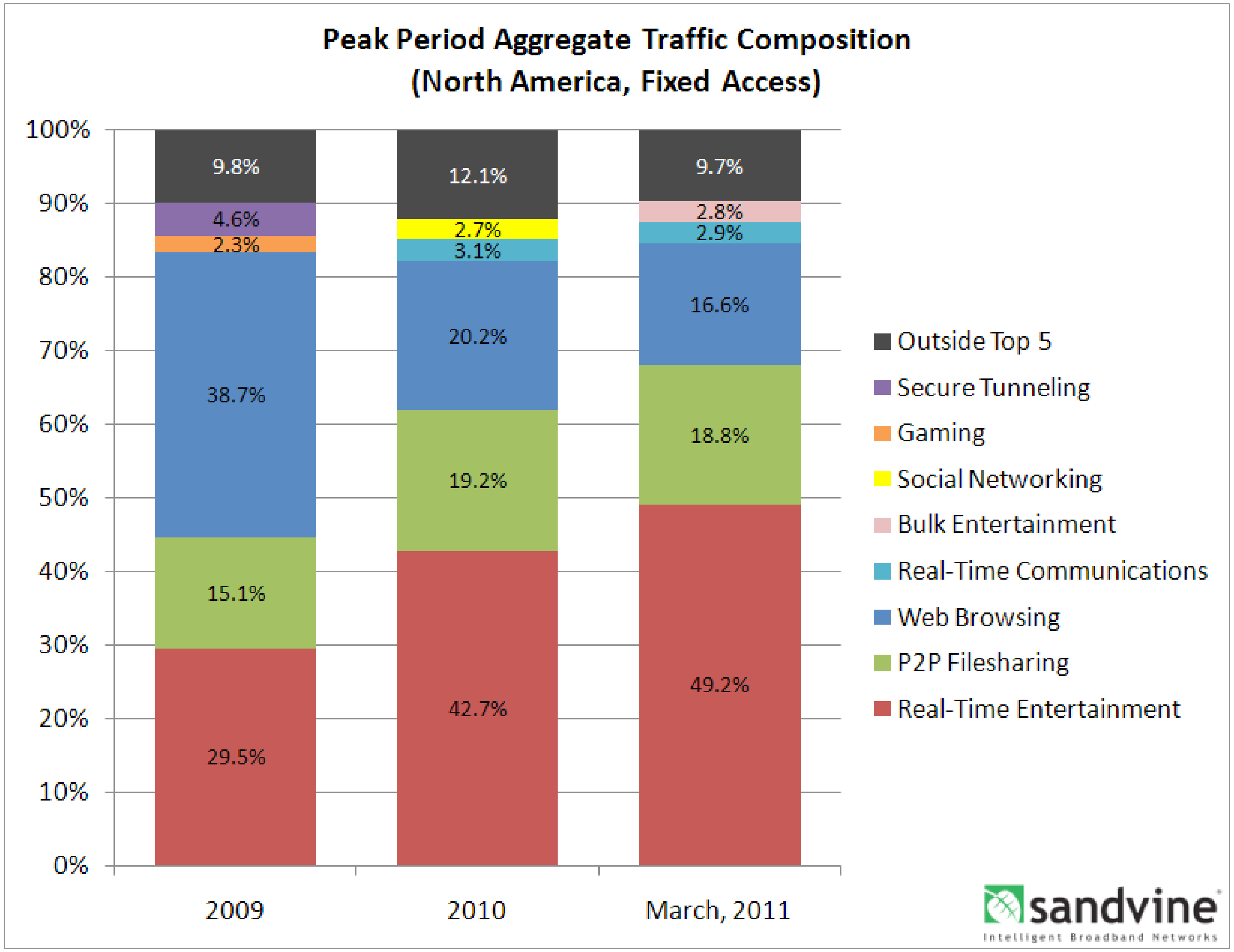
Two important aspects

- *Peer selection*
  - How peers choose other peers to exchange data with
- *Piece selection*
  - How peers choose the data to be exchanged

BitTorrent

- Successful system
  - More than 70 client implementations!
  - Mainline
    - More than 40 million downloads in 2006
  - Azureus
    - More than 70M downloads in 2009 Q1 and 160M in 2008
- Considers practical issues
  - TCP slow start
  - TCP congestion control

BitTorrent in 2011



BitTorrent in 2011

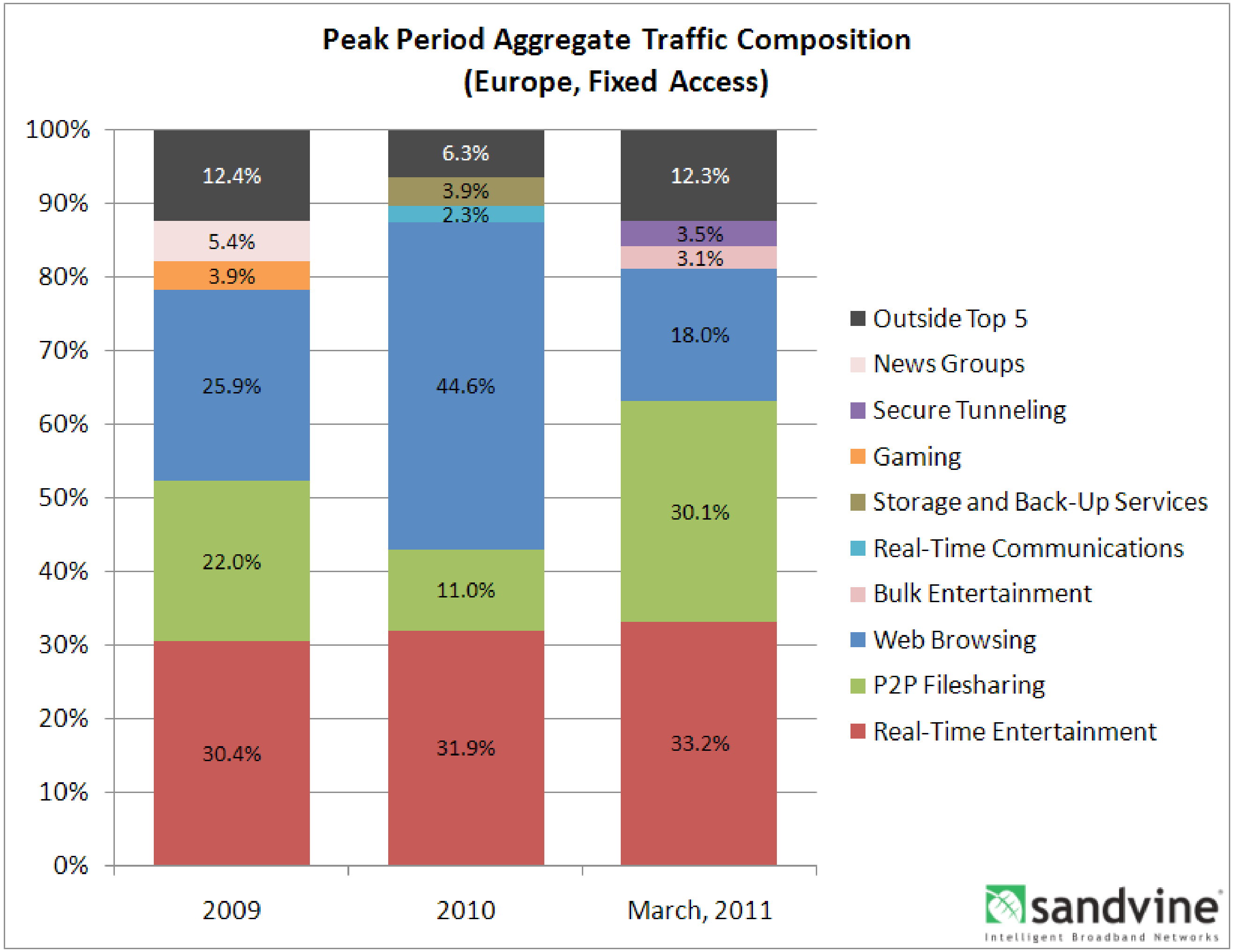
Table 1 - North America, Fixed Access, Peak Period, Top Applications by Bytes

