Live Streaming with Gossip

Maxime Monod

June 30, 2010

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Live streaming

A source produces **multimedia content**

n viewers (*n* large)

Regular TV: everything **HD**

VS

IP TV, Web TV, P2P TV, …

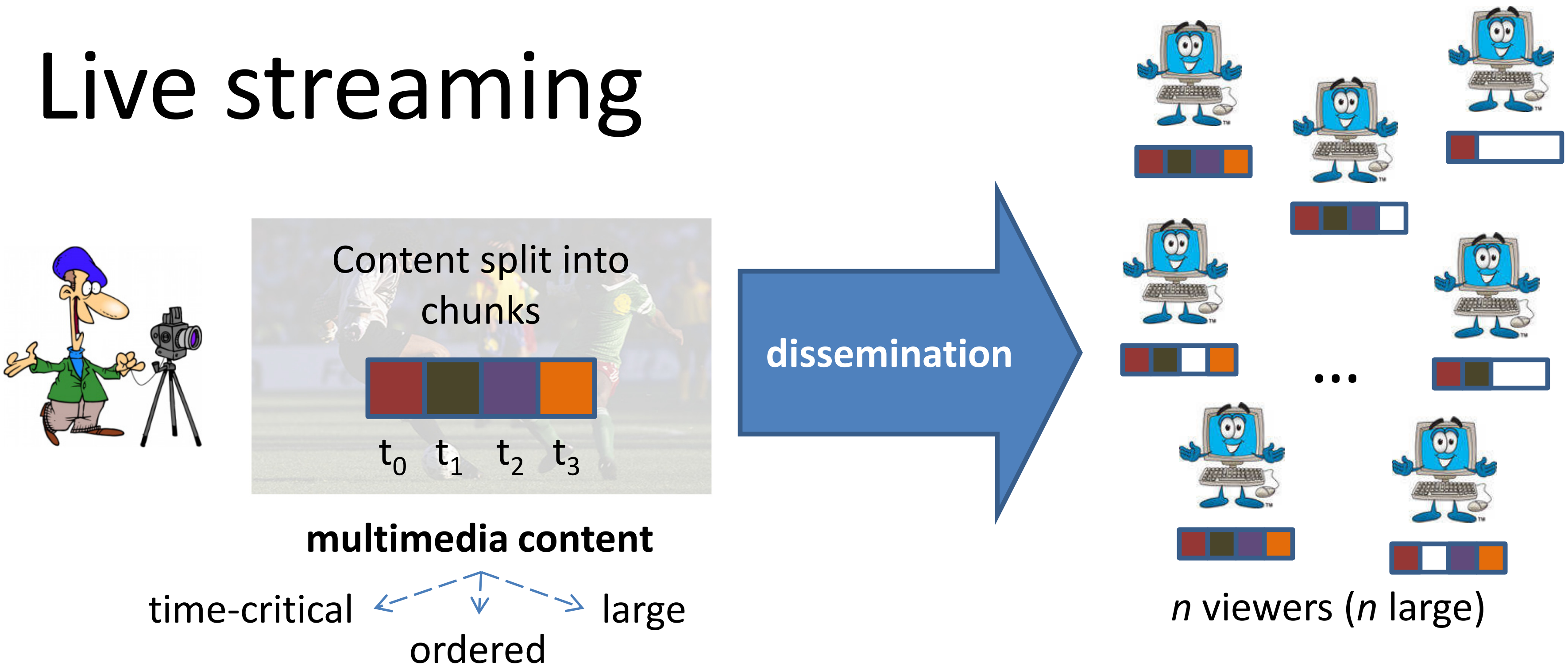
192K requests/day

78K users/day

244K simultaneous users (incl. VoD)

BBC iStats (April 2010)

Live streaming

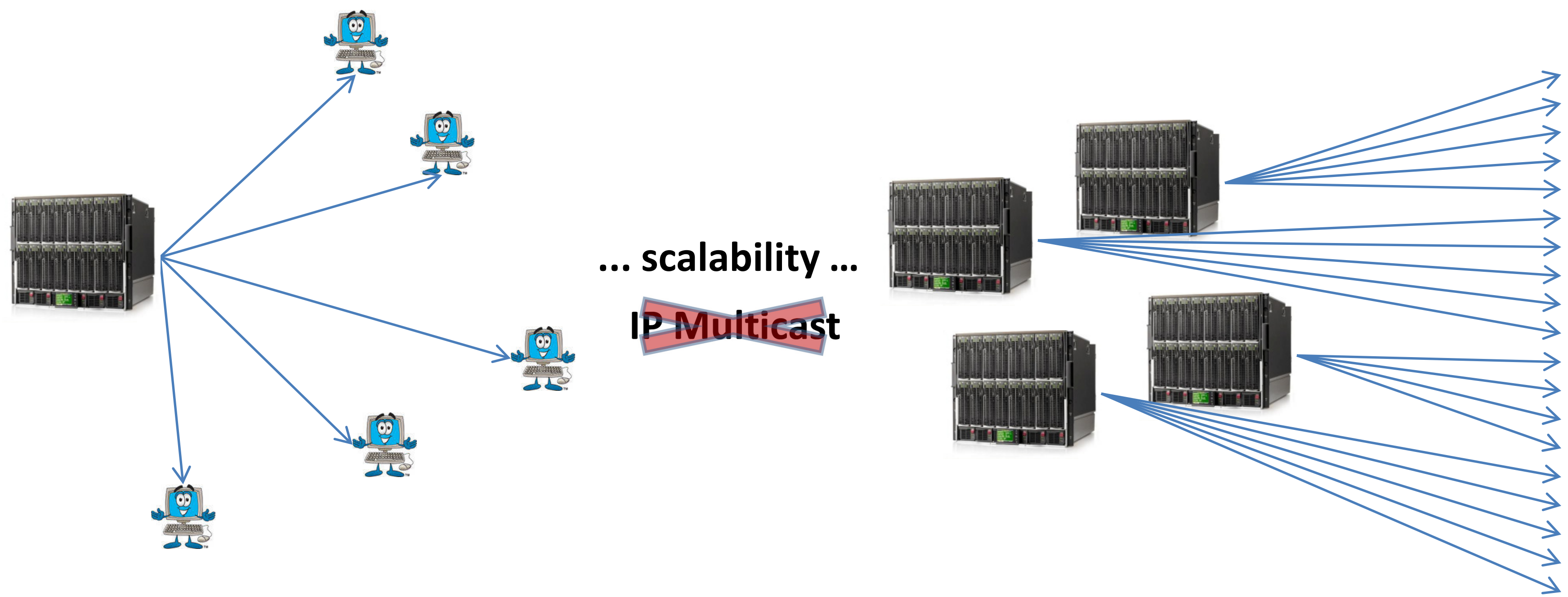


- Stream rate ***s*** [kbps]
- ***n*** viewers want to receive ***s***

Demand = Supply

Natural solution

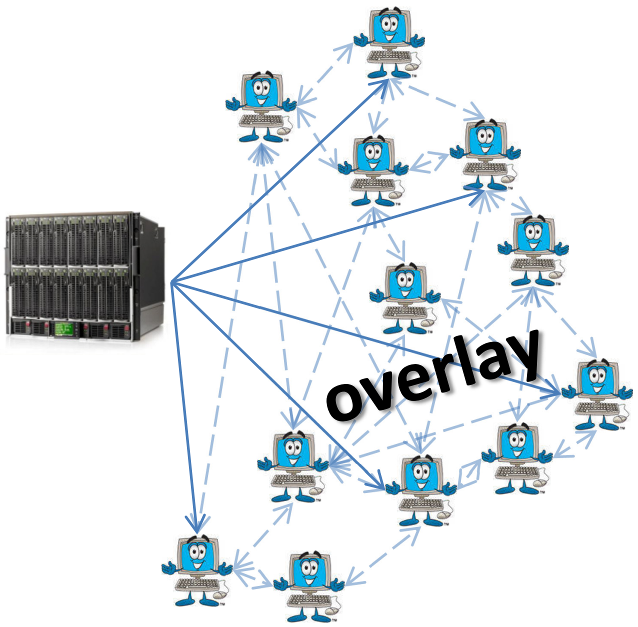
- "Centralized" solution



Participants are pure consumer

Context of this thesis

- "Decentralized" solution

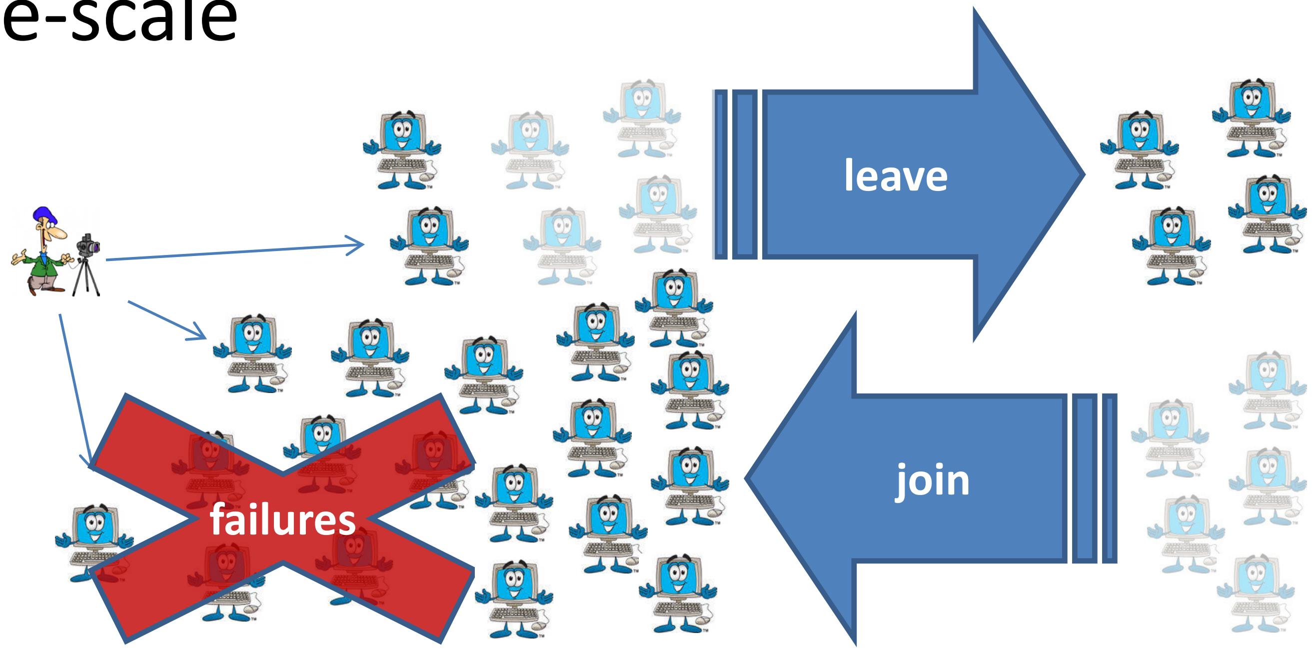


Participants **collaborate**

...most of them!

