Peer-to-peer live streaming

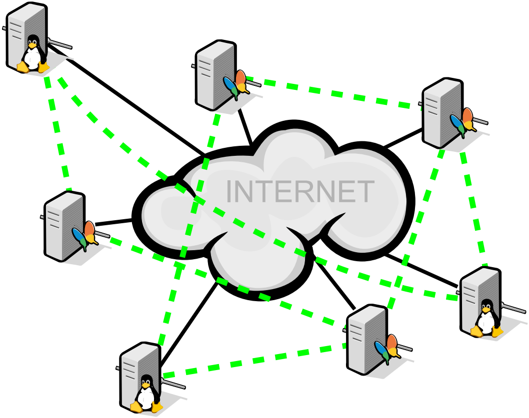
Remigiusz Jan Andrzej Modrzejewski

MASCOTTE Project
I3S(CNRS/UNS)–INRIA

April 12, 2011

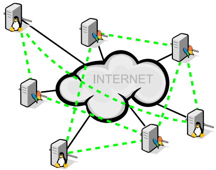
Outline

Introduction: P2P



Peer to peer networks — end systems creating a virtual overlay

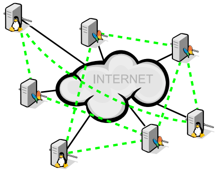
Introduction: Video distribution



File sharing

Live streaming

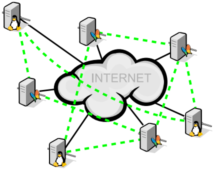
Introduction: Video distribution



File sharing

Live streaming

Introduction: Video distribution



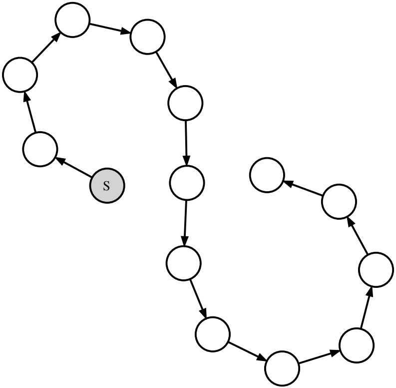
File sharing

Live streaming

Problem definition

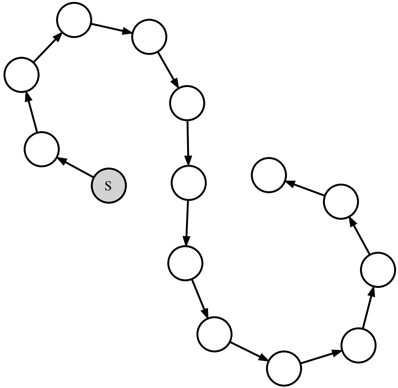
- Disseminate a stream of data
- Single source
- Multiple recipients
- Recipients contribute to further disseminate

Overlay with 1x bandwidth



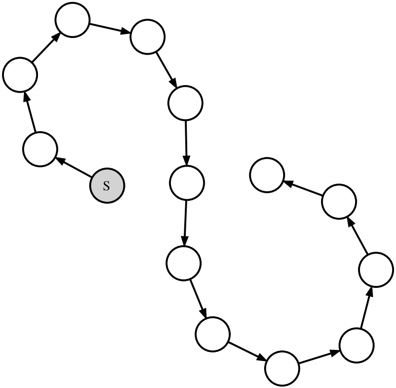
- Bandwidth efficient
  - Lower bound for feasibility
  - In real world clients have just enough bandwidth
- Simple construction algorithm
- Easy to build reliability
- Linear delay

Overlay with 1x bandwidth



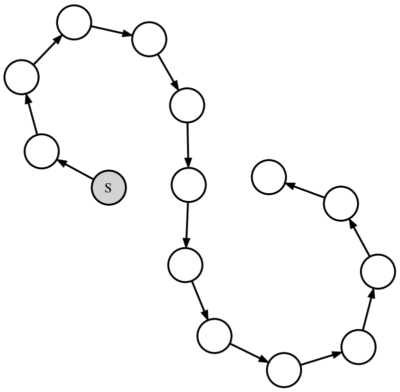
- Bandwidth efficient
  - Lower bound for feasibility
  - In real world clients have just enough bandwidth
- Simple construction algorithm
- Easy to build reliability
  - Without it single failure kills half of overlay
  - Still recovery very simple
- Linear delay

Overlay with 1x bandwidth



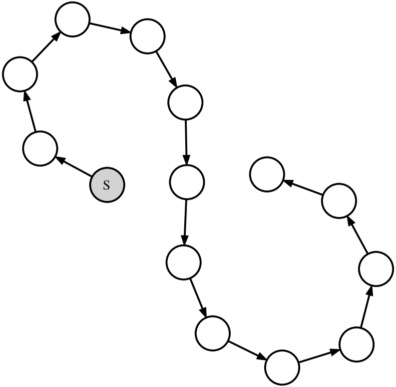
- Bandwidth efficient
  - Lower bound for feasibility
  - In real world clients have just enough bandwidth
- Simple construction algorithm
- Easy to build reliability
- Linear delay

Overlay with 1x bandwidth



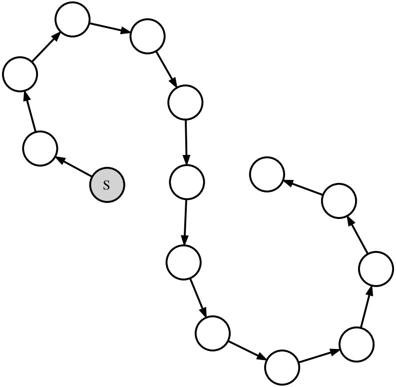
- Bandwidth efficient
  - Lower bound for feasibility
  - In real world clients have just enough bandwidth
- Simple construction algorithm
- Easy to build reliability
  - Without it single failure kills half of overlay
  - Still recovery very simple
- Linear delay

Overlay with 1x bandwidth



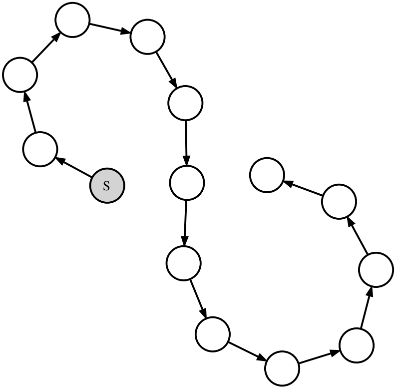
- Bandwidth efficient
  - Lower bound for feasibility
  - In real world clients have just enough bandwidth
- Simple construction algorithm
- Easy to build reliability
  - Without it single failure kills half of overlay
  - Still recovery very simple
- Linear delay

Overlay with 1x bandwidth



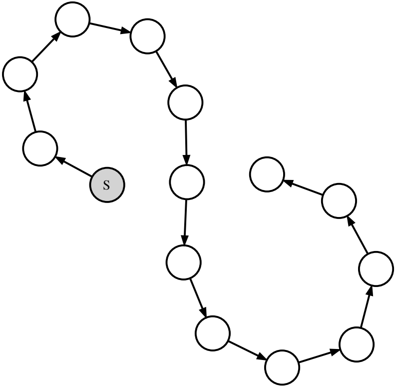
- Bandwidth efficient
  - Lower bound for feasibility
  - In real world clients have just enough bandwidth
- Simple construction algorithm
- Easy to build reliability
  - Without it single failure kills half of overlay
  - Still recovery very simple
- Linear delay