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	52.01%	Netflix	29.70%	Netflix	24.71%
2	HTTP	8.31%	HTTP	18.36%	BitTorrent	17.23%
3	Skype	3.81%	YouTube	11.04%	HTTP	17.18%
4	Netflix	3.59%	BitTorrent	10.37%	YouTube	9.85%
5	PPStream	2.92%	Flash Video	4.88%	Flash Video	3.62%
6	MGCP	2.89%	iTunes	3.25%	iTunes	3.01%
7	RTP	2.85%	RTMP	2.92%	RTMP	2.46%
8	SSL	2.75%	Facebook	1.91%	Facebook	1.86%
9	Gnutella	2.12%	SSL	1.43%	SSL	1.68%
10	Facebook	2.00%	Hulu	1.09%	Skype	1.29%
	Top 10	83.25%	**Top 10**	84.95%	**Top 10**	82.89%

SOURCE: SANDVINE NETWORK DEMOGRAPHICS

sandvine

BitTorrent in 2011



BitTorrent in 2011

Table 6 - Europe, Fixed Access, Peak Period, Top Applications by Bytes

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	59.68%	BitTorrent	21.63%	BitTorrent	28.40%
2	Skype	7.16%	HTTP	20.47%	HTTP	18.08%
3	HTTP	7.02%	YouTube	14.13%	YouTube	11.93%
4	PPStream	3.64%	RTMP	4.58%	RTMP	3.90%
5	Spotify	2.91%	Flash Video	3.99%	Flash Video	3.38%
6	SSL	2.66%	iTunes	3.65%	SSL	3.09%
7	eDonkey	1.76%	SSL	3.18%	iTunes	3.07%
8	YouTube	1.76%	NNTP	2.73%	Skype	2.44%
9	Facebook	1.42%	Facebook	1.71%	NNTP	2.30%
10	Teredo	1.18%	Skype	1.42%	PPStream	1.77%
	Top 10	89.19%	**Top 10**	77.49%	**Top 10**	78.36%

SOURCE: SANDVINE NETWORK DEMOGRAPHICS

sandvine

BitTorrent strategy

- Fact: Total download = total upload
- Try to make the download rate proportional to the upload rate for each peer
  - Helps to avoid free riders
- Create a random graph between peers
  - Good robustness
- "The BitTorrent file distribution system uses *tit-for-tat* as a method of seeking Pareto efficiency."

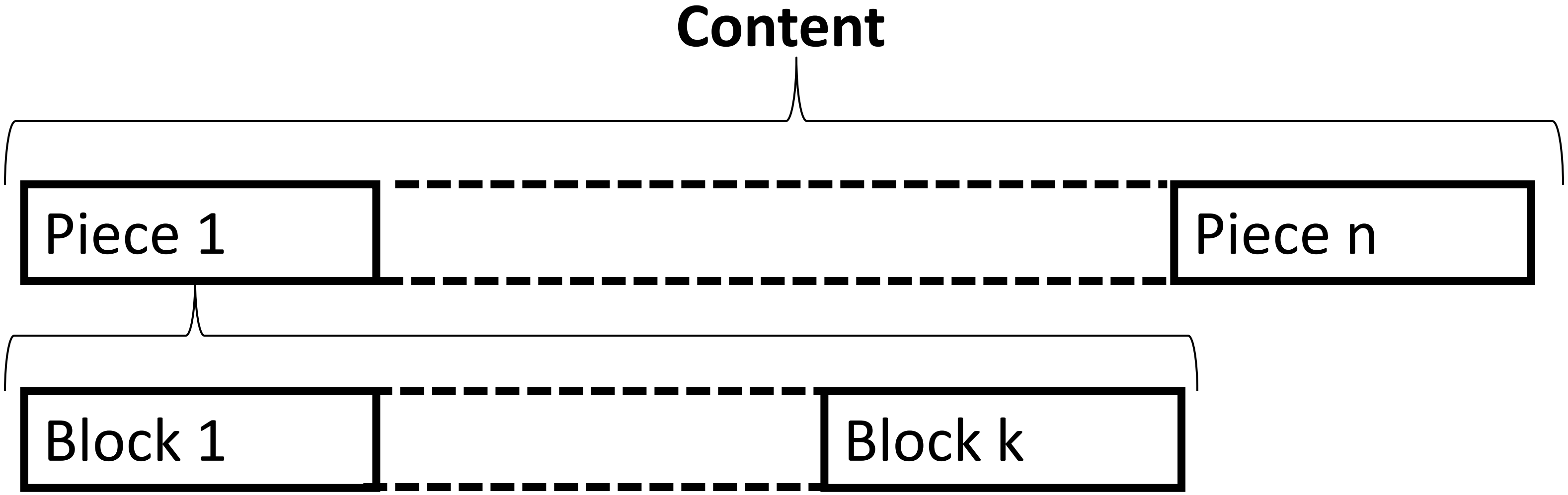
Tit for Tat

- Best deterministic strategy for the Iterated Prisoner's Dilemma
  - Unless provoked, the agent will always cooperate
  - If provoked, the agent will retaliate
  - The agent is quick to forgive
  - The agent must have a good chance of competing against the opponent more than once.

http://en.wikipedia.org/wiki/Tit_for_tat

Pieces and Blocks

- Content is split into *pieces* (256KB-2MB)
- Pieces are split into *blocks* (16KB)



BitTorrent terminology

- A peer who has all the pieces is called a *seed*
- A peer who does not have all the pieces is called a *leecher*
- A *tracker* keeps track of all peers in the swarm
- A *torrent* file contains swarm metadata:
  - Tracker address, the piece size, the # of pieces, a hash of each piece, the file(s) name and size

Publishing content

- Split content into pieces, compute hashes for each piece, and create a meta-data torrent file
- Register the torrent with a tracker
- Start the BitTorrent client acting as seed
- Publish the torrent file on a web server or using a decentralized tracker

.torrent file

- Encoded using bencoding
- Info key
  - Length on the content in bytes
  - File Name
  - Piece length
  - SHA-1 hashes for all pieces
- Announce URL of the tracker (HTTP)
- Some optional fields
  - Creation date, comment, created by

Joining a swarm

- Downloaders find the meta-data torrent file
- Retrieve from the tracker a list of peers who are already in the swarm (50 random peers)
- Tracker is centralized but it is not involved in data transfer
- The tracker only keeps track of the peers currently involved in the torrent

Neighbor peers

- Peer registers with the tracker after join and every 30 minutes sends its state to the tracker
- Each peer has a *neighbor set* of other peers
  - Initially retrieved from the tracker
  - Maximum size of the neighbor set is 80
- Peer keeps open TCP connections to the peers in its neighbor set
  - If |neighbors| < 20 ask tracker for more peers
  - Peer initiated a maximum of 40 connections
  - Rest of 40 are connection accepted from other peers

Peer-to-Peer data transfer

- Peers exchange blocks of content with neighbor peers over TCP connections
- *Pipelining:* to avoid TCP's "slow start" delay, 5 block requests are kept active at once
  - "This is the most crucial performance item"
- At all times, a peer uploads data to no more than 4 neighbor peers, its *active neighbor set*
  - "This allows TCP's built-in congestion control to reliably saturate upload capacity."