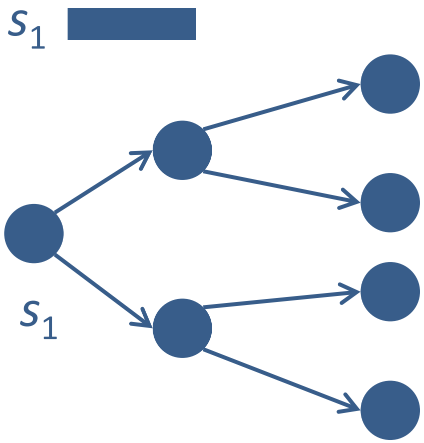
Environment

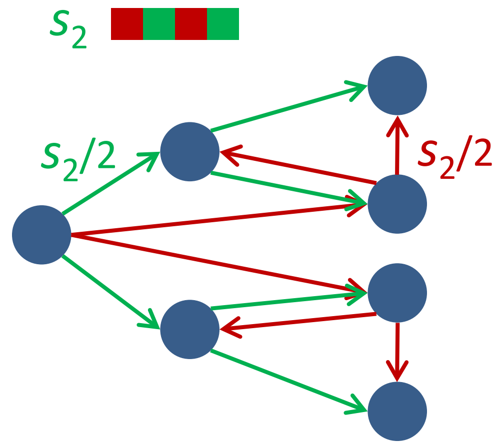
- Large-scale

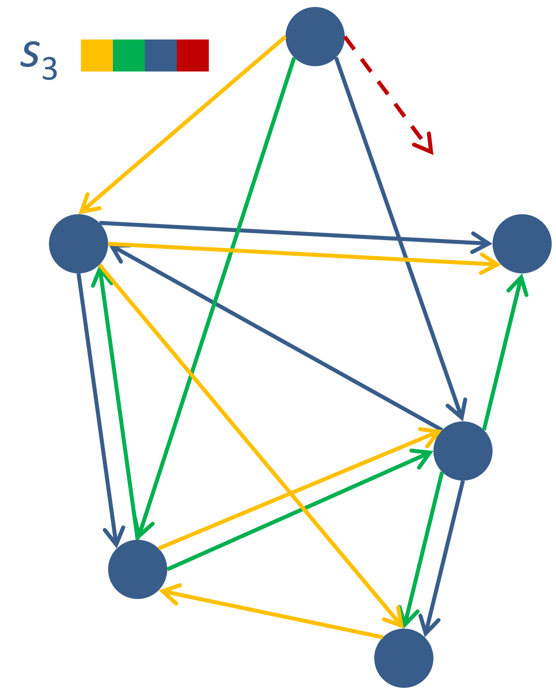


- Constrained bandwidth
  - Asymmetric (e.g., ADSL)

Existing approaches



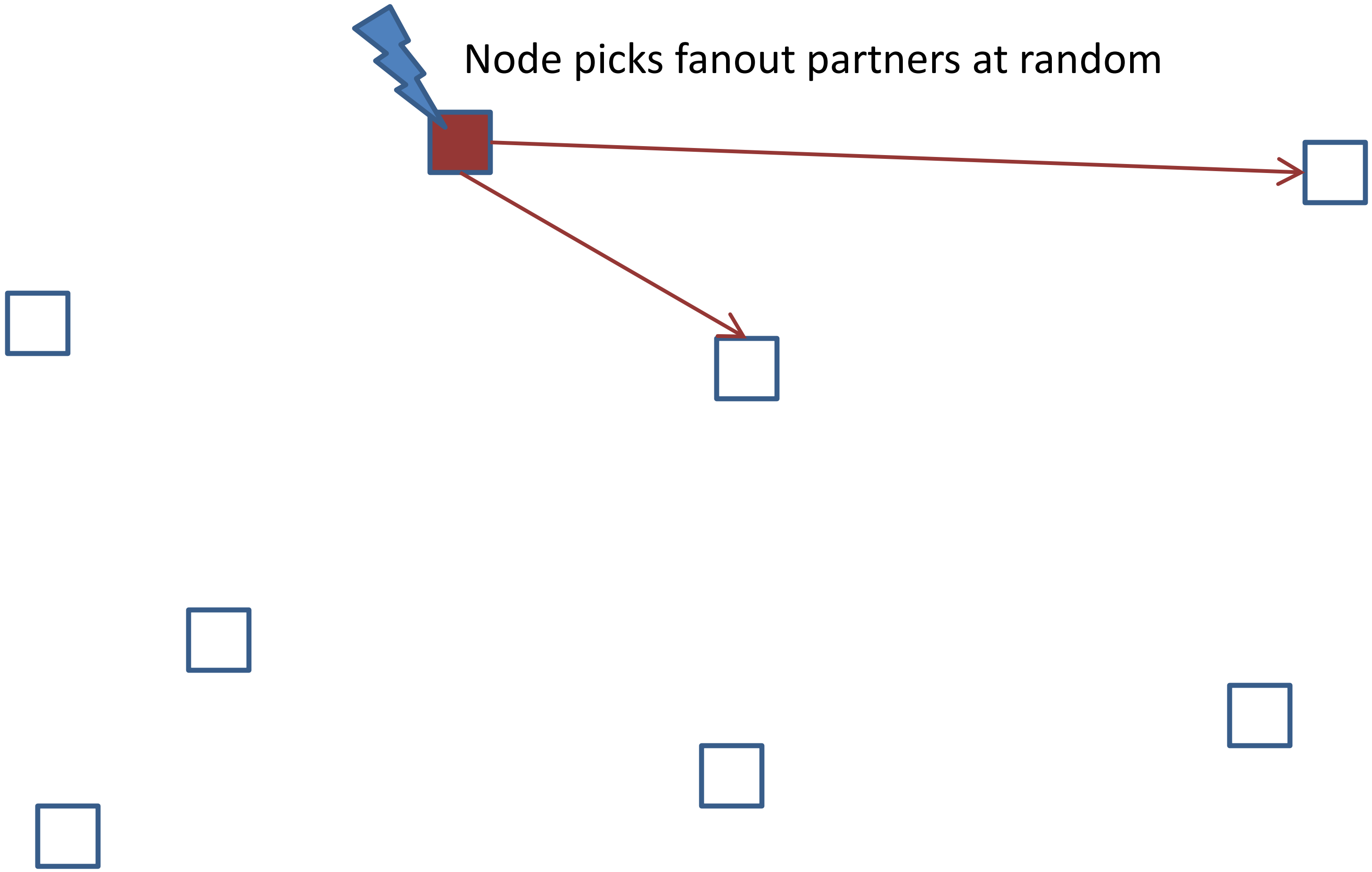




Single tree	**Multiple trees**	**Mesh**
s_1 is constrained by design	**Upload of nodes: multiple of s_2/z**	**s_3 optimal**
Disconnection	**Partial disconnection**	**Connected is not enough**
Build/maintain tree	**Build/maintain *z* trees**	**Peer selection, Packet scheduling**