Overlay with 1x bandwidth



- Bandwidth efficient
  - Lower bound for feasibility
  - In real world clients have just enough bandwidth
- Simple construction algorithm
- Easy to build reliability
  - Without it single failure kills half of overlay
  - Still recovery very simple
- Linear delay

Overlay with 1x bandwidth



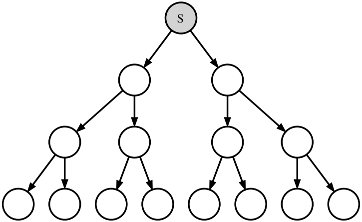
- Bandwidth efficient
  - Lower bound for feasibility
  - In real world clients have just enough bandwidth
- Simple construction algorithm
- Easy to build reliability
  - Without it single failure kills half of overlay
  - Still recovery very simple
- Linear delay

Problem definition 2

- Disseminate a stream of data
- Single source
- Multiple recipients
- Recipients contribute to further disseminate
- Finite dissemination deadline

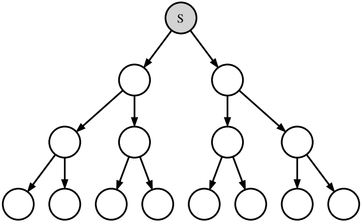
Overlay with 2x bandwidth

- Logarithmic delay
- Still simple to construct
    - $O(1)$ time and $O(n)$ memory
    - $O(\log n)$ time and $O(1)$ memory in each node
- Hard to ensure reliability
    - Failure brings down $\log_2 n/2$ peers on average
    - Costly rebalance
- Loses half of bandwidth



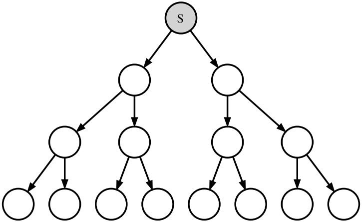
Overlay with 2x bandwidth

- Logarithmic delay
- Still simple to construct
  - $O(1)$ time and $O(n)$ memory
  - $O(\log n)$ time and $O(1)$ memory in each node
- Hard to ensure reliability
  - Failure brings down only $\frac{\log_2 n}{2}$ peers on average
  - Costly rebalance
- Loses half of bandwidth



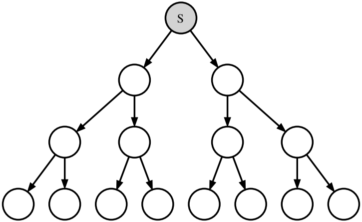
Overlay with 2x bandwidth

- Logarithmic delay
- Still simple to construct
  - $O(1)$ time and $O(n)$ memory
  - $O(\log n)$ time and $O(1)$ memory in each node
- Hard to ensure reliability
  - Failure brings down only $\frac{\log n}{2}$ peers on average
  - Costly rebalance
- Loses half of bandwidth



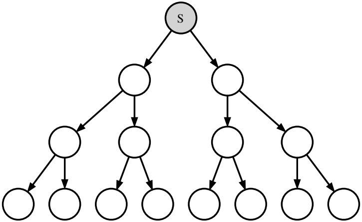
Overlay with 2x bandwidth

- Logarithmic delay
- Still simple to construct
    - $O(1)$ time and $O(n)$ memory
    - $O(\log n)$ time and $O(1)$ memory in each node
- Hard to ensure reliability
    - Failure brings down only $\frac{\log_2 n}{2}$ peers on average
- Loses half of bandwidth



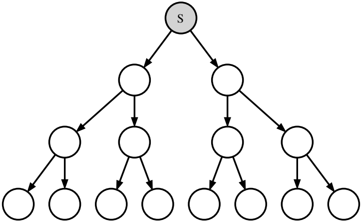
Overlay with 2x bandwidth

- Logarithmic delay
- Still simple to construct
  - $O(1)$ time and $O(n)$ memory
  - $O(\log n)$ time and $O(1)$ memory in each node
- Hard to ensure reliability
  - Failure brings down only $\frac{\log_2 n}{2}$ peers on average
  - Costly rebalance
- Loses half of bandwidth



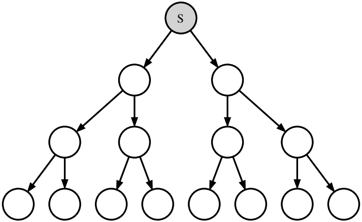
Overlay with 2x bandwidth

- Logarithmic delay
- Still simple to construct
  - $O(1)$ time and $O(n)$ memory
  - $O(\log n)$ time and $O(1)$ memory in each node
- Hard to ensure reliability
  - Failure brings down only $\frac{\log_2 n}{2}$ peers on average
  - Costly rebalance
- Loses half of bandwidth



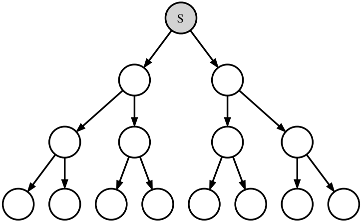
Overlay with 2x bandwidth

- Logarithmic delay
- Still simple to construct
  - $O(1)$ time and $O(n)$ memory
  - $O(\log n)$ time and $O(1)$ memory in each node
- Hard to ensure reliability
  - Failure brings down only $\frac{\log_2 n}{2}$ peers on average
  - Costly rebalance
- Loses half of bandwidth



Overlay with 2x bandwidth

- Logarithmic delay
- Still simple to construct
  - $O(1)$ time and $O(n)$ memory
  - $O(\log n)$ time and $O(1)$ memory in each node
- Hard to ensure reliability
  - Failure brings down only $\frac{\log_2 n}{2}$ peers on average
  - Costly rebalance
- Loses half of bandwidth



Problem definition 3

- Disseminate a stream of data
- Single source
- Multiple recipients
- Recipients contribute to further disseminate
- Finite dissemination deadline
- High bandwidth utilization

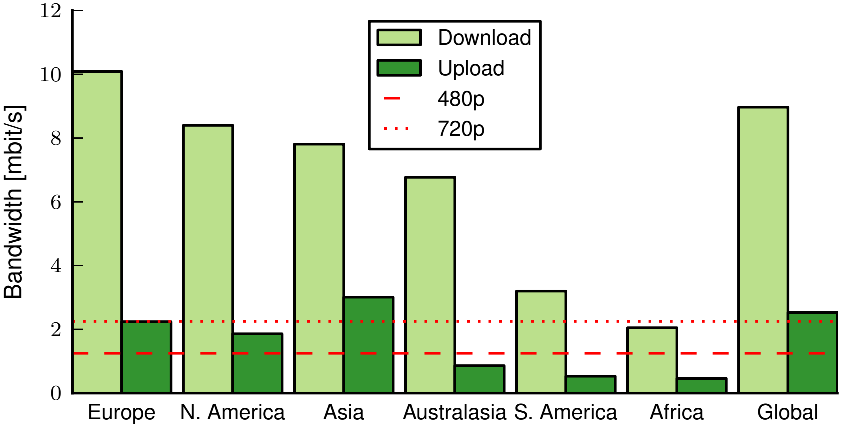
Video bit rates

Format name	Resolution	Approximate bit rate target
360p	480×360	768kbit/sec
480p	640×480	768kbit/sec
480p	854×480	1.25mbit/sec
720p	1280×720	2.25mbit/sec
1080p	1920×1080	3.75mbit/sec