Piece information

- After establishing a connection, peers shake hands and exchange their piece *bitfields*
- After the bitfield exchange both peers know what pieces the other peer has
  - Peer A is *interested* in peer B if peer B has pieces that peer A does not have
  - Peer A is *not interested* in peer B if peer B has a subset of the pieces that peer A has
- When a peer acquires a new piece it tells all its neighbors by sending them a HAVE message

Peer connections

- To avoid the cost of handshaking and bitfield exchange, peers keep the connections open
- Keep-alive messages are sent every 2 minutes
- A neighbor peer is either *choked* or *unchoked*
  - **am_choking**: this client is choking the peer
  - **am_interested**: this client is interested in the peer
  - **peer_choking**: peer is choking this client
  - **peer_interested**: peer is interested in this client

Peer (un)choking

- Unchoked peers form the *active neighbor set*
- The *active* neighbor set is updated periodically and determined by the *choke algorithm*
- The choke algorithm selects the neighbors to which the local peer uploads (*peer selection*)
- Two versions
  - Leecher choke algorithm
  - Seeder choke algorithm

Leecher Choke Algorithm

- Runs periodically every 10 seconds
- Also runs when a peer leaves the neighbor set or when an unchoked peer becomes interested or not interested
- We call each run of the algorithm a *round*
- Step 1: every 3 rounds a random neighbor that is **choked** and **interested** is selected as the *planned optimistic unchoked peer* (POU)

Leecher Choke Algorithm

- Step 2: Sort all interested peers that have uploaded at least 1 block in the last 30s, by their *current* upload rate to the local peer
  - Exclude *snubbed* peers, the ones who didn’t upload anything in the last 30 seconds
  - The current upload rate of the peer is computed a rolling average over the last 20 seconds
- Step 3: The three fastest peers are unchoked
  - We call these the regular unchoked (RU) peers

Leecher Choke Algorithm

- Step 4: If the POU peer is *not* one of the RU peers, it is unchoked and the round completes
- Step 5: Else, another peer is chosen at random to be the POU peer
  - 5a: If this POU peer is interested, it is unchoked and the round completes
  - 5b: Else, the POU peer is unchoked and a new POU peer is selected at random. Step 5a is repeated with the new POU peer

Leecher Choke Algorithm

- In one round 4 interested peers are unchoked
- More than 4 peers (*uninterested*) are unchoked
- As soon as one of these unchoked peers becomes interested, a new round runs
- *Optimistic unchoking* (steps 4 and 5a)
  - Finds potentially faster peers
  - Allows new peers with no pieces to *bootstrap*, by giving them their first piece

Seeder Choke Algorithm

- Old version similar to the leecher version but sorting peers (step 2) by their download rate
  - Problematic since high download leechers can monopolize seeds
- New version
  - Runs periodically every 10 seconds
  - Also runs also when a peer leaves the neighbor set, and when an unchoked peer becomes interested or not interested
  - We call each run of the algorithm a *round*

New Seeder Choke Algorithm

- Step 1: All interested peers that were unchoked in the last 20 seconds or that have pending block requests are sorted by the *time they were last unchoked* (most recent first)
- On a tie, priority is given to the peers with the highest download rate (from this peer)
- Step 2: All other peers are sorted by their download rate (from this peer) and concatenated to the sorted peer list from step 1

New Seeder Choke Algorithm

- Step 3: during 2/3 rounds the first three peers are kept unchoked and one other random interested peer is also unchoked
- Every third round, the first four peers are kept unchoked
- As a consequence of step 1 the peers in the active neighbor set are rotated frequently
- A seed thus uniformly divides its upload capacity to all its peers

Anti-snubbing

- When over a minute has gone by without receiving a single sub-piece from a particular peer, do not upload to it except as an optimistic unchoke
- A peer is said to be *snubbed* if all its peers choke it
- To handle this, a snubbed peer stops uploading to its peers
- Download will lag until optimistic unchoke finds better peers
- Increase the number of optimistic unchokes
  - Hope that will discover a new peer that will upload to us

Piece selection strategies

- Strict Priority
  - Other blocks from same source
- Rarest First
  - Common parts left for later
- Random First Piece
  - Start-up need to get a complete piece
- Endgame Mode
  - Broadcast for all remaining blocks

Strict priority

- Once a block has been requested from a piece, the remaining blocks of the same piece are requested with highest priority
- Get complete pieces as soon as possible
- Important to minimize the number of partially received pieces, since only complete pieces can be uploaded to other peers

Rarest-first

- A peer knows what pieces its neighbors have
- Can compute *local availability* for each piece
  - How many times the piece is available on the peers in the neighbor set
- Assume the minimum local availability among all pieces is *m*
  - The *rarest-pieces set* is the set of all pieces with local availability *m*
  - The rarest-pieces set is updated every time the peer receives a HAVE or a BITFIELD message

Rarest-first

- A *random* piece is selected from the rarest-pieces set
  - Randomization avoid many peers in the same neighborhood crowding on the same piece
- Rarest-first aims to maximize the entropy of the pieces in the torrent
  - Peers get the pieces that their neighbors will need
  - Different pieces are downloaded from seeds
  - Prolongs the life a torrent by reducing the risk that a piece becomes extinct

Random first-piece

- Used in the beginning of the download, before having received 4 complete pieces
- Pieces are selected at random and different blocks can be requested from different peers
- Get complete pieces as soon as possible
- Important to have some pieces to reciprocate for the choke algorithm.

End-game mode

- Piece selection strategy adopter at the very end of the download
  - once all remaining blocks were requested
- All remaining blocks are requested from all peers in the neighbor set
- Once a block is received, a CANCEL message is sent to all peers