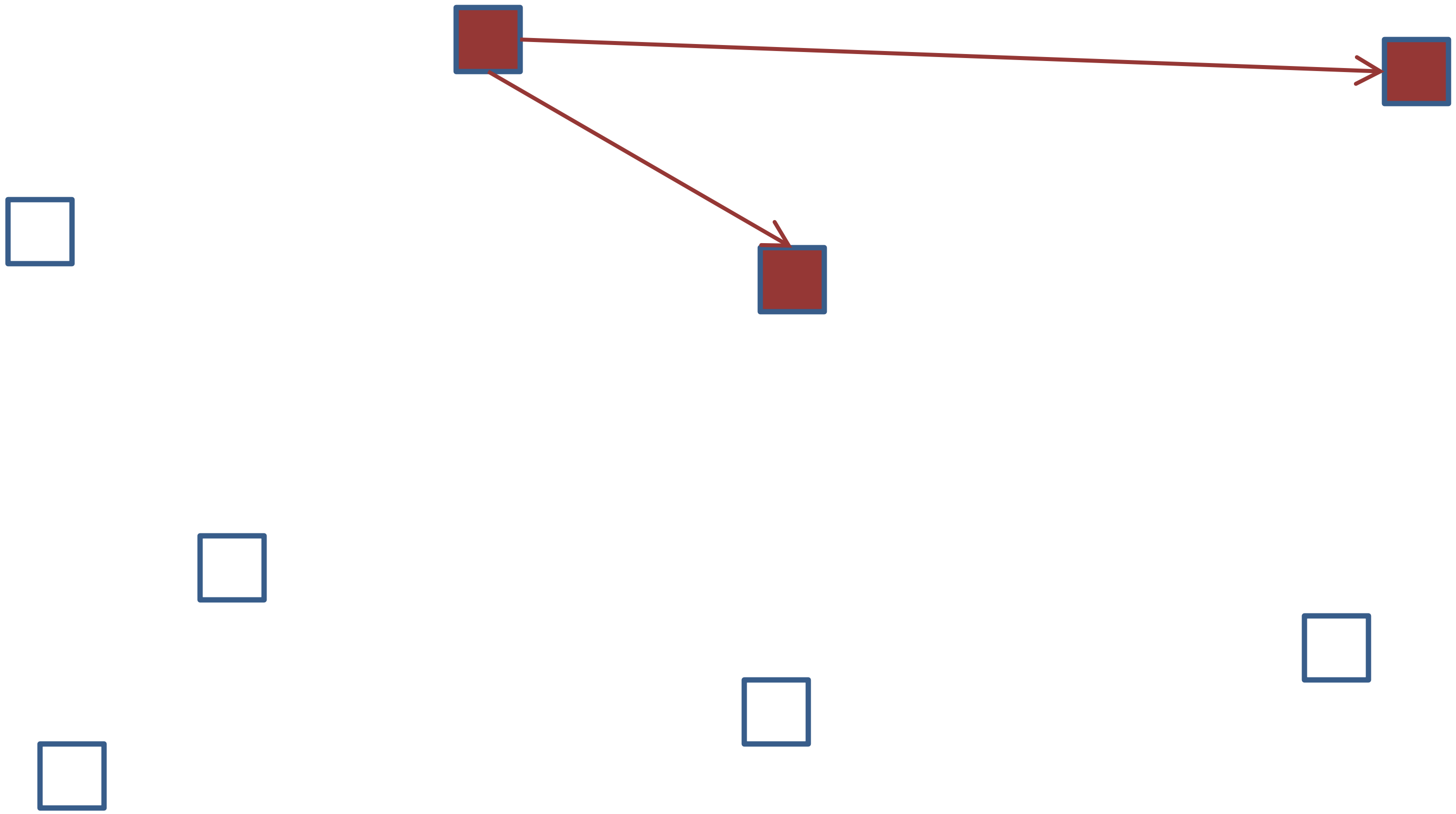
Beyond mesh: Gossip

- Gossip-based dissemination



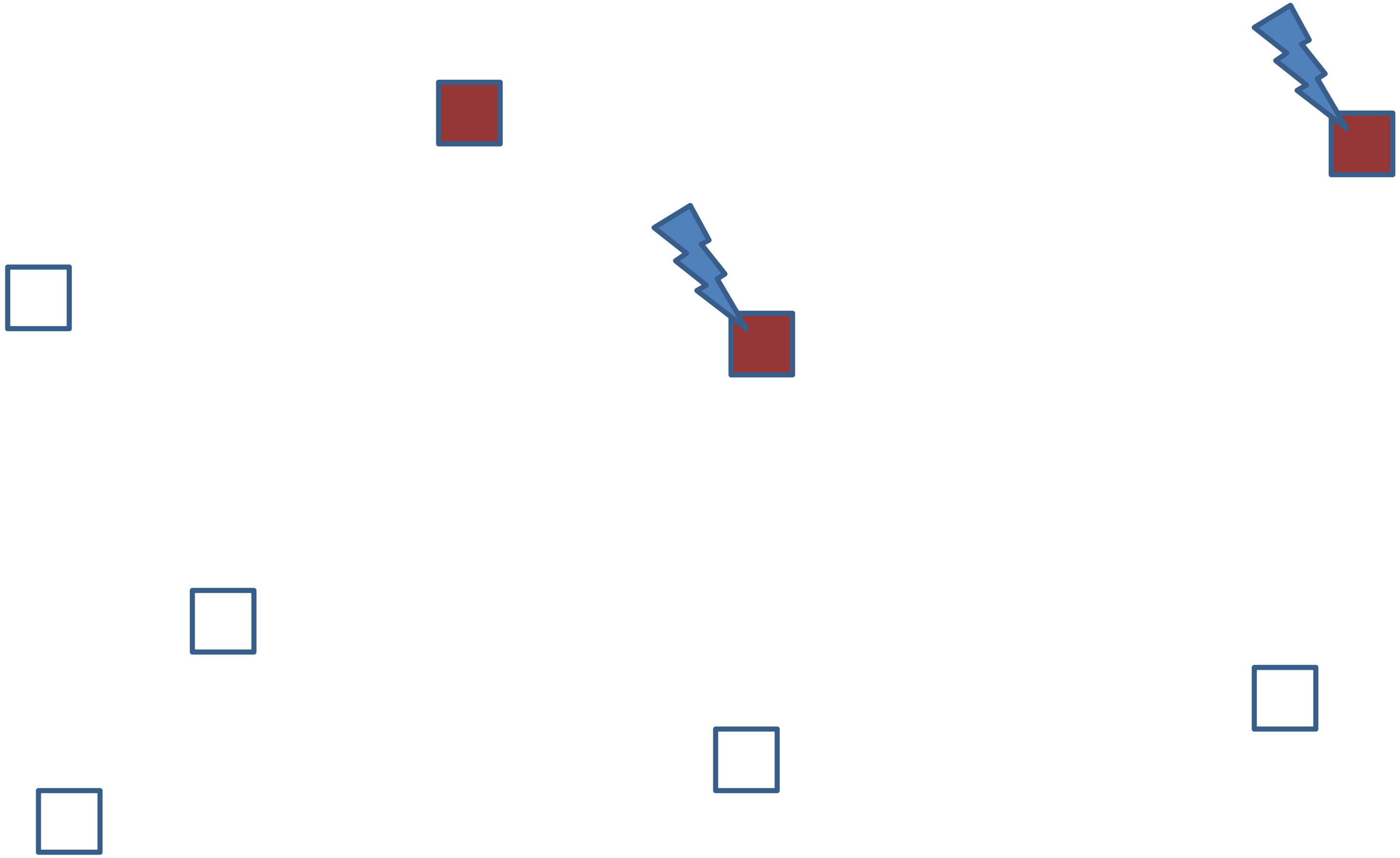
Beyond mesh: Gossip

- Gossip-based dissemination



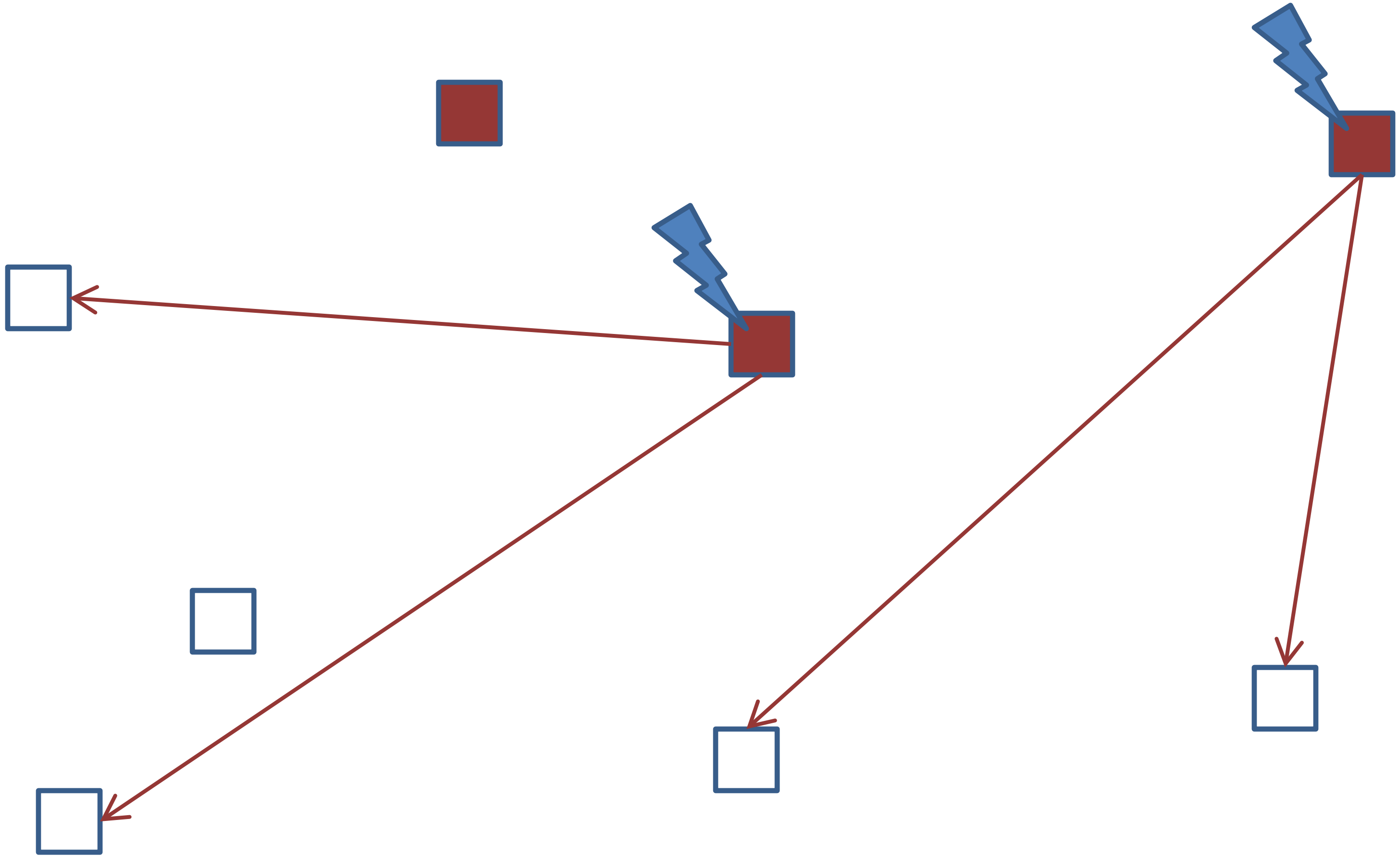
Beyond mesh: Gossip

- Gossip-based dissemination