Approximate bit rates in various resolutions, served by the most popular online provider — *YouTube*

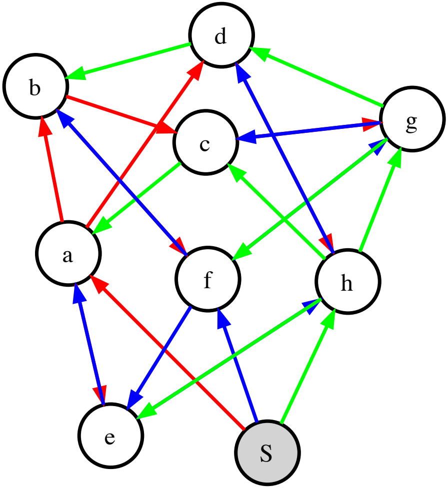
Source: *Approximate youtube bitrates*, McFarland, 2010

Available bandwidth



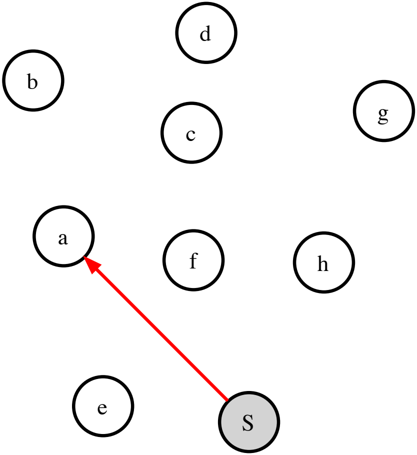
Average client bandwidth in February 2011 broken down by continent, measured using *Speedtest.net*, with marked bit rates required for 480p and 720p video

Overlay with 1x bandwidth



Both bandwidth efficient and $O(\log n)$ delay

Overlay with 1x bandwidth

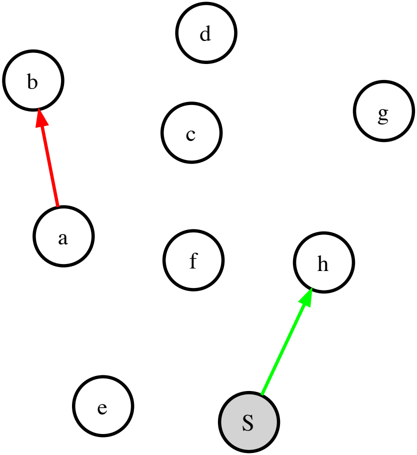


Step by step:

node	1	2	3	4	5
a	1				
b					
c					
d					
e					
f					
g					
h					

Indicates chunk currently replicated by each peer

Overlay with 1x bandwidth

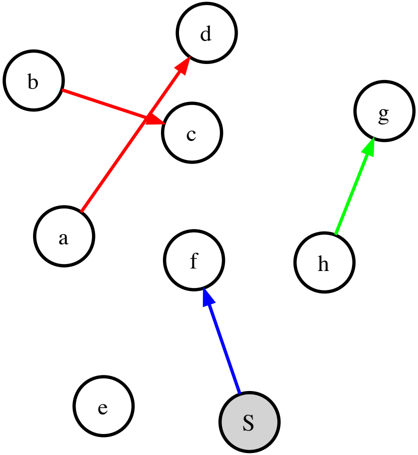


Step by step:

node	1	2	3	4	5
a	1	1			
b		1			
c					
d					
e					
f					
g					
h		2			

Indicates chunk currently replicated by each peer

Overlay with 1x bandwidth

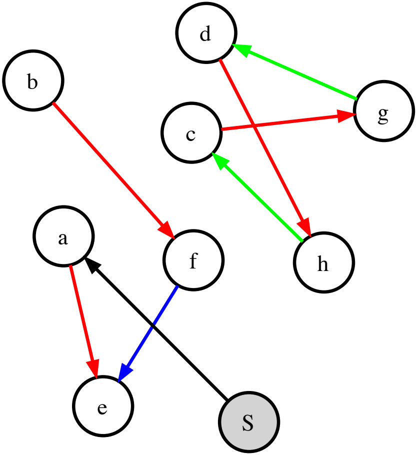


Step by step:

node	1	2	3	4	5
a	1	1	1		
b		1	1		
c			1		
d			1		
e					
f			3		
g			2		
h		2	2		

Indicates chunk currently replicated by each peer

Overlay with 1x bandwidth

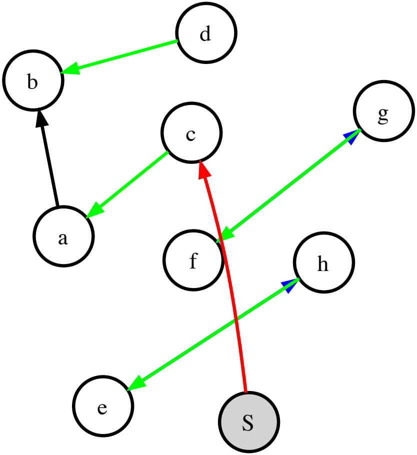


Step by step:

node	1	2	3	4	5
a	1	1	1	4	
b		1	1		
c			1	2	
d			1	2	
e				3	
f			3	3	
g			2	2	
h		2	2	2	

Indicates chunk currently replicated by each peer

Overlay with 1x bandwidth

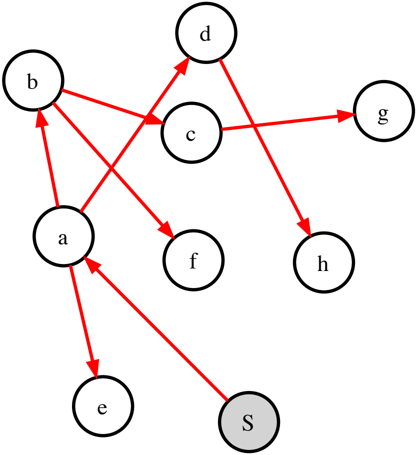


Step by step:

node	1	2	3	4	5
a	1	1	1	4	4
b		1	1		4
c			1	2	5
d			1	2	
e				3	3
f			3	3	3
g			2	2	3
h		2	2	2	3

Indicates chunk currently replicated by each peer

Overlay with 1x bandwidth



Step by step:

node	1	2	3	4	5
a	1	1	1	4	4
b		1	1		4
c			1	2	5
d			1	2	
e				3	3
f			3	3	3
g			2	2	3
h		2	2	2	3