Study results

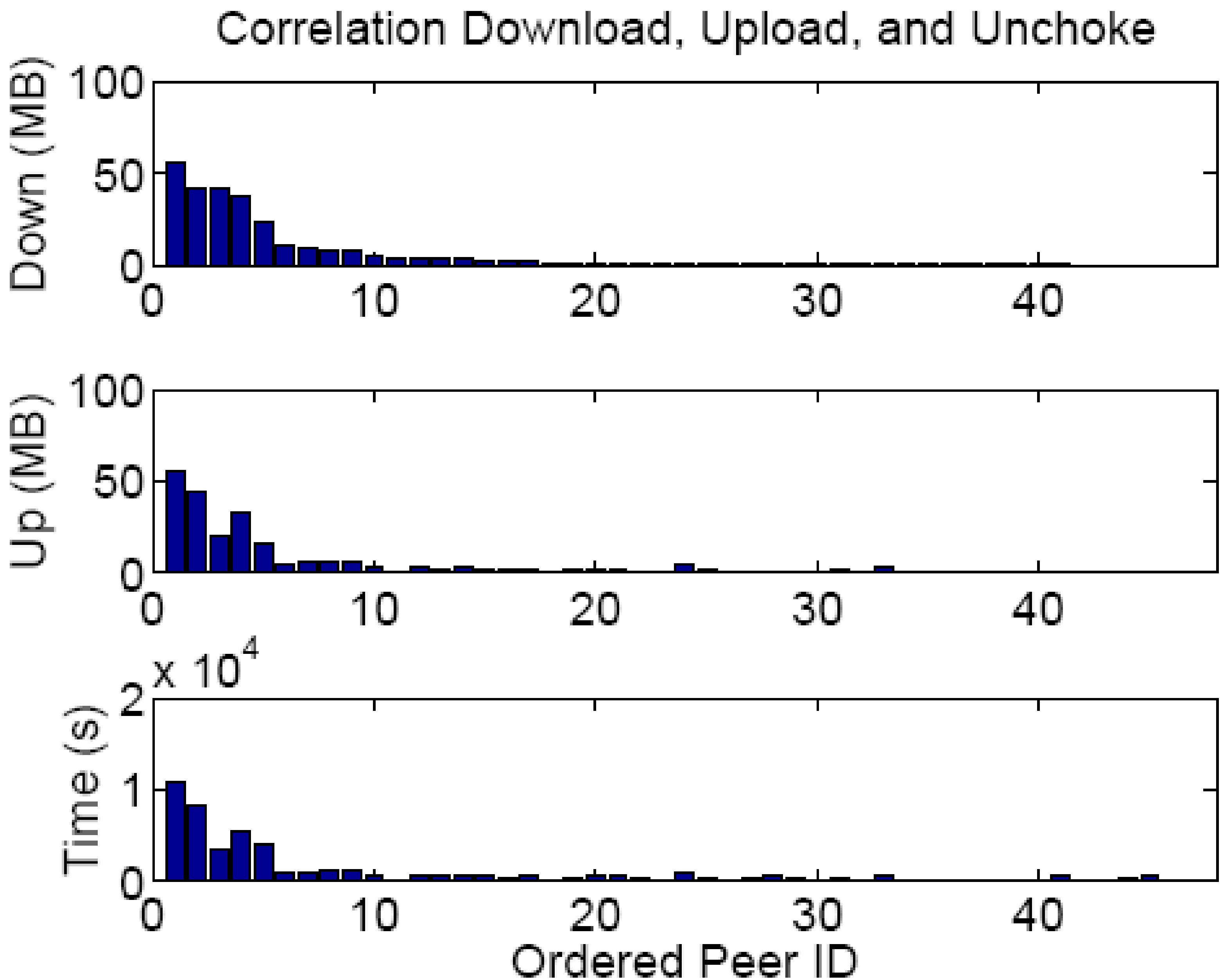
- Very low protocol overhead (< 2%)
- Choke algorithm
  - gives a fair chance to each peer to be served by a given peer
  - achieves a reasonable reciprocation with respect to the amount of data exchanged between leechers
  - Seeder algorithm evenly shares the capacity offered by a seed among all candidate leechers

[Legout *et al.*, INRIA-TR-2006]