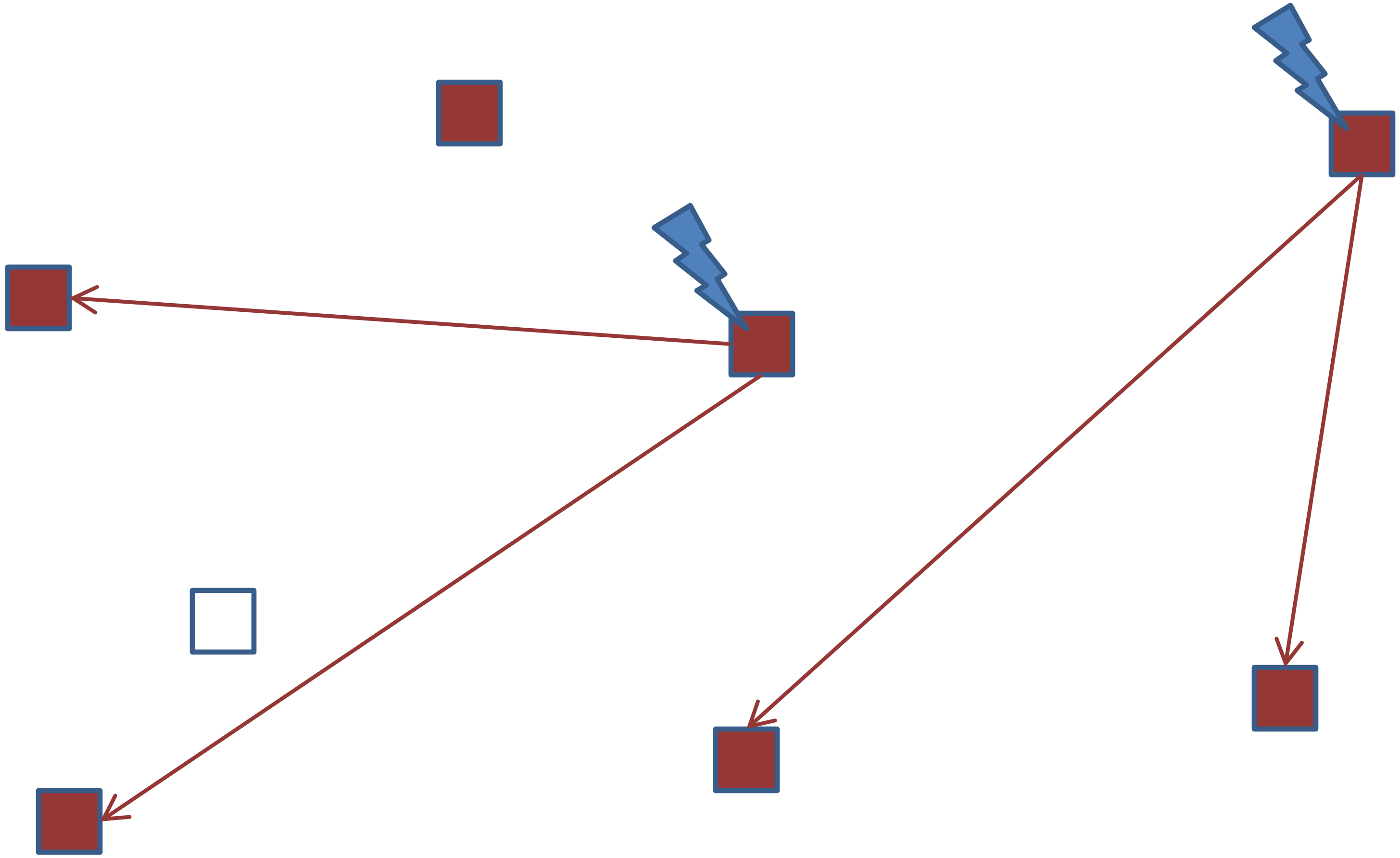
Beyond mesh: Gossip

- Gossip-based dissemination



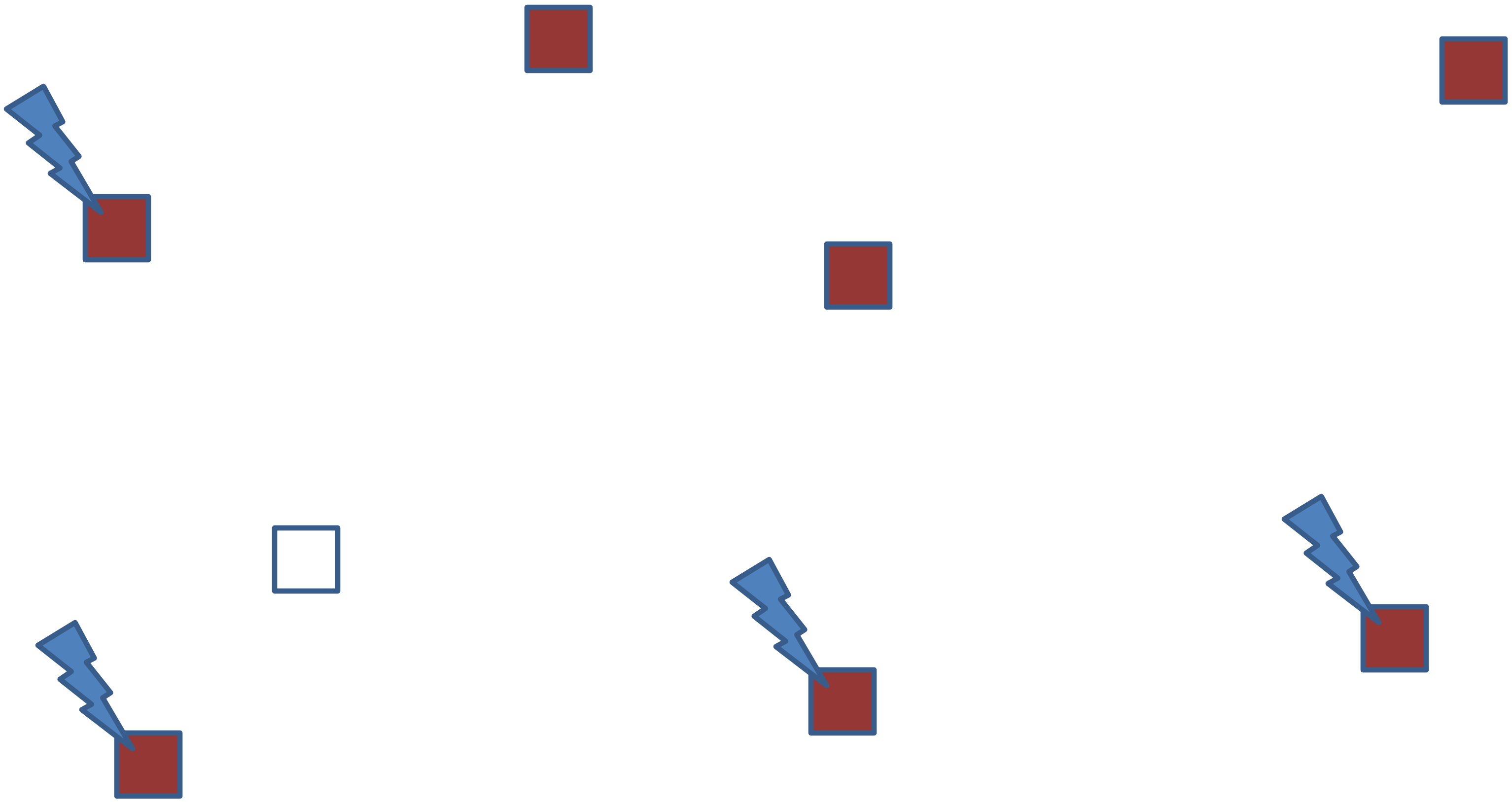
Beyond mesh: Gossip

- Gossip-based dissemination



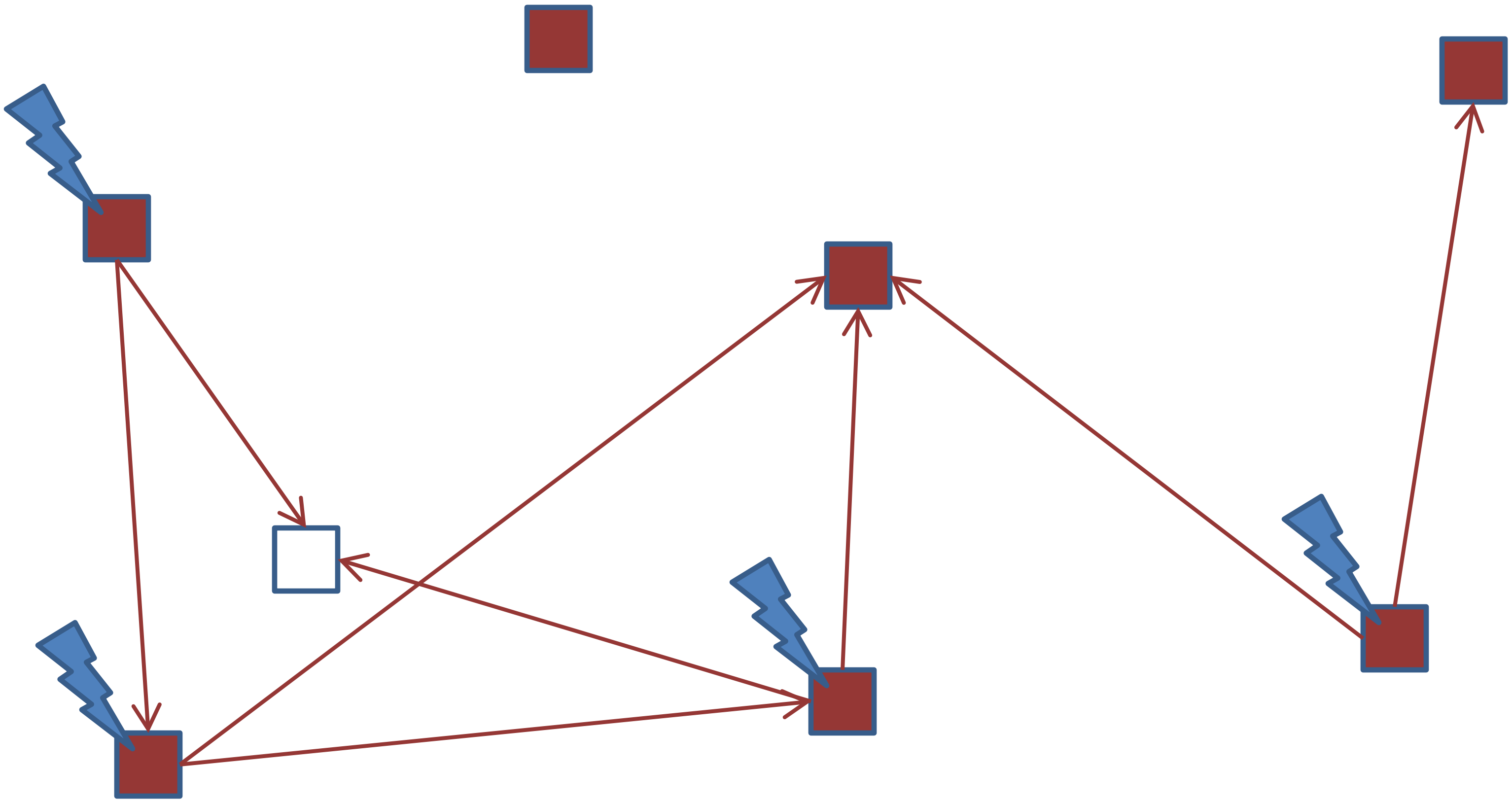
Beyond mesh: Gossip