Indicates chunk currently replicated by each peer

Optimal algorithm feasibility

- Sustainable, $\frac{n}{2}$ peers forwarding oldest piece, $\frac{n}{4}$ next one, $\frac{n}{8}$ next one and so on; $\sum_{i=1}^{\infty} \frac{n}{2} = n$
- After modification sustainable also for $n \neq 2^k$ (in $\lceil \log_2 n \rceil + 1$ time) (e.g. for $n = 9$ we need 1, 2, 3, 3 peers for each chunk, for $n = 11$ — 1, 2, 4, 4 etc.)
- Centralized algorithm will not scale
- Distributed implementation impossible?
  - Needs knowledge of whole O in every peer
  - Needs up to date knowledge of $H[i]$ (buffer states) and F (free peers)
  - Needs knowledge of other peers decisions

Optimal algorithm feasibility

- Sustainable, $\frac{n}{2}$ peers forwarding oldest piece, $\frac{n}{4}$ next one, $\frac{n}{8}$ next one and so on; $\sum_{i=1}^{\infty} \frac{n}{2} = n$
- After modification sustainable also for $n \neq 2^k$ (in $\lceil \log_2 n \rceil + 1$ time) (e.g. for $n = 9$ we need 1, 2, 3, 3 peers for each chunk, for $n = 11$ — 1, 2, 4, 4 etc.)
- Centralized algorithm will not scale
- Distributed implementation impossible?
  - Needs knowledge of whole G in every peer
  - Needs knowledge of other peers decisions

Optimal algorithm feasibility

- Sustainable, $\frac{n}{2}$ peers forwarding oldest piece, $\frac{n}{4}$ next one, $\frac{n}{8}$ next one and so on; $\sum_{i=1}^{\infty} \frac{n}{2} = n$
- After modification sustainable also for $n \neq 2^k$ (in $\lceil \log_2 n \rceil + 1$ time) (e.g. for $n = 9$ we need 1, 2, 3, 3 peers for each chunk, for $n = 11$ — 1, 2, 4, 4 etc.)
- Centralized algorithm will not scale
- Distributed implementation impossible?
  - Needs knowledge of whole O in every peer
  - Needs up to date knowledge of $H[i]$ (buffer states) and F (free peers)
  - Needs knowledge of other peers decisions

Optimal algorithm feasibility

- Sustainable, $\frac{n}{2}$ peers forwarding oldest piece, $\frac{n}{4}$ next one, $\frac{n}{8}$ next one and so on; $\sum_{i=1}^{\infty} \frac{n}{2} = n$
- After modification sustainable also for $n \neq 2^k$ (in $\lceil \log_2 n \rceil + 1$ time) (e.g. for $n = 9$ we need 1, 2, 3, 3 peers for each chunk, for $n = 11$ — 1, 2, 4, 4 etc.)
- Centralized algorithm will not scale
- Distributed implementation impossible?
  - Needs knowledge of whole O in every peer
  - Needs up to date knowledge of $H[i]$ (buffer states) and F (free peers)
  - Needs knowledge of other peers decisions

Optimal algorithm feasibility

- Sustainable, $\frac{n}{2}$ peers forwarding oldest piece, $\frac{n}{4}$ next one, $\frac{n}{8}$ next one and so on; $\sum_{i=1}^{\infty} \frac{n}{2} = n$
- After modification sustainable also for $n \neq 2^k$ (in $\lceil \log_2 n \rceil + 1$ time) (e.g. for $n = 9$ we need 1, 2, 3, 3 peers for each chunk, for $n = 11$ — 1, 2, 4, 4 etc.)
- Centralized algorithm will not scale
- Distributed implementation impossible?
  - Needs knowledge of whole O in every peer
  - Needs up to date knowledge of $H[i]$ (buffer states) and F (free peers)
  - Needs knowledge of other peers decisions

Optimal algorithm feasibility

- Sustainable, $\frac{n}{2}$ peers forwarding oldest piece, $\frac{n}{4}$ next one, $\frac{n}{8}$ next one and so on; $\sum_{i=1}^{\infty} \frac{n}{2} = n$
- After modification sustainable also for $n \neq 2^k$ (in $\lceil \log_2 n \rceil + 1$ time) (e.g. for $n = 9$ we need 1, 2, 3, 3 peers for each chunk, for $n = 11$ — 1, 2, 4, 4 etc.)
- Centralized algorithm will not scale
- Distributed implementation impossible?
  - Needs knowledge of whole O in every peer
  - Needs up to date knowledge of $H[i]$ (buffer states) and F (free peers)
  - Needs knowledge of other peers decisions

Optimal algorithm feasibility

- Sustainable, $\frac{n}{2}$ peers forwarding oldest piece, $\frac{n}{4}$ next one, $\frac{n}{8}$ next one and so on; $\sum_{i=1}^{\infty} \frac{n}{2} = n$
- After modification sustainable also for $n \neq 2^k$ (in $\lceil \log_2 n \rceil + 1$ time) (e.g. for $n = 9$ we need 1, 2, 3, 3 peers for each chunk, for $n = 11$ — 1, 2, 4, 4 etc.)
- Centralized algorithm will not scale
- Distributed implementation impossible?
    - Needs knowledge of whole O in every peer
    - Needs up to date knowledge of $H[i]$ (buffer states) and F (free peers)
    - Needs knowledge of other peers decisions

Problem definition 4

- Disseminate a stream of data
- Single source
- Multiple recipients
- Recipients contribute to further disseminate
- Finite dissemination deadline
- High bandwidth utilization
- Participants are autonomous
- Local, delayed view

Why harder than BT?

Similar to BitTorrent, but also very different:

- Always in *flash crowd* state
- Each piece has a deadline
- Limited number of pieces *alive*
- "Computer working, but unattended" improbable

Why harder than BT?

Similar to BitTorrent, but also very different:

- Always in *flash crowd* state
- Each piece has a deadline
- Limited number of pieces *alive*
- "Computer working, but unattended" improbable

Why harder than BT?

Similar to BitTorrent, but also very different:

- Always in *flash crowd* state
- Each piece has a deadline
- Limited number of pieces *alive*
- "Computer working, but unattended" improbable

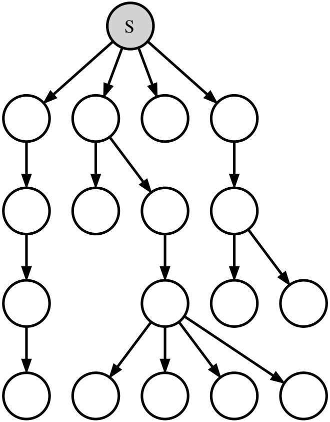
Why harder than BT?