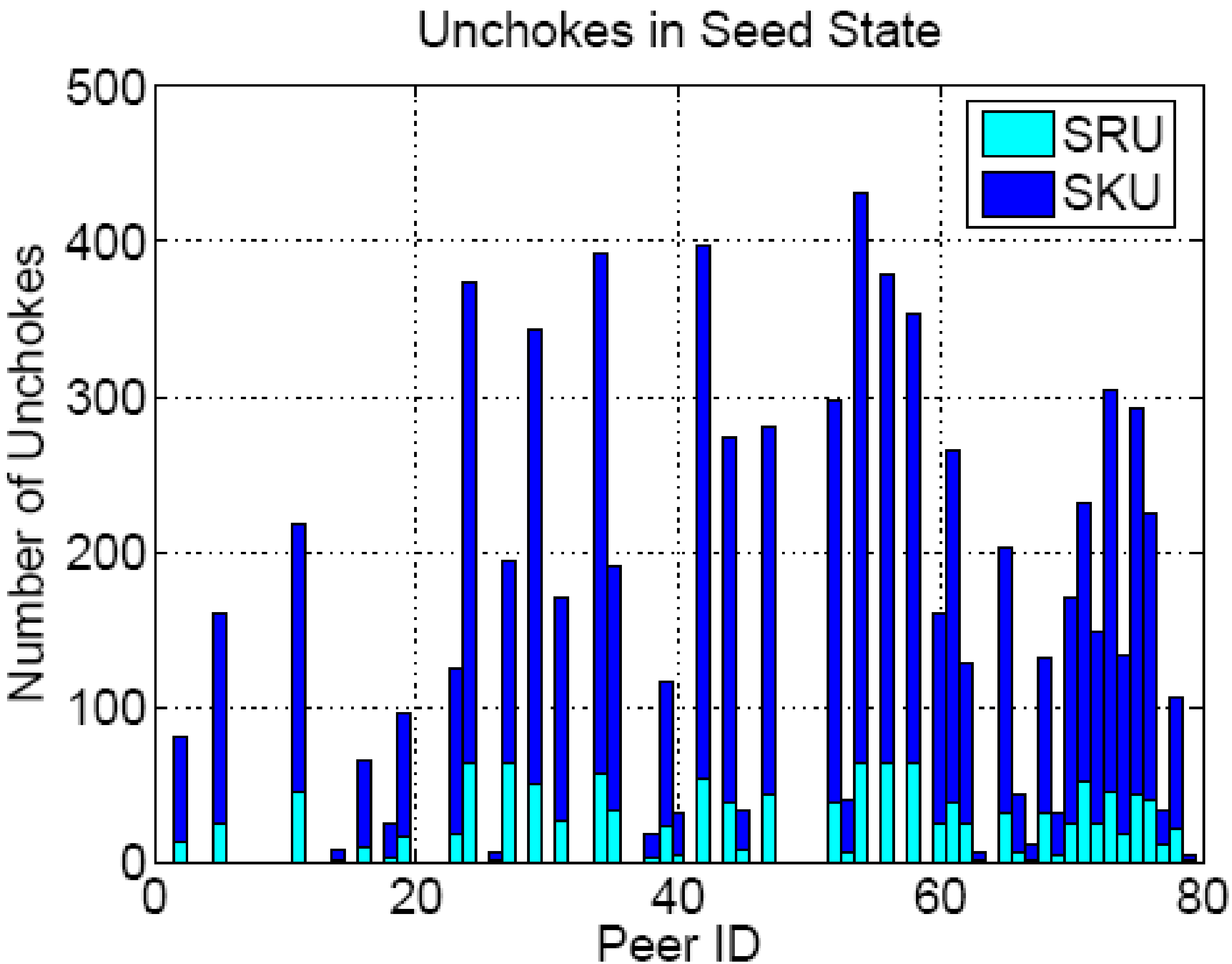
Study results

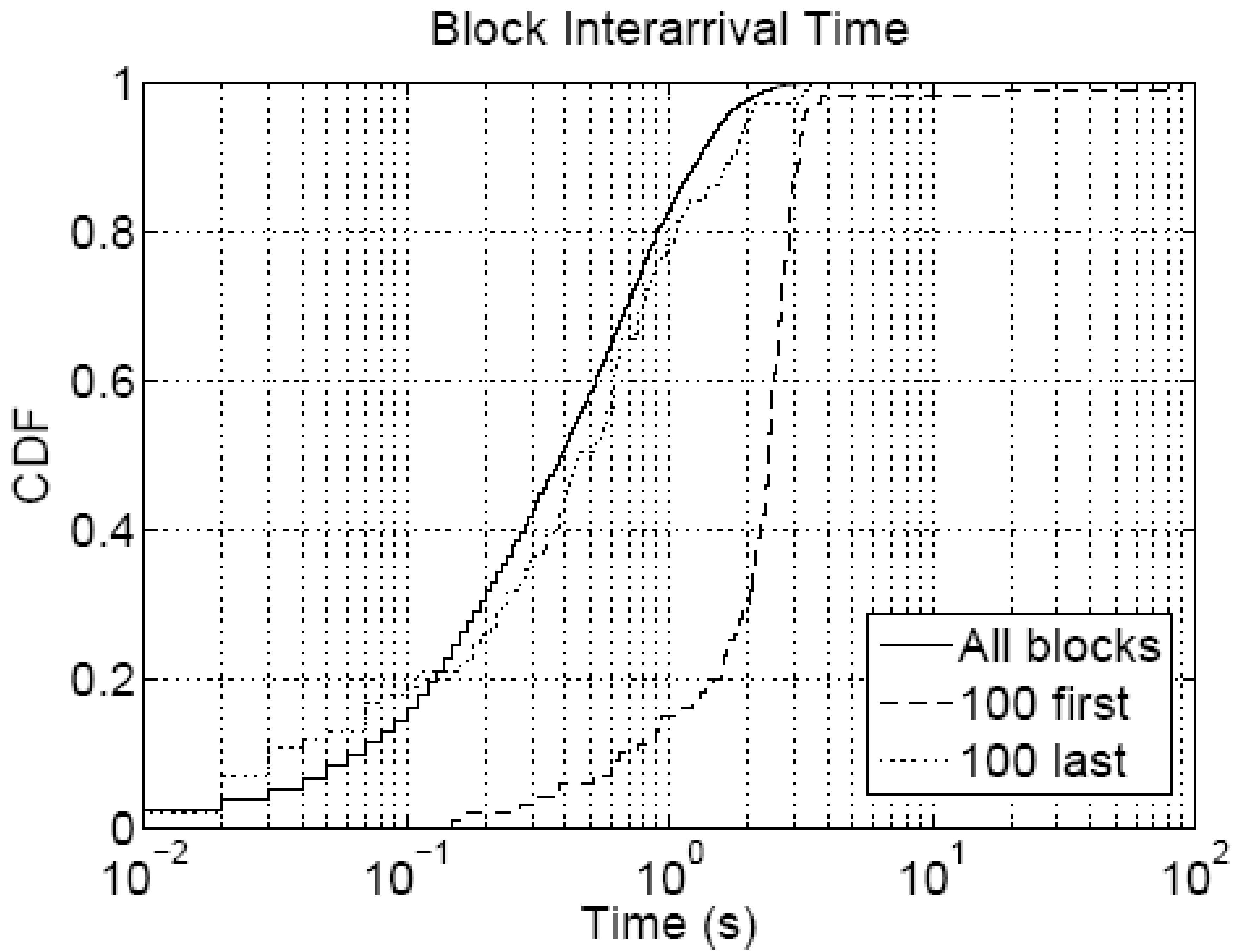
- Rarest-first piece selection strategy consistently increases with time the diversity (entropy) of the pieces in the peer set
- The last pieces problem is overstated whereas the first pieces problem is underestimated

[Legout *et al.*, INRIA-TR-2006]