- Gossip-based dissemination



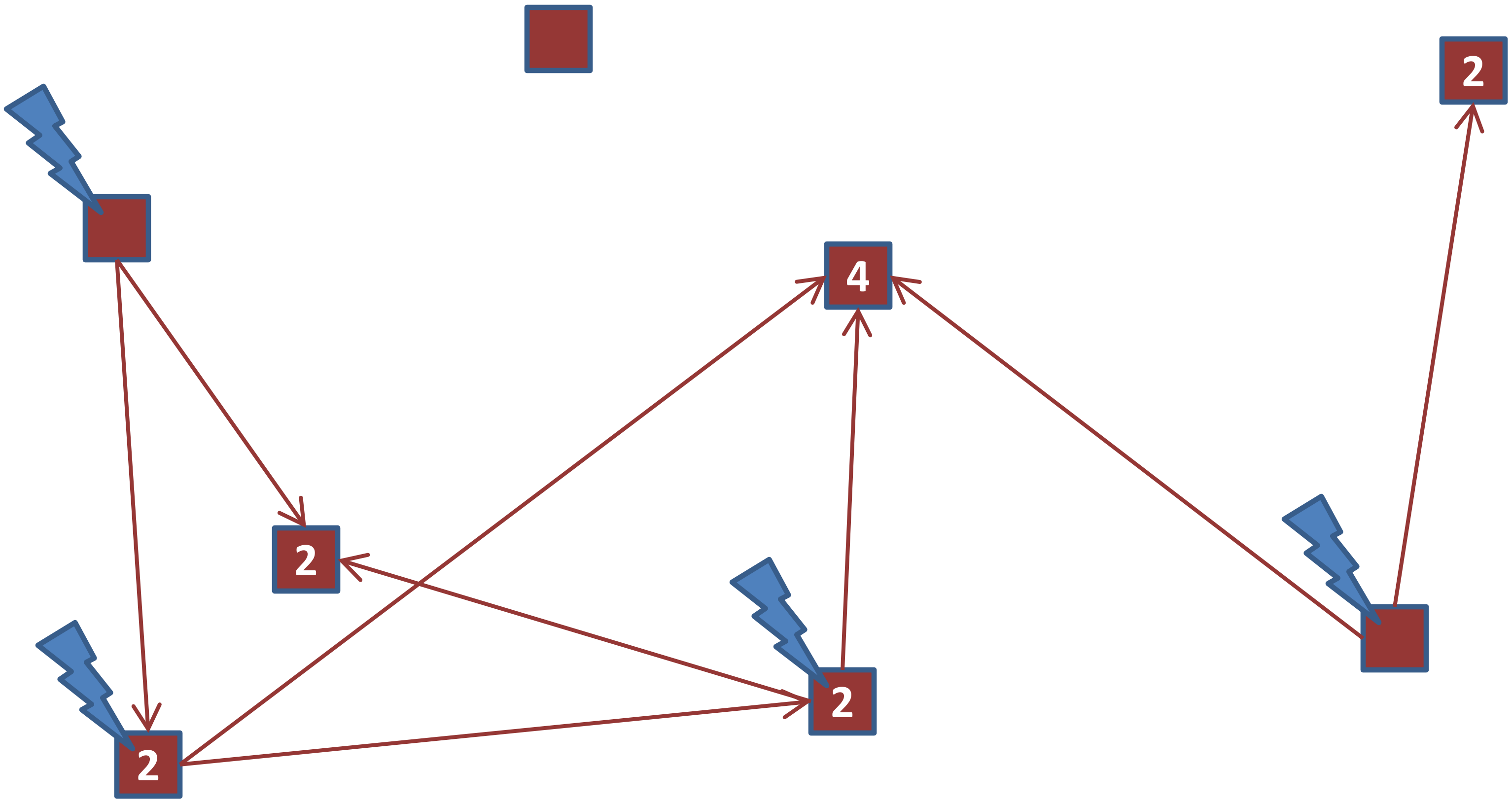
Beyond mesh: Gossip

- Gossip-based dissemination



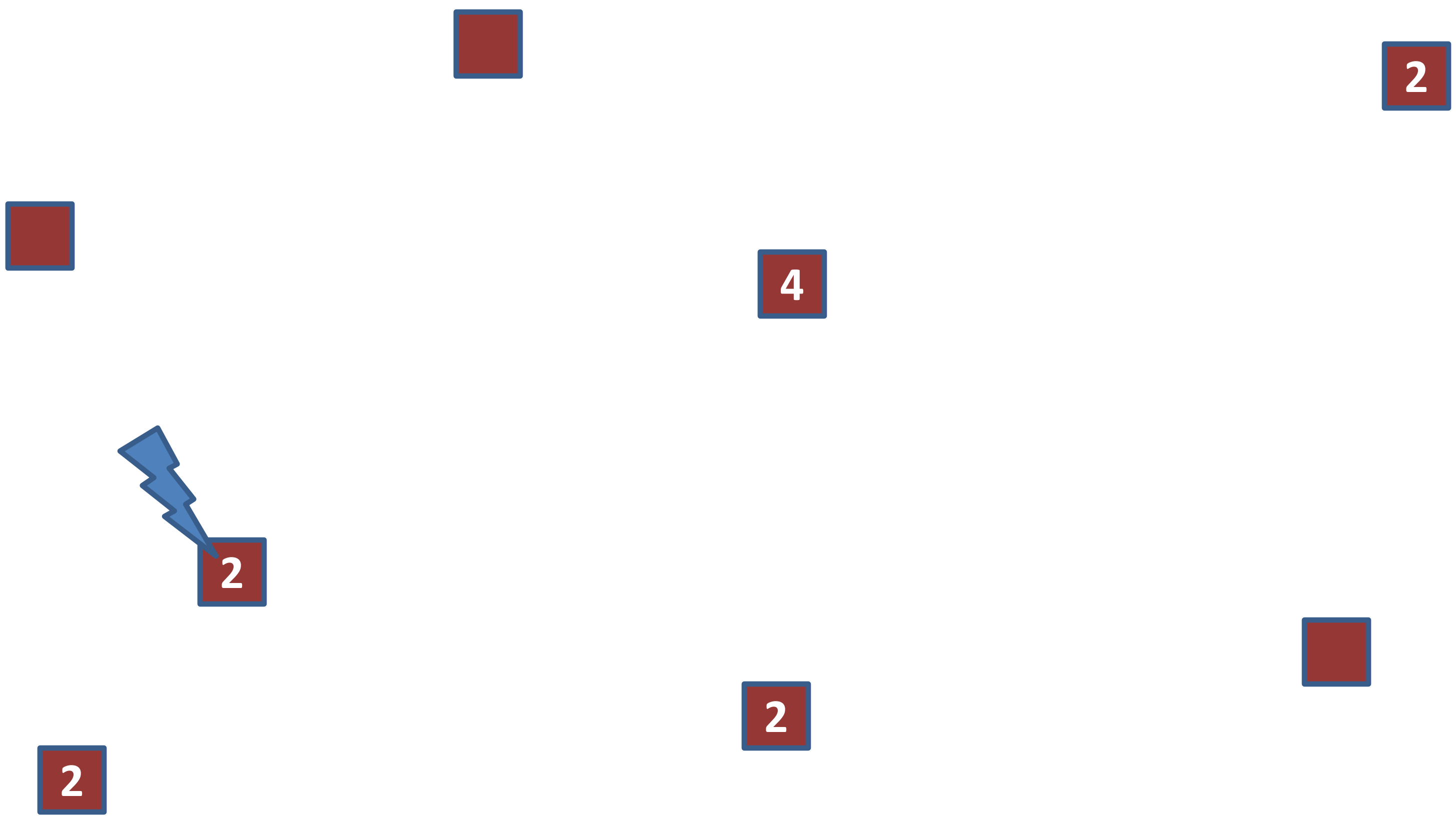
Beyond mesh: Gossip

- Gossip-based dissemination



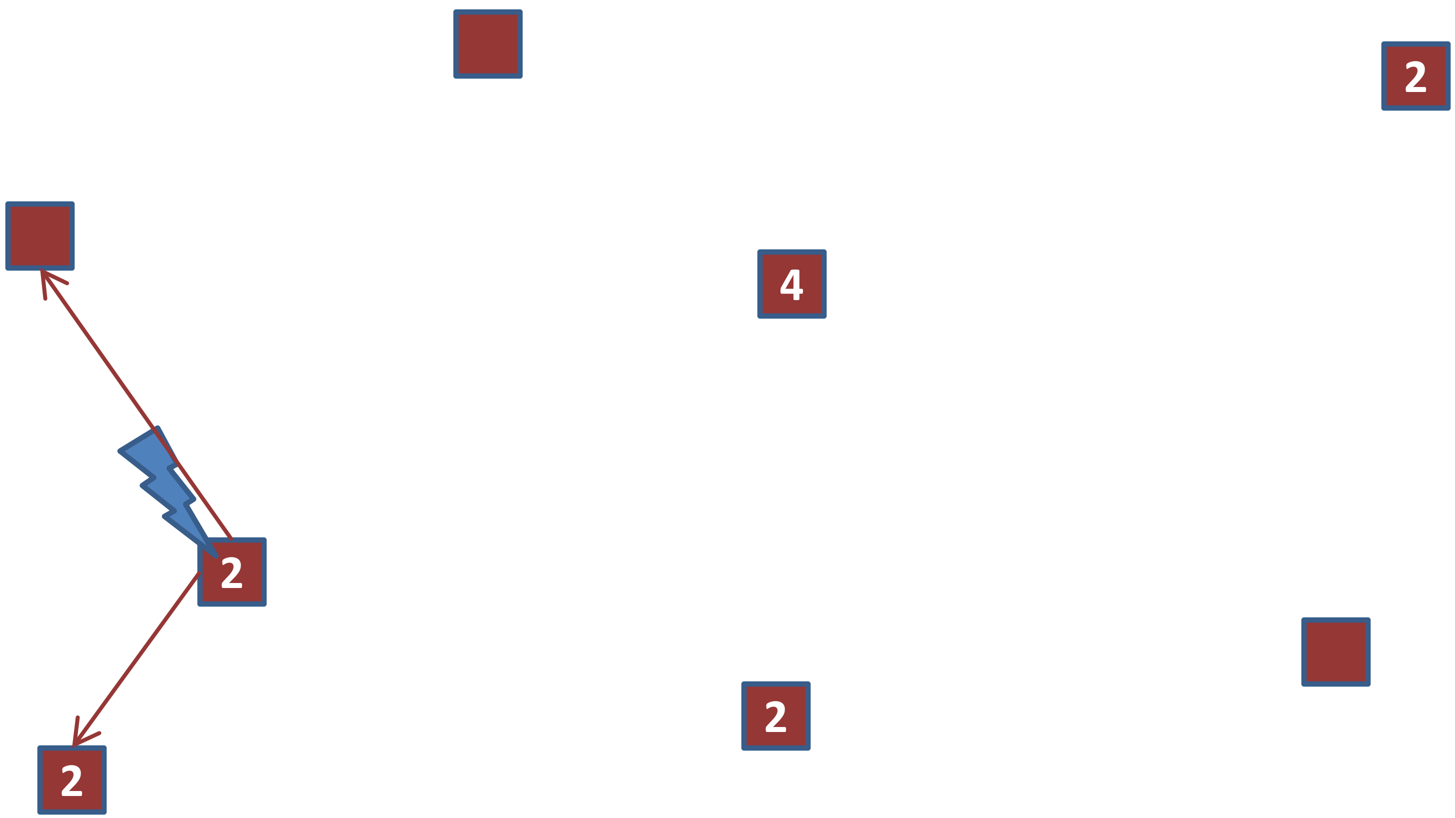
Beyond mesh: Gossip