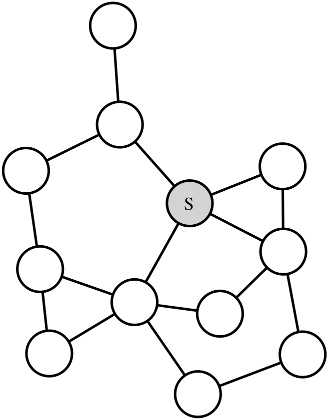
Similar to BitTorrent, but also very different:

- Always in *flash crowd* state
- Each piece has a deadline
- Limited number of pieces *alive*
- "Computer working, but unattended" improbable

Outline

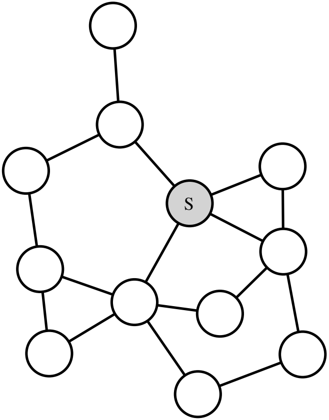
Types of overlays





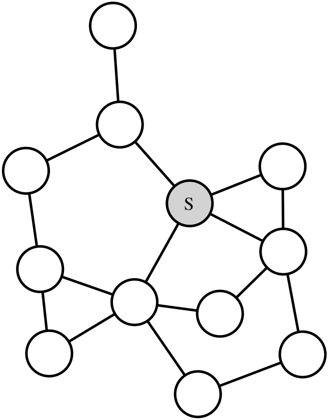
Unstructured overlays

- Many names, similar idea:
  - Gossiping
  - Flood routing
  - BitTorrent-like
- Peers arrange a random graph
- Simple algorithms
- Robust
- Most popular

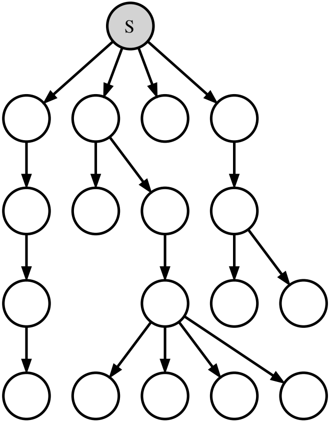


Unstructured overlays

- Many names, similar idea:
  - Gossiping
  - Flood routing
  - BitTorrent-like
- Peers arrange a random graph
- Simple algorithms
- Robust
- Most popular



Structured overlays

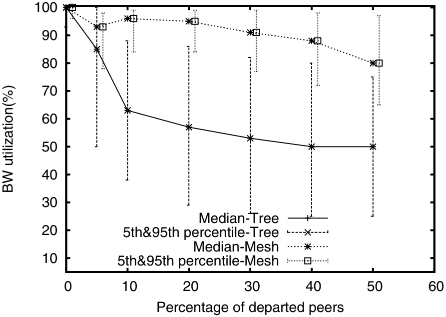


- Define explicit structure, usually forest
- Much easier to understand
- Much harder to construct
- Employs DHT
- Prone to disruptions

Structured vs. Unstructured

Comprehensive comparison: *Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches* by Magharei et al.

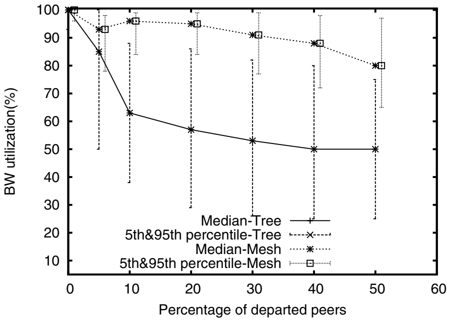
- State of the art overlays of both types
- Comparison over a broad range of scenarios
- Many observed characteristics
- Packet-level simulations
- Explanations for observed phenomena
- Pretty conclusive: unstructured overlays are better



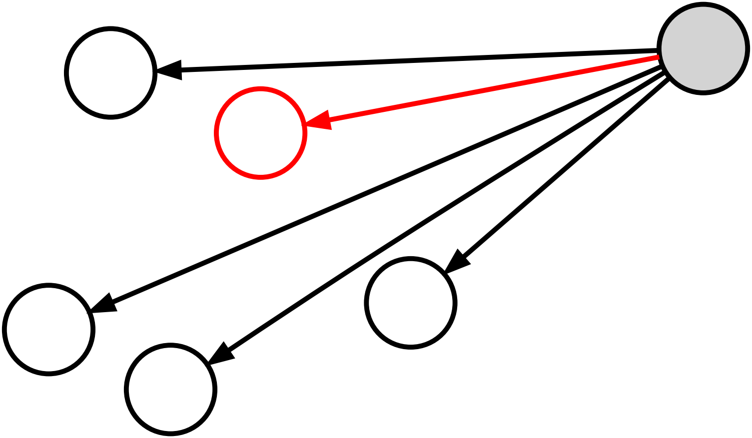
Structured vs. Unstructured

Comprehensive comparison: *Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches* by Magharei et al.

- State of the art overlays of both types
- Comparison over a broad range of scenarios
- Many observed characteristics
- Packet-level simulations
- Explanations for observed phenomena
- Pretty conclusive: unstructured overlays are better

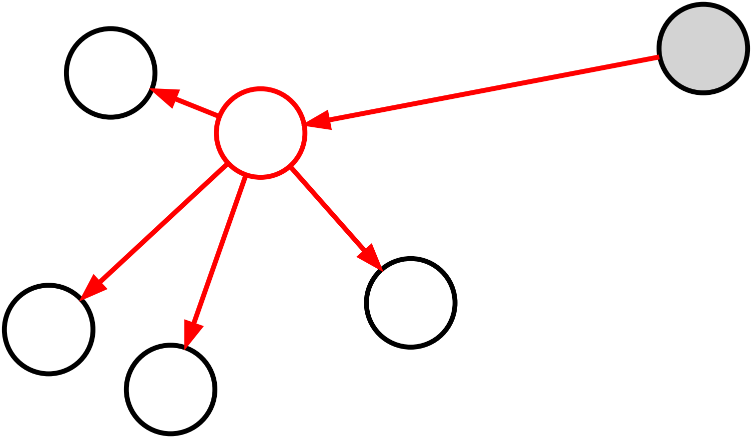


Adaptive Queue-based Chunk Scheduling



Source pushes a single copy of each fragment to a single replicator.
That replicator pushes it to everyone else.

Adaptive Queue-based Chunk Scheduling



Source pushes a single copy of each fragment to a single replicator.
That replicator pushes it to everyone else.

AQCS Properties

- Very simple
- Very robust
- Achieves optimal performance — providing that:
  - Chunk size is an *common divisor* of all bandwidths
  - Chunk size is smaller than $\frac{bandwidth \cdot delay}{peers\#}$
  - Theoretical proof for infinitesimal chunk size and zero propagation delay
- Practical limit, as found by authors, is about 40 peers

For more details read *Aqcs: Adaptive queue-based chunk scheduling for P2P live streaming* by Guo, Liang and Liu

AQCS Properties

- Very simple
- Very robust
- Achieves optimal performance — providing that:
  - Chunk size is an *common divisor* of all bandwidths
  - Chunk size is smaller than $\frac{bandwidth \cdot delay}{peers\#}$
  - Theoretical proof for infinitesimal chunk size and zero propagation delay
- Practical limit, as found by authors, is about 40 peers

For more details read *Aqcs: Adaptive queue-based chunk scheduling for P2P live streaming* by Guo, Liang and Liu