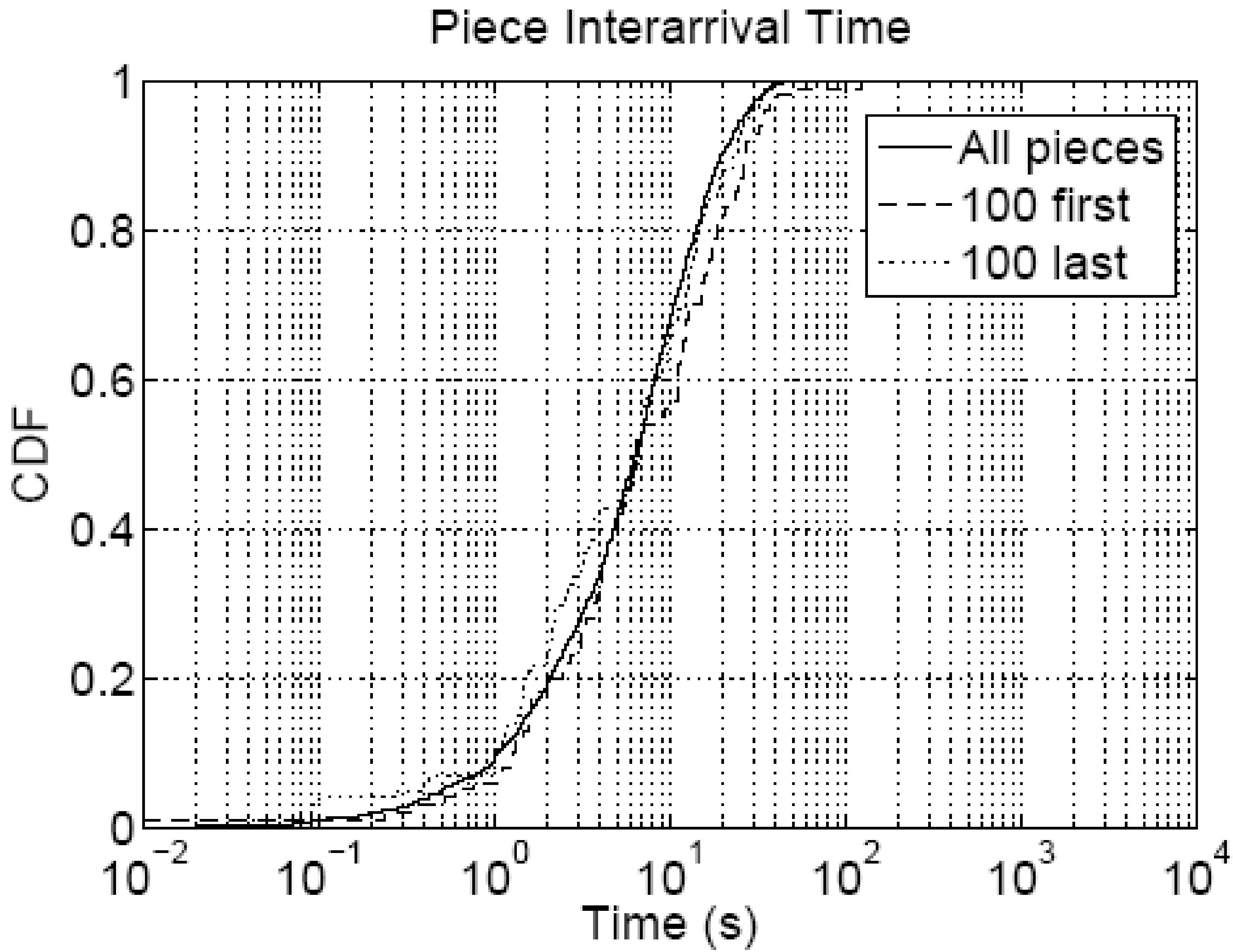


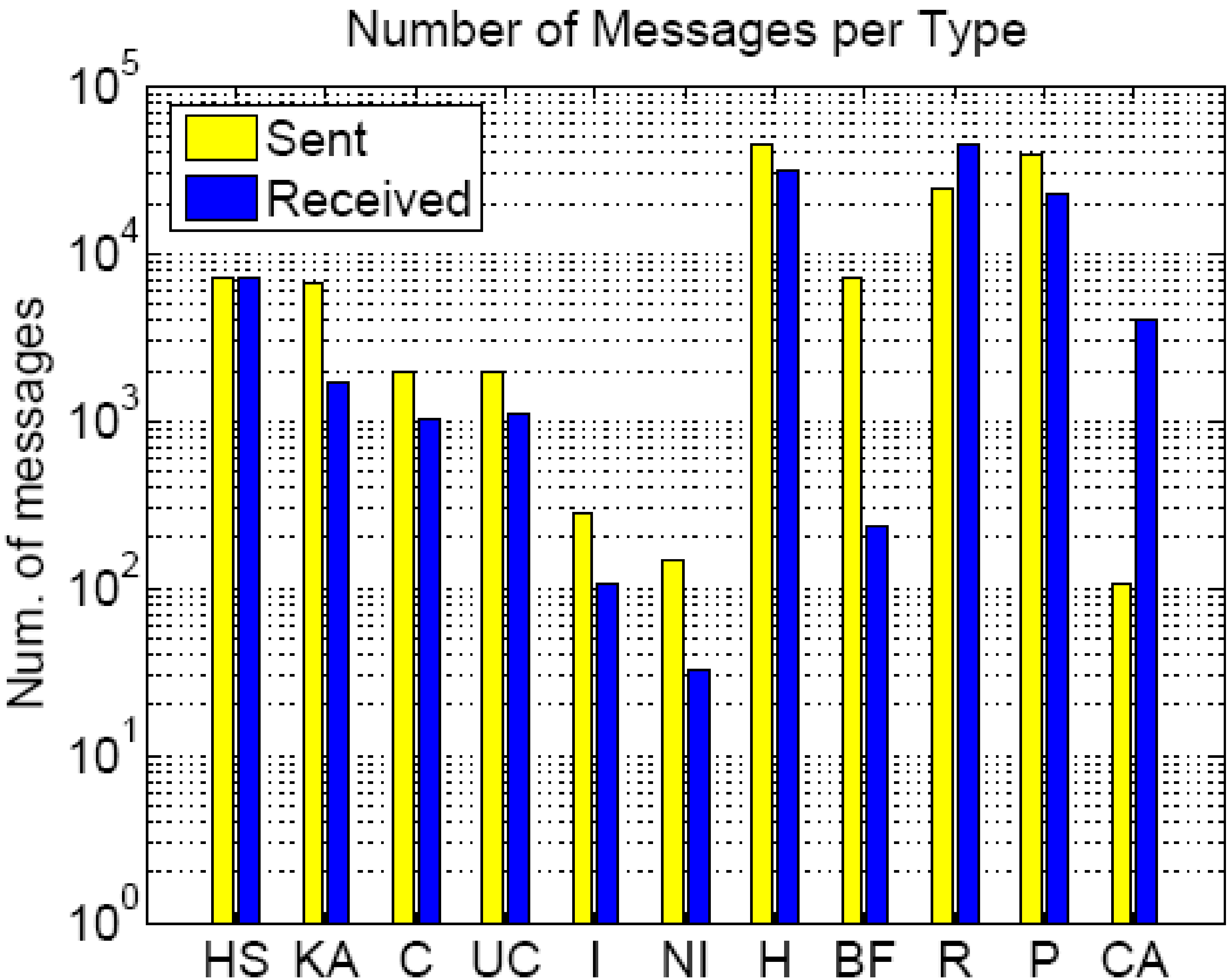
Correlation Download, Upload, and Unchoke
Down (MB)
100
50
0
0
10
20
30
40
Up (MB)
100
50
0
0
10
20
30
40
x 10^4
Time (s)
2
1
0
0
10
20
30
40
Ordered Peer ID


Unchokes in Seed State
SRU
SKU
Number of Unchokes
Peer ID
0
100
200
300
400
500
0
20
40
60
80

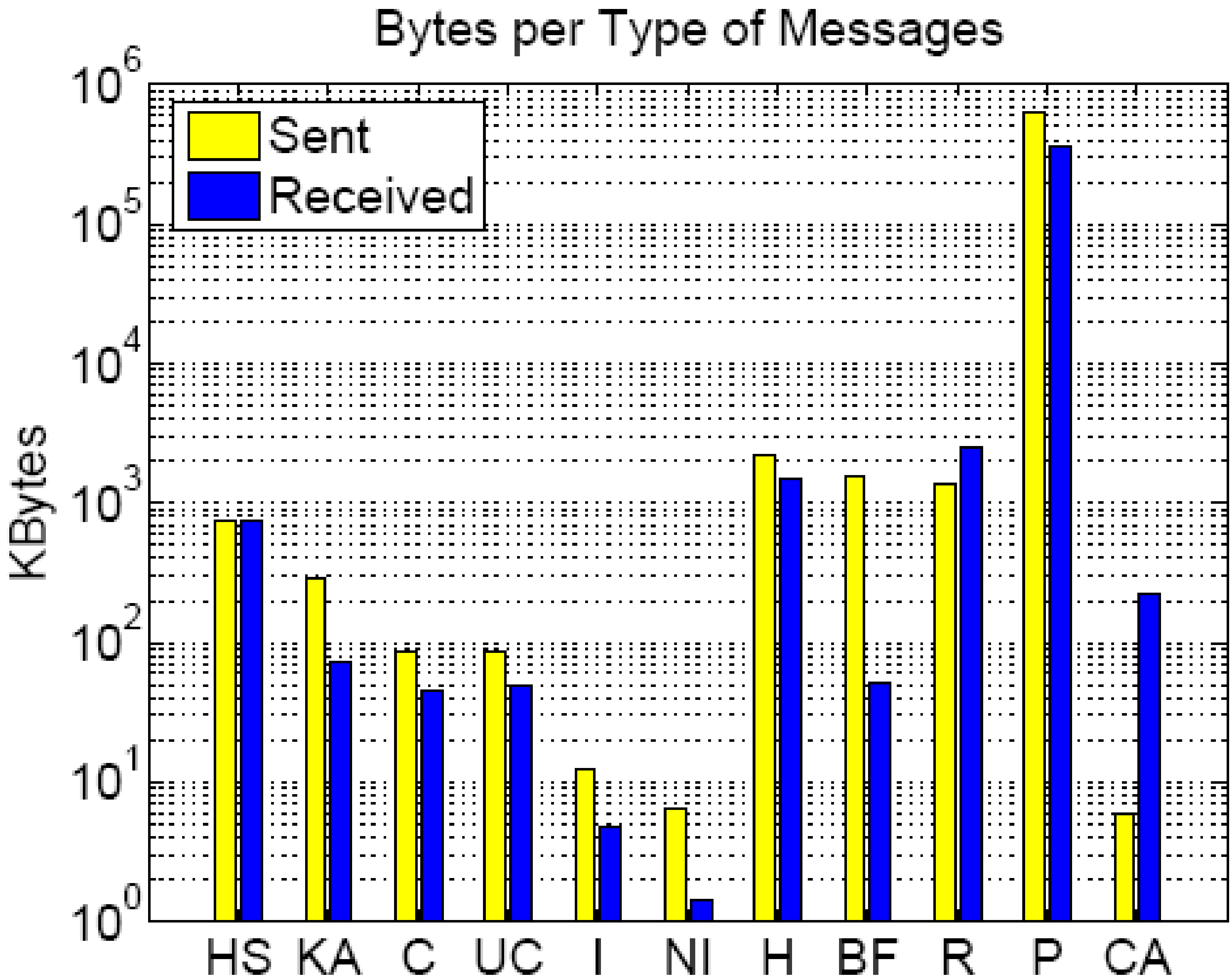


Block Interarrival Time
1
0.8
0.6
0.4
0.2
0
CDF
10^-2
10^-1
10^0
10^1
10^2
Time (s)
All blocks
100 first
100 last