- Gossip-based dissemination



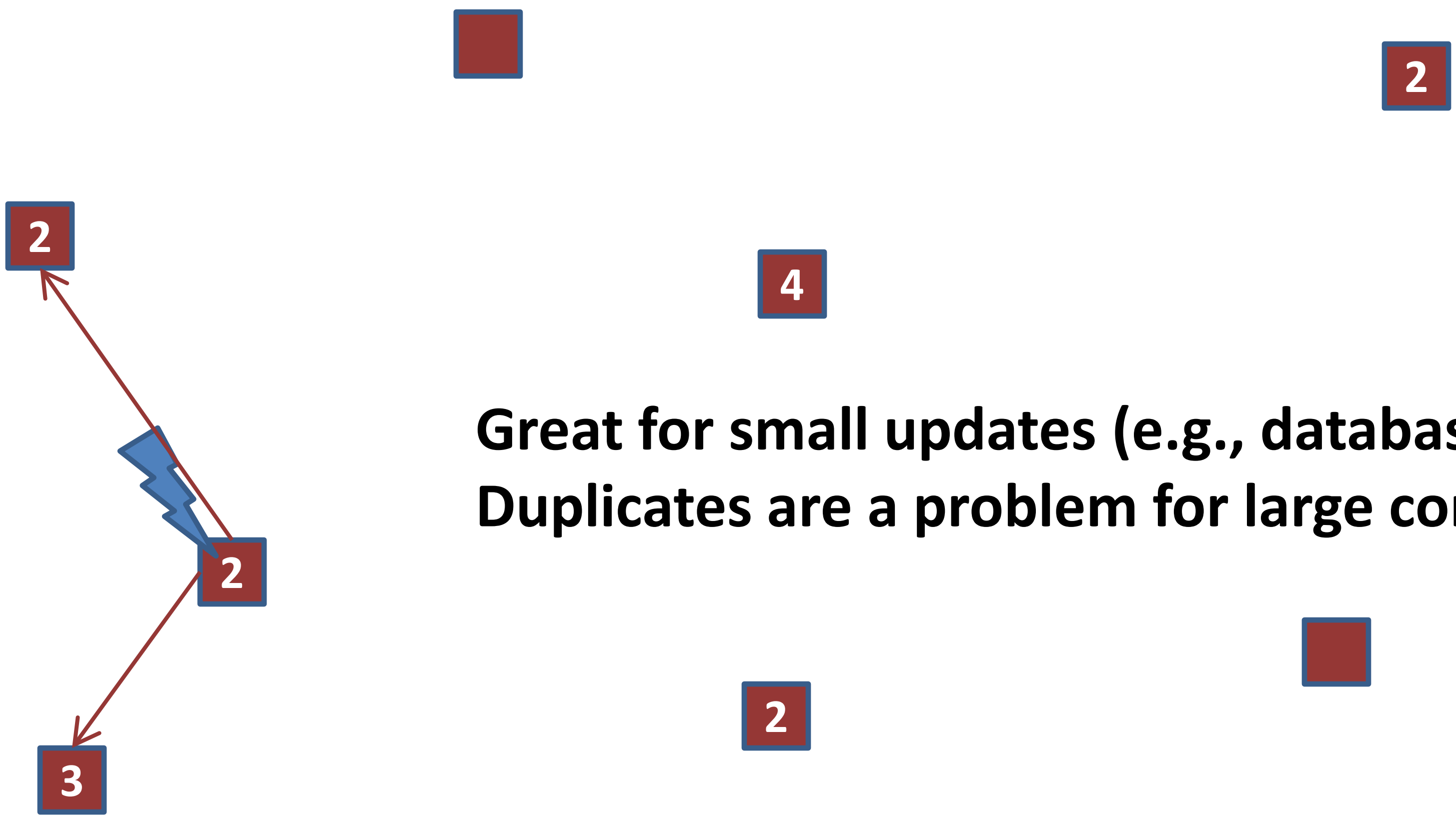
Beyond mesh: Gossip

- Gossip-based dissemination



Beyond mesh: Gossip

- Gossip-based dissemination



Great for small updates (e.g., databases)
Duplicates are a problem for large content…