AQCS Properties

- Very simple
- Very robust
- Achieves optimal performance — providing that:
  - Chunk size is an *common divisor* of all bandwidths
  - Chunk size is smaller than $\frac{bandwidth \cdot delay}{peers\#}$
  - Theoretical proof for infinitesimal chunk size and zero propagation delay
- Practical limit, as found by authors, is about 40 peers

For more details read *Aqcs: Adaptive queue-based chunk scheduling for P2P live streaming* by Guo, Liang and Liu

AQCS Properties

- Very simple
- Very robust
- Achieves optimal performance — providing that:
  - Chunk size is an *common divisor* of all bandwidths
  - Chunk size is smaller than $\frac{bandwidth \cdot delay}{peers\#}$
  - Theoretical proof for infinitesimal chunk size and zero propagation delay
- Practical limit, as found by authors, is about 40 peers

For more details read *Aqcs: Adaptive queue-based chunk scheduling for P2P live streaming* by Guo, Liang and Liu

Local view randomness

We can assume a few things about the local view of a node:

- Approximates a random sample of overlay
- Constantly changing
- Resilient
- *CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays* by Voulgaris et al. proposes a simple algorithm that's good against massive failures, by neighbour exchange
- Random walk algorithms may help against Byzantine adversaries, as shown in *Uniform and Ergodic Sampling in Unstructured Peer-to-Peer Systems with Malicious Nodes* by Anceaume et al.

Unstructured overlay basic algorithms

- *Random push* (or *random pull*) based
  - Each peer chooses each turn a peer to send to at random
  - Proved to propagate information in $\Theta(\log n)$ steps
  - Other simple peer selection schemes: tit-for-tat, deprived peer
  - Also possible to first select chunk and then peer for that
- *Chunk selection* algorithms can be divided into main groups:
  - By order:
    - Random
    - Latest
  - By awareness:
    - Useful
    - Blind

Unstructured overlay basic algorithms

- *Random push* (or *random pull*) based
  - Each peer chooses each turn a peer to send to at random
  - Proved to propagate information in $\Theta(\log n)$ steps
  - Other simple peer selection schemes: tit-for-tat, deprived peer
  - Also possible to first select chunk and then peer for that
- *Chunk selection* algorithms can be divided into main groups:
  - By order
    - Random
    - Latest
  - By awareness
    - Useful
    - Blind

Unstructured overlay basic algorithms

- *Random push* (or *random pull*) based
  - Each peer chooses each turn a peer to send to at random
  - Proved to propagate information in $\Theta(\log n)$ steps
  - Other simple peer selection schemes: tit-for-tat, deprived peer
  - Also possible to first select chunk and then peer for that
- *Chunk selection* algorithms can be divided into main groups:
  - By order:
    - Random
    - Latest
  - By awareness:
    - Useful
    - Blind

Unstructured overlay basic algorithms

- *Random push* (or *random pull*) based
  - Each peer chooses each turn a peer to send to at random
  - Proved to propagate information in $\Theta(\log n)$ steps
  - Other simple peer selection schemes: tit-for-tat, deprived peer
  - Also possible to first select chunk and then peer for that
- *Chunk selection* algorithms can be divided into main groups:
  - By order:
    - Random
    - Latest
  - By awareness:
    - Useful
    - Blind

Unstructured overlay basic algorithms

- *Random push* (or *random pull*) based
  - Each peer chooses each turn a peer to send to at random
  - Proved to propagate information in $\Theta(\log n)$ steps
  - Other simple peer selection schemes: tit-for-tat, deprived peer
  - Also possible to first select chunk and then peer for that
- *Chunk selection* algorithms can be divided into main groups:
  - By order:
    - Random
    - Latest
  - By awareness:
    - Useful
    - Blind

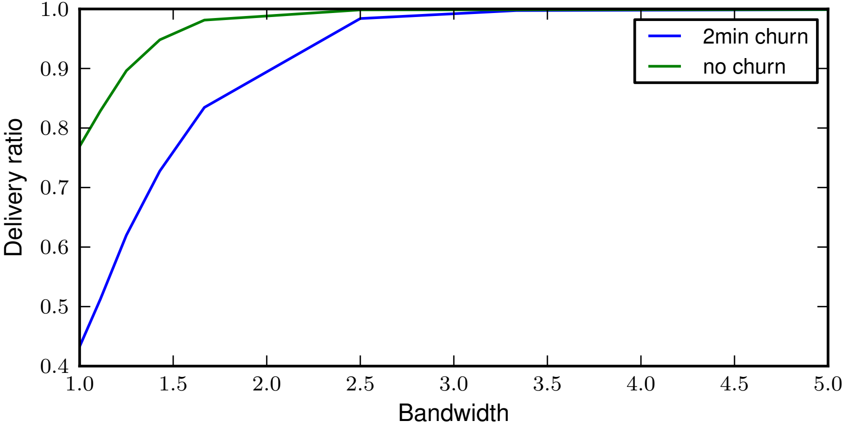
Unstructured overlay basic algorithms

- *Random push* (or *random pull*) based
  - Each peer chooses each turn a peer to send to at random
  - Proved to propagate information in $\Theta(\log n)$ steps
  - Other simple peer selection schemes: tit-for-tat, deprived peer
  - Also possible to first select chunk and then peer for that
- *Chunk selection* algorithms can be divided into main groups:
  - By order:
    - Random
    - Latest
  - By awareness:
    - Useful
    - Blind

Unstructured overlay basic algorithms

- *Random push* (or *random pull*) based
  - Each peer chooses each turn a peer to send to at random
  - Proved to propagate information in $\Theta(\log n)$ steps
  - Other simple peer selection schemes: tit-for-tat, deprived peer
  - Also possible to first select chunk and then peer for that
- *Chunk selection* algorithms can be divided into main groups:
  - By order:
    - Random
    - Latest
  - By awareness:
    - Useful
    - Blind

Why simple push/pull schemes insufficient?



- Random push loses bandwidth on duplicate transfers
- Random pull has higher chance of content bottleneck
- Both ways simple schemes utilize a fraction of bandwidth

Idea — push-pull scheme

Very simple basic idea:

- When a chunk is new, most peers don't have it — push it without asking
- If have only chunks with high expected popularity — respond to pull requests

Connecting best of both approaches:

- Initial exponential growth of chunk owners
- Almost no duplicate transfers

Funny problem: many different approaches under this name

Thank you for your attention

¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ? ? ? ? ? ? ? ? ? ?