Piece Interarrival Time
CDF
Time (s)
All pieces
100 first
100 last
0
0.2
0.4
0.6
0.8
1
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}



Number of Messages per Type
Sent
Received
Num. of messages
10^5
10^4
10^3
10^2
10^1
10^0
HS
KA
C
UC
I
NI
H
BF
R
P
CA



Bytes per Type of Messages
KBytes
10^0
10^1
10^2
10^3
10^4
10^5
10^6
Sent
Received
HS
KA
C
UC
I
NI
H
BF
R
P
CA

BitTorrent Extensions

- Distributed tracker
- Peer-exchange
- Multiple trackers

Summary of issues

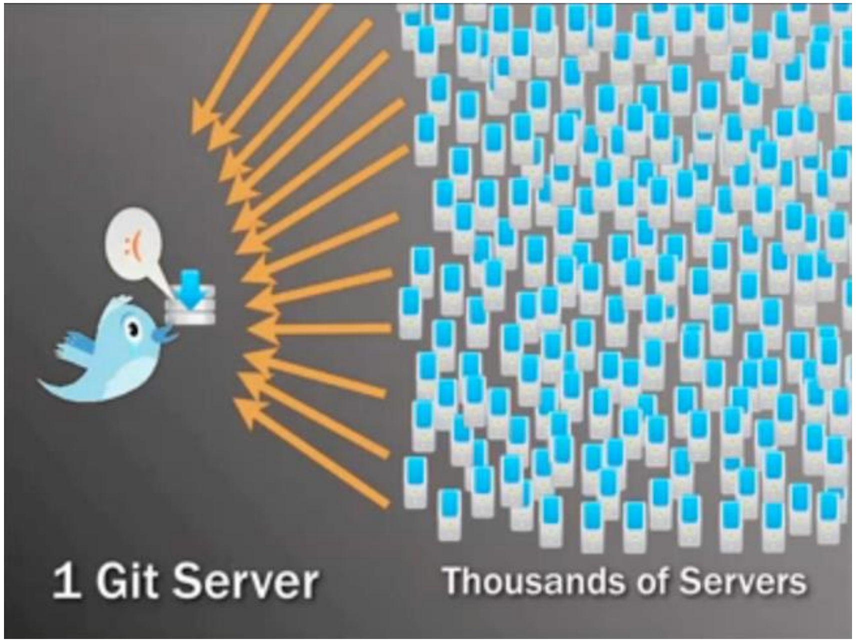
- Peer discovery
  - Central tracker, distributed tracker, peer-exchange
- Data discovery
  - Exchanged by peers
- Peer selection
  - Choke algorithms
- Piece selection
  - Rarest-first

Applications of BitTorrent

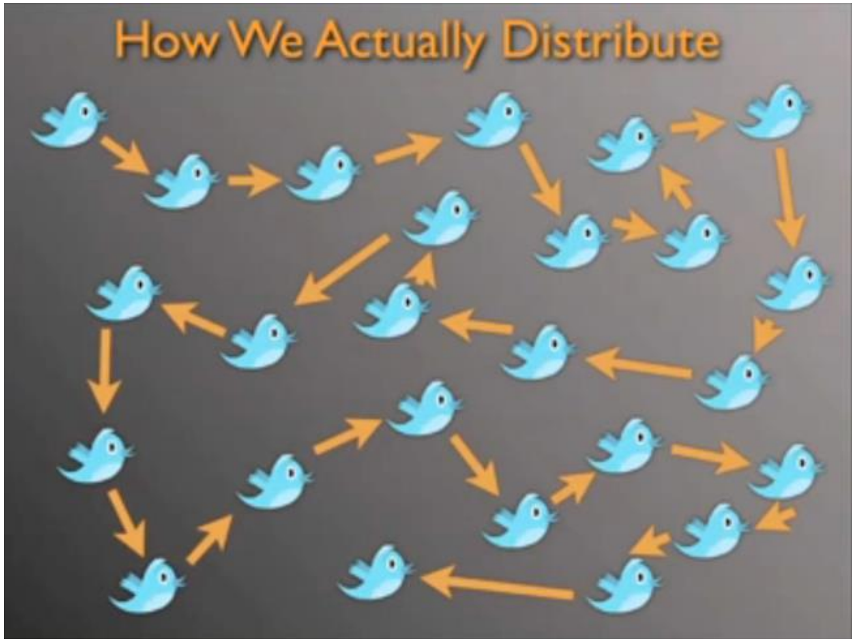
- A BitTorrent-based file transfer protocol
- Twitter uses Murder to update the software running on Twitter servers
  - 75x faster
  - http://engineering.twitter.com/2010/07/murder-fast-datacenter-code-deploys.html