Gossip for live streaming

1. Gossip content location
   – Propose chunk ids
     – to fanout partners
     – every gossip period

1. Request (chunk ids)

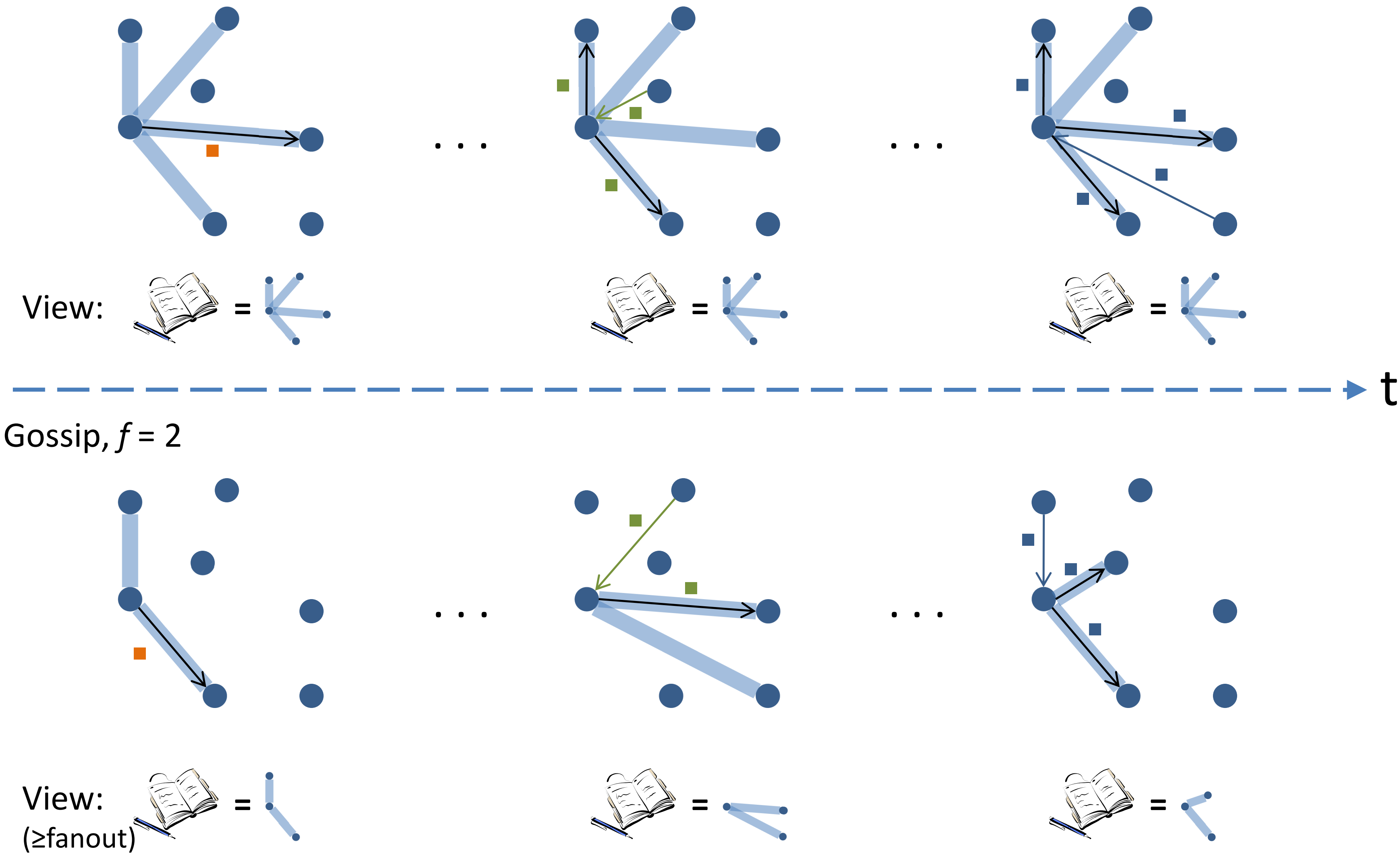
2. Serve chunks (payload)

Fanout modulates contribution of nodes

Mesh vs Gossip

Peering degree = |view| = 4

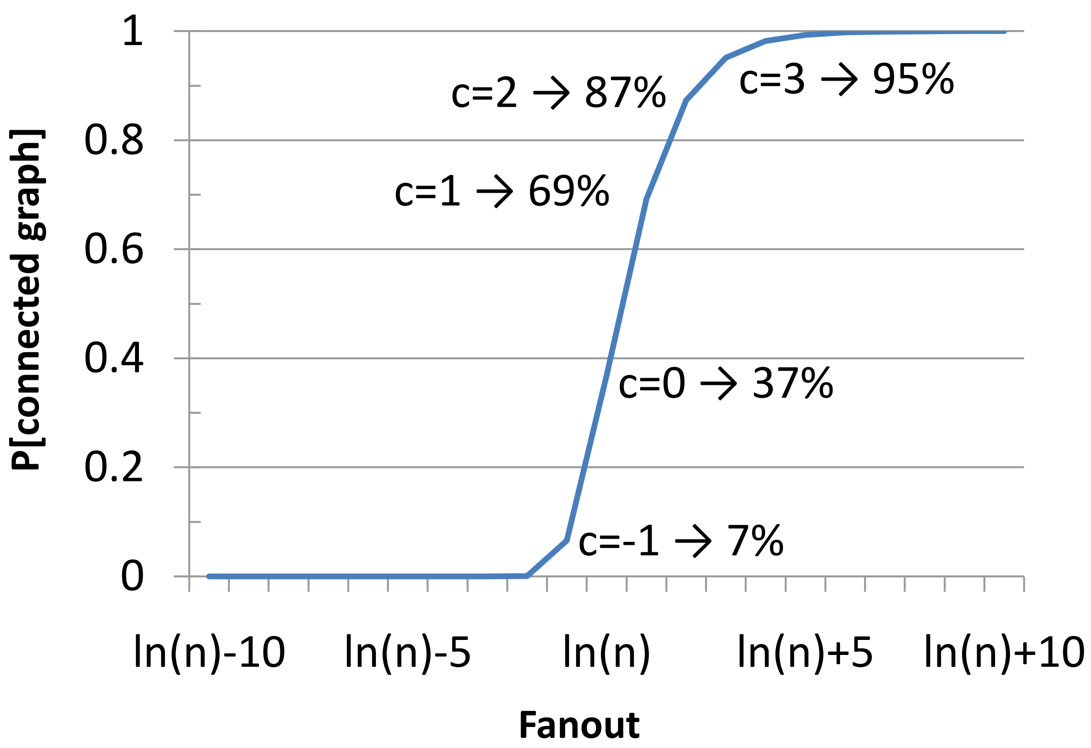
BitTorrent default is 50 ($e^{50} = 5.18 \cdot 10^{21}$)



Gossip – Theory

1. **fanout = ln(*n*) + *c***

   P[connected graph] goes to exp(-exp(-*c*))



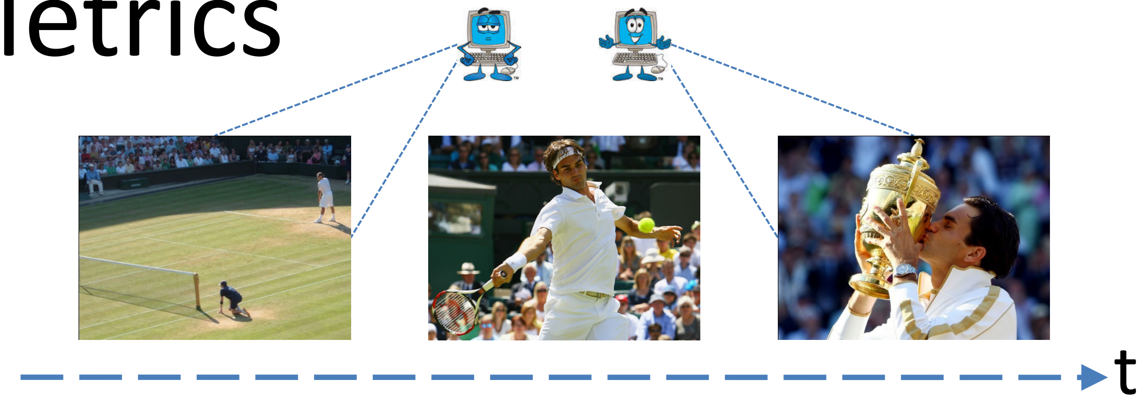
2. Holds as long as the fanout is ln(*n*) + *c* **on average**

Experimental Setup

		Grid'5000	PlanetLab
Environment	Nodes	200 (40*5)	230-300
Environment	BW cap	Token bucket (200KB)	Throttling
Environment	Transport layer	UDP + losses (1-5%)	UDP
Environment	Stream rate *s*	680 kbps	551 kbps
	FEC	5%	10%
	Stream (incl. FEC)	714 kbps	600 kbps
Gossip	T_g (gossip period)	200 ms	200-500 ms
Gossip	fanout (*f*)	8	7-8
Gossip	source's fanout	5	7
Gossip	Retransmission	ARQ/Claim	ARQ
Gossip	Membership	RPS (Cyclon) and full membership	

Evaluation Metrics

- **Stream lag**



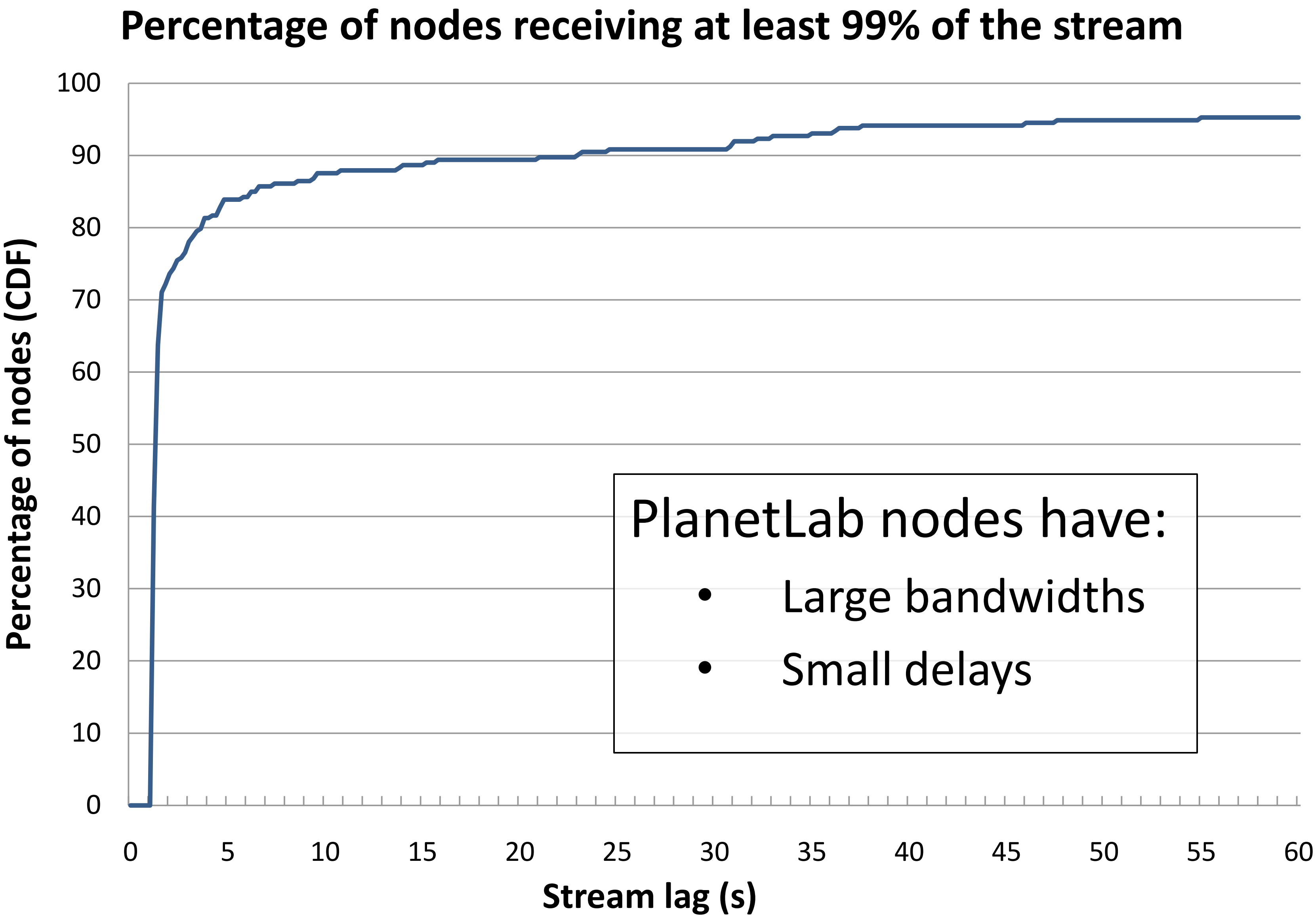
  – Time difference between creation at the source and delivery to the clients' player

- **Stream quality**

  – Maximum 1% jitter means at least 99% of the groups are complete
    - Incomplete groups does not mean "blank"

Gossip – Practice

PlanetLab (230)
s = 600 kbps
f = 7



Live Streaming with Gossip

Constrained environment
- Observations
- Gossip++

Heterogeneous environment
- HEAP

Presence of freeriders
- LiFT

Stretching Gossip

… in a constrained environment

Fanout

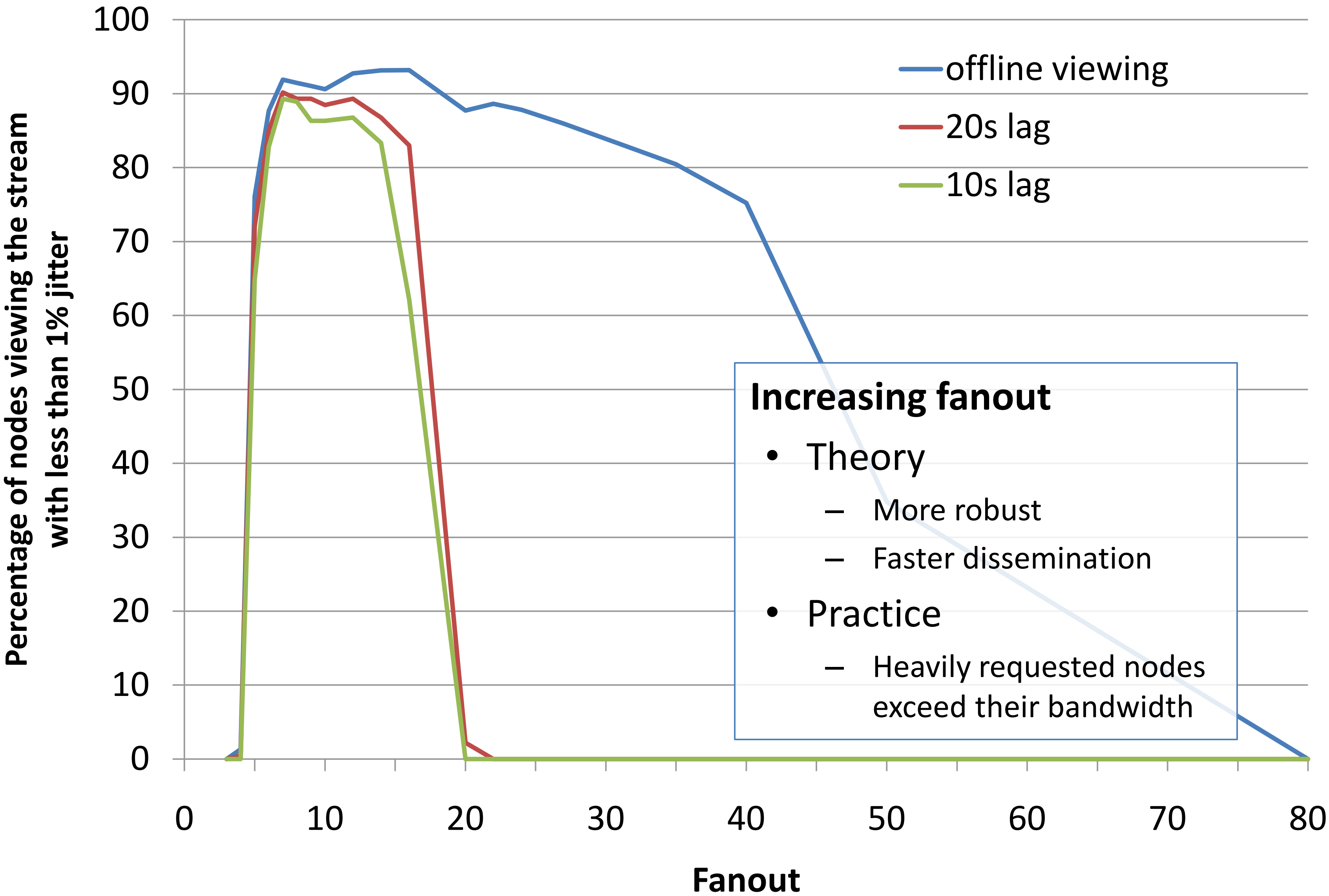
The larger the better?

Proactiveness

How often should a node change its fanout partners?

Optimal fanout

PlanetLab (230)
700 kbps cap
s = 600 kbps

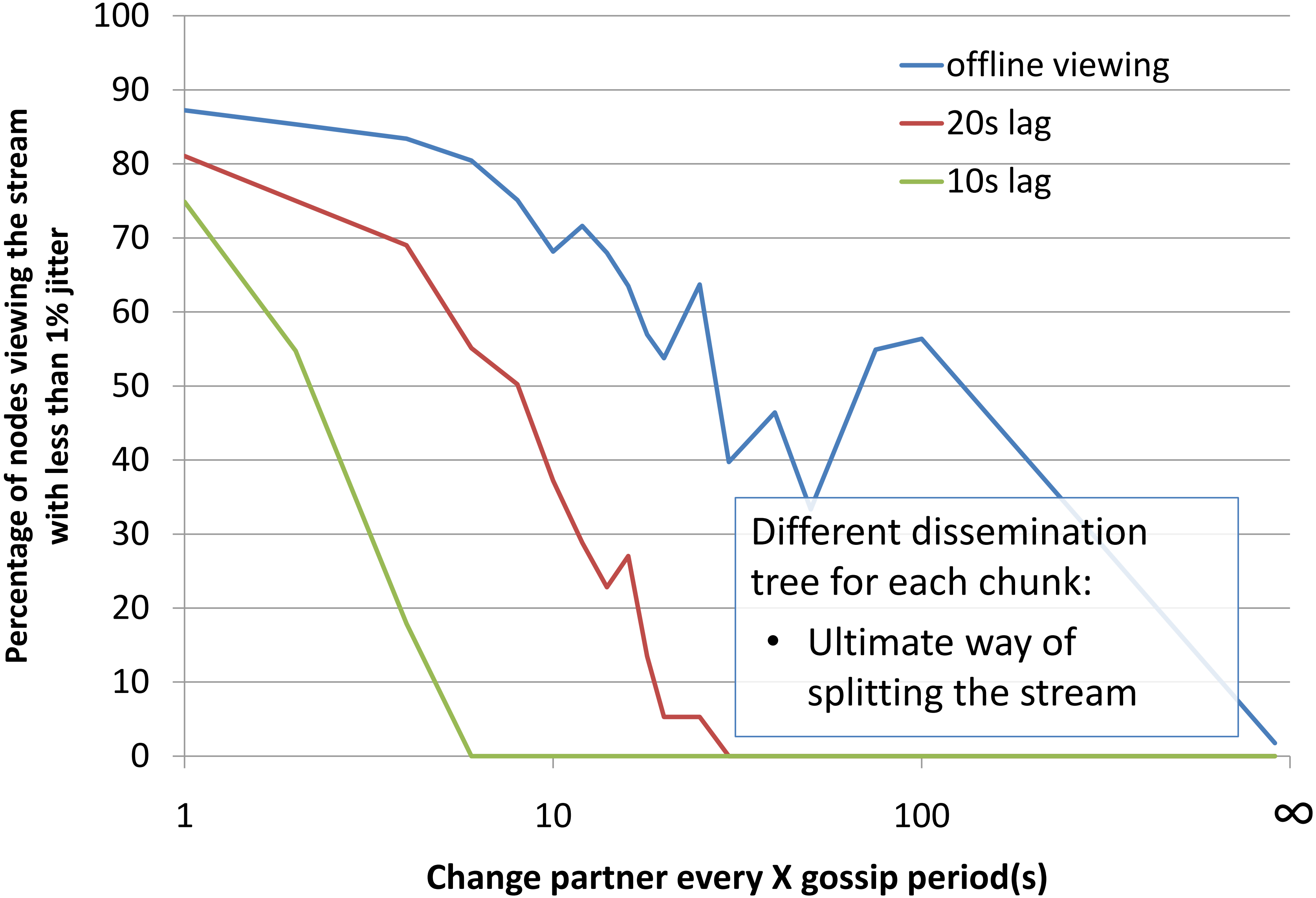


Increasing fanout

- Theory
  - More robust
  - Faster dissemination
- Practice
  - Heavily requested nodes exceed their bandwidth

Optimal proactiveness

PlanetLab (230)
700 kbps cap
s = 600 kbps
f = 7



Different dissemination tree for each chunk:

- Ultimate way of splitting the stream