Centralized optimal algorithm

- n – number of peers in overlay, $n = 2^k + 1$ including source
- O – overlay
- F – *free* peers, initially equal to $O \setminus \textit{source}$
- o – oldest chunk in transfer
- $H[i]$ – set of peers who have chunk i

1. If $|H[o]| = \frac{n}{2}$, then push from each peer p in $H[o]$ chunk o to some peer in $O \setminus H[o]$, add p to F, let $o = o + 1$
2. For $i = o, \ldots$, for each peer p in $H[i]$ push chunk i to some peer q in F, remove q from F
3. Push newest chunk from source to some p in F, remove p from F
4. Return to step 1

Centralized optimal algorithm

- n – number of peers in overlay, $n = 2^k + 1$ including source
- O – overlay
- F – *free* peers, initially equal to $O \setminus \mathit{source}$
- o – oldest chunk in transfer
- $H[i]$ – set of peers who have chunk i

1. If $|H[o]| = \frac{n}{2}$, then push from each peer p in $H[o]$ chunk o to some peer in $O \setminus H[o]$, add p to F, let $o = o + 1$
2. For $i = o, \ldots$, for each peer p in $H[i]$ push chunk i to some peer q in F, remove q from F
3. Push newest chunk from source to some p in F, remove p from F
4. Return to step 1

Churn

- *Node dynamics* shown to be biggest problem of live systems
- When $n \approx 20000$, almost 1000 peers join and leave per minute
- Biggest reason for unstructured overlay popularity
- Almost no insight in literature
- No difference between new and returning peers — buffers probably outdated

Churn

- *Node dynamics* shown to be biggest problem of live systems
- When $n \approx 20000$, almost 1000 peers join and leave per minute
- Biggest reason for unstructured overlay popularity
- Almost no insight in literature
- No difference between new and returning peers — buffers probably outdated

Churn

- *Node dynamics* shown to be biggest problem of live systems
- When $n \approx 20000$, almost 1000 peers join and leave per minute
- Biggest reason for unstructured overlay popularity
- Almost no insight in literature
- No difference between new and returning peers — buffers probably outdated

Churn

- *Node dynamics* shown to be biggest problem of live systems
- When $n \approx 20000$, almost 1000 peers join and leave per minute
- Biggest reason for unstructured overlay popularity
- Almost no insight in literature
- No difference between new and returning peers — buffers probably outdated

Churn

- *Node dynamics* shown to be biggest problem of live systems
- When $n \approx 20000$, almost 1000 peers join and leave per minute
- Biggest reason for unstructured overlay popularity
- Almost no insight in literature
- No difference between new and returning peers — buffers probably outdated

Effects of churn

- Chunks transferred to leaving peer are lost
- New peer has empty buffer
  - Nothing to push
- Interrupts both incoming and outgoing transfers
- Problems with interpreting the performance

Effects of churn

- Chunks transferred to leaving peer are lost
- New peer has empty buffer
  - Nothing to push
  - Can't do tit-for-tat
  - May attract duplicate transfers
  - First chunk we get will be the most popular one
- Interrupts both incoming and outgoing transfers
- Problems with interpreting the performance
  - Allowing buffering time we allow a peer with unobserved performance
  - Without buffering time statistics biased by initially empty buffer
  - If peer with bad buffer leaves, the overlay performance goes up
  - My solution: observer peer – a peer that does not experience churn

Effects of churn

- Chunks transferred to leaving peer are lost
- New peer has empty buffer
  - Nothing to push
  - Can't do tit-for-tat
  - May attract duplicate transfers
  - First chunk we get will be the most popular one
- Interrupts both incoming and outgoing transfers
- Problems with interpreting the performance

Effects of churn

- Chunks transferred to leaving peer are lost
- New peer has empty buffer
  - Nothing to push
  - Can't do tit-for-tat
  - May attract duplicate transfers
  - First chunk we get will be the most popular one
- Interrupts both incoming and outgoing transfers
- Problems with interpreting the performance

Effects of churn

- Chunks transferred to leaving peer are lost
- New peer has empty buffer
  - Nothing to push
  - Can't do tit-for-tat
  - May attract duplicate transfers
  - First chunk we get will be the most popular one
- Interrupts both incoming and outgoing transfers
- Problems with interpreting the performance
  - Allowing buffering time we allow a peer with unobserved performance
  - Without buffering time statistics biased by initially empty buffer
  - If peer with bad buffer leaves, the overlay performance goes up
  - My solution: observer peer – a peer that does not experience churn

Effects of churn

- Chunks transferred to leaving peer are lost
- New peer has empty buffer
  - Nothing to push
  - Can't do tit-for-tat
  - May attract duplicate transfers
  - First chunk we get will be the most popular one
- Interrupts both incoming and outgoing transfers
- Problems with interpreting the performance
  - Allowing buffering time we allow a peer with unobserved performance
  - Without buffering time statistics biased by initially empty buffer
  - If peer with bad buffer leaves, the overlay performance goes up
  - My solution: observer peer – a peer that does not experience churn

Effects of churn

- Chunks transferred to leaving peer are lost
- New peer has empty buffer
  - Nothing to push
  - Can't do tit-for-tat
  - May attract duplicate transfers
  - First chunk we get will be the most popular one
- Interrupts both incoming and outgoing transfers
- Problems with interpreting the performance
  - Allowing buffering time we allow a peer with unobserved performance
  - Without buffering time statistics biased by initially empty buffer
  - If peer with bad buffer leaves, the overlay performance goes up
  - My solution: observer peer – a peer that does not experience churn

Effects of churn

- Chunks transferred to leaving peer are lost
- New peer has empty buffer
  - Nothing to push
  - Can't do tit-for-tat
  - May attract duplicate transfers
  - First chunk we get will be the most popular one
- Interrupts both incoming and outgoing transfers
- Problems with interpreting the performance
  - Allowing buffering time we allow a peer with unobserved performance
  - Without buffering time statistics biased by initially empty buffer
  - If peer with bad buffer leaves, the overlay performance goes up
  - My solution: observer peer – a peer that does not experience churn

Effects of churn

- Chunks transferred to leaving peer are lost
- New peer has empty buffer
  - Nothing to push
  - Can't do tit-for-tat
  - May attract duplicate transfers
  - First chunk we get will be the most popular one
- Interrupts both incoming and outgoing transfers
- Problems with interpreting the performance
  - Allowing buffering time we allow a peer with unobserved performance
  - Without buffering time statistics biased by initially empty buffer
  - If peer with bad buffer leaves, the overlay performance goes up
  - My solution: observer peer – a peer that does not experience churn

Effects of churn

- Chunks transferred to leaving peer are lost
- New peer has empty buffer
  - Nothing to push
  - Can't do tit-for-tat
  - May attract duplicate transfers
  - First chunk we get will be the most popular one
- Interrupts both incoming and outgoing transfers
- Problems with interpreting the performance
  - Allowing buffering time we allow a peer with unobserved performance
  - Without buffering time statistics biased by initially empty buffer
  - If peer with bad buffer leaves, the overlay performance goes up
  - My solution: observer peer – a peer that does not experience churn

Effects of churn

- Chunks transferred to leaving peer are lost
- New peer has empty buffer
  - Nothing to push
  - Can't do tit-for-tat
  - May attract duplicate transfers
  - First chunk we get will be the most popular one
- Interrupts both incoming and outgoing transfers
- Problems with interpreting the performance
  - Allowing buffering time we allow a peer with unobserved performance
  - Without buffering time statistics biased by initially empty buffer
  - If peer with bad buffer leaves, the overlay performance goes up
  - My solution: observer peer – a peer that does not experience churn

Effects of churn

- Chunks transferred to leaving peer are lost
- New peer has empty buffer
  - Nothing to push
  - Can't do tit-for-tat
  - May attract duplicate transfers
  - First chunk we get will be the most popular one
- Interrupts both incoming and outgoing transfers
- Problems with interpreting the performance
  - Allowing buffering time we allow a peer with unobserved performance
  - Without buffering time statistics biased by initially empty buffer
  - If peer with bad buffer leaves, the overlay performance goes up
  - My solution: observer peer – a peer that does not experience churn

Simulator

- Core: 2550 lines of Python
- Can do about 70000 individual transfers/second
- 50000 peers requires only 300MB of RAM
- Well tested
- Easily extensible
- Scripts for preparing simulation series, distributed running, results analysis; mostly Perl and shell

Simulator

- Core: 2550 lines of Python
- Can do about 70000 individual transfers/second
- 50000 peers requires only 300MB of RAM
- Well tested
- Easily extensible
- Scripts for preparing simulation series, distributed running, results analysis; mostly Perl and shell

Simulator

- Core: 2550 lines of Python
- Can do about 70000 individual transfers/second
- 50000 peers requires only 300MB of RAM
- Well tested
- Easily extensible
- Scripts for preparing simulation series, distributed running, results analysis; mostly Perl and shell

Simulation results

- Expected number of peers is 500
- Lasts 10000 fragments
- Multiple runs per data point in plots
- Data from over 5000 simulations
- Peers join according to a Poisson process
- Peers have an exponentially distributed life time

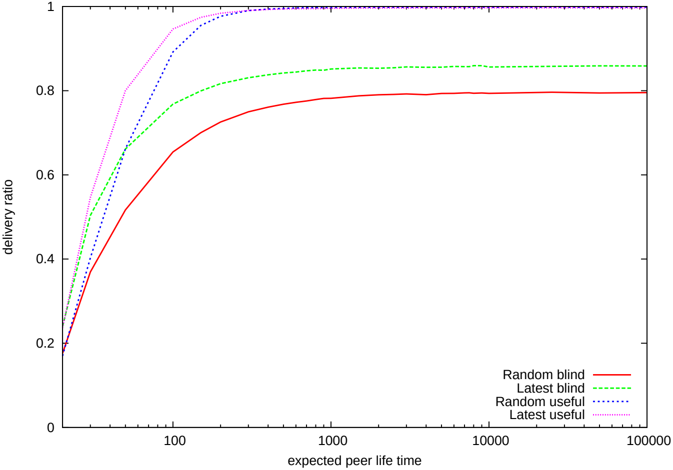
Simulation results

- Expected number of peers is 500
- Lasts 10000 fragments
- Multiple runs per data point in plots
- Data from over 5000 simulations
- Peers join according to a Poisson process
- Peers have an exponentially distributed life time

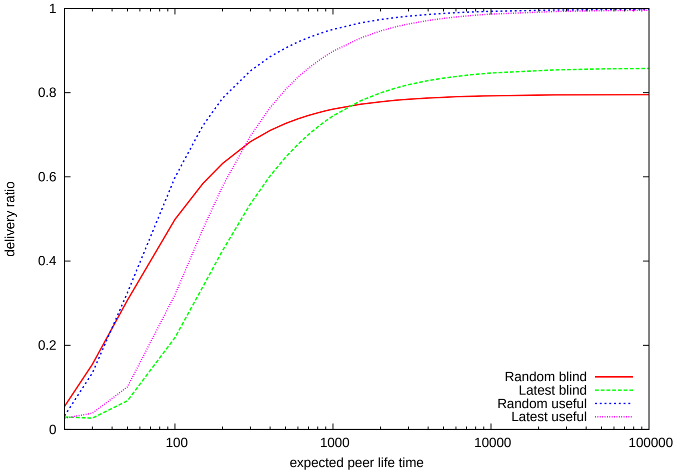
Simulation results

- Expected number of peers is 500
- Lasts 10000 fragments
- Multiple runs per data point in plots
- Data from over 5000 simulations
- Peers join according to a Poisson process
- Peers have an exponentially distributed life time

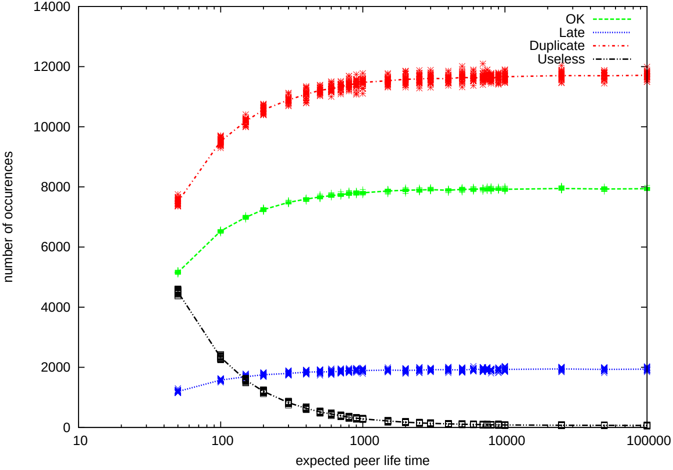
Algorithms comparison — observer



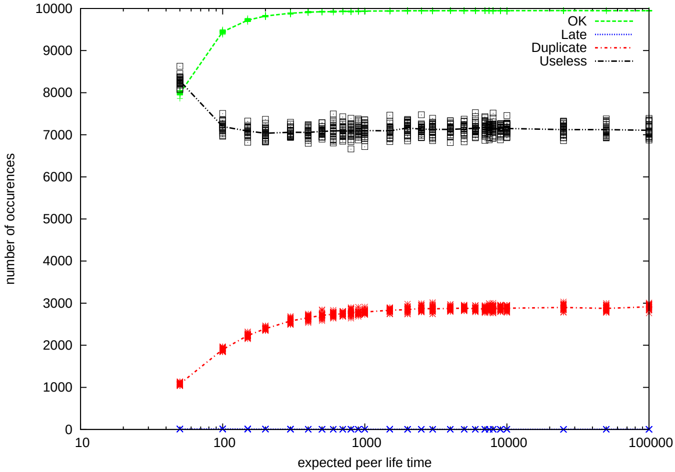
Algorithms comparison — global



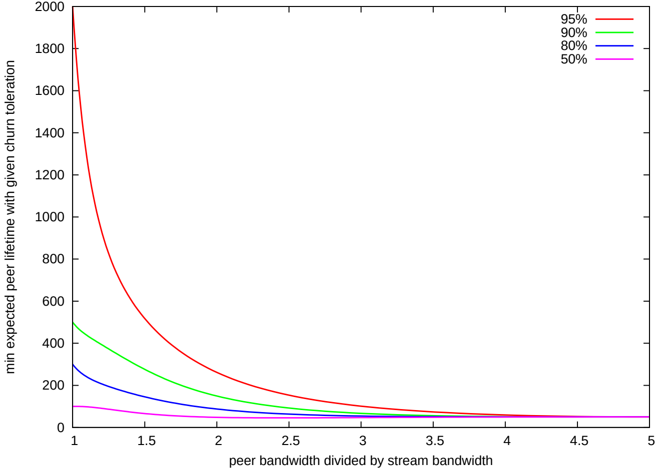
Transfer outcomes — random blind



Transfer outcomes — latest useful



Latest useful churn toleration



Latest useful bandwidth/deadline performance