Murder

Centralized software updates using Git

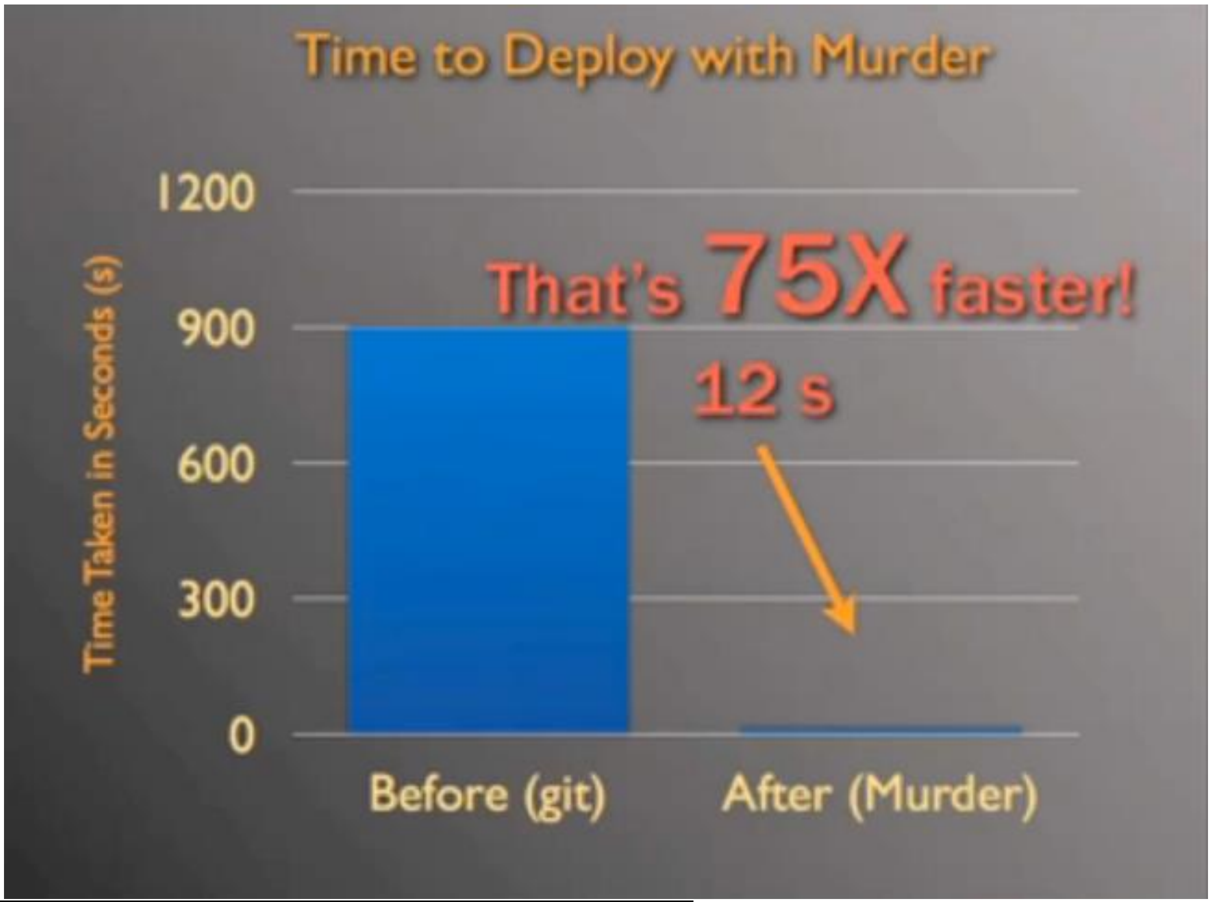


Decentralized software updates using Murder



Credit: Larry Gadea

Murder Performance



Credit: Larry Gadea

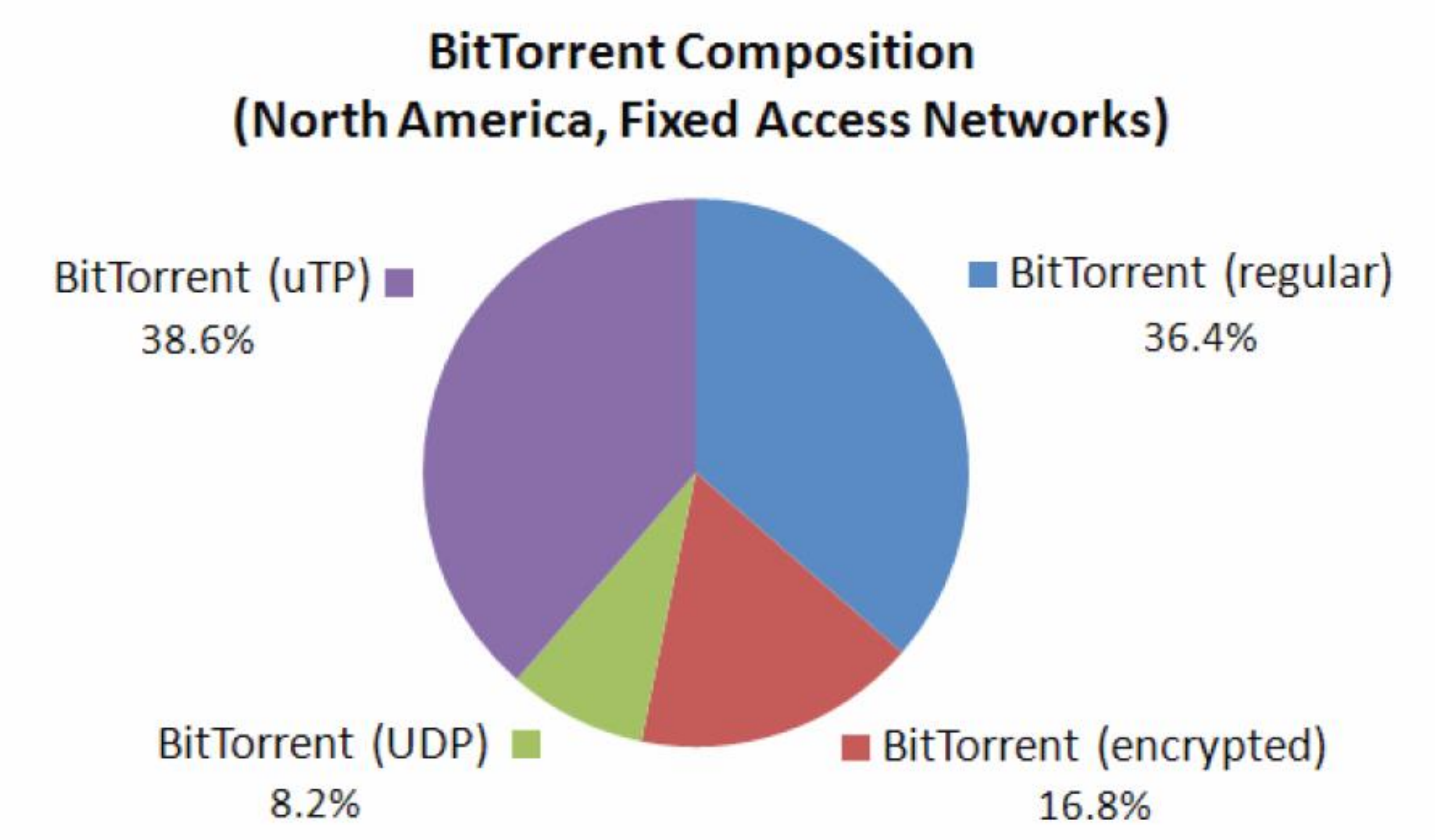
Applications of BitTorrent

- P2P Video-on-Demand
  - P2P-Next used by Wikipedia is based on a modified BitTorrent called Swift.
    - http://www.libswift.org/
  - Problems:
    - Piece sizes of 512KB are too large, resulting in delays in downloading the first pieces for playback.
      - However, decreasing pieces sizes linearly increases the amount of advertising overhead in BitTorrent...
    - In-order piece selection instead of rarest-piece selection
      - What are the implications for the overlay topology?

Future of BitTorrent

- Move from TCP to UDP
  - Reliable and in-order delivery not critical
  - TCP has a high per-connection memory footprint
    - Prevents large numbers of connections to peers
  - TCP is very poor at NAT traversal
  - Congestion control in TCP means that your OS treats BitTorrent's TCP connections as equally as important as your Browser or Email client's single TCP connection
- uTorrent has moved from TCP to Ledbat/UDP

TCP and uTP usage



Credit: sandvine 2011

Reducing Inter-ISP Traffic

- ISPs have high costs for P2P traffic
  - BitTorrent does not take into account the cost of sending packets to peers in different ISPs
  - ISPs have resorted to blocking and shaping P2P traffic
- Most ISPs are stub Autonomous Systems (AS) with a Transit AS link and maybe some peering AS links
  - Would like to bias BitTorrent traffic to reduce the amount sent over costly transit AS links.
  - Trade-off with user experience, as this may increase download times.

References

- Basic BitTorrent mechanisms [Cohen, P2PECON'03]
- BitTorrent specification Wiki http://wiki.theory.org/BitTorrentSpecification
- Measurement studies
  [Izal *et al.,* PAM'04],
  [Pouwelse *et al*., Delft TR 2004 and IPTPS'05],
  [Guo *et al.,* IMC'05], and
  [Legout *et al.*, INRIA-TR-2006]

References

- Theoretical analysis and modeling [Qiu *et al.*, SIGCOMM'04], and [Tian *et al.*, Infocom'06]
- Simulations [Bharambe *et al.*, MSR-TR-2005]
- Incentives and exploiting them [Shneidman *et al.*, PINS'04], [Jun *et al.*, P2PECON'05], and [Liogkas *et al.*, IPTPS'06]
- Sandvine. "Global Internet Phenomena Report", Spring 2011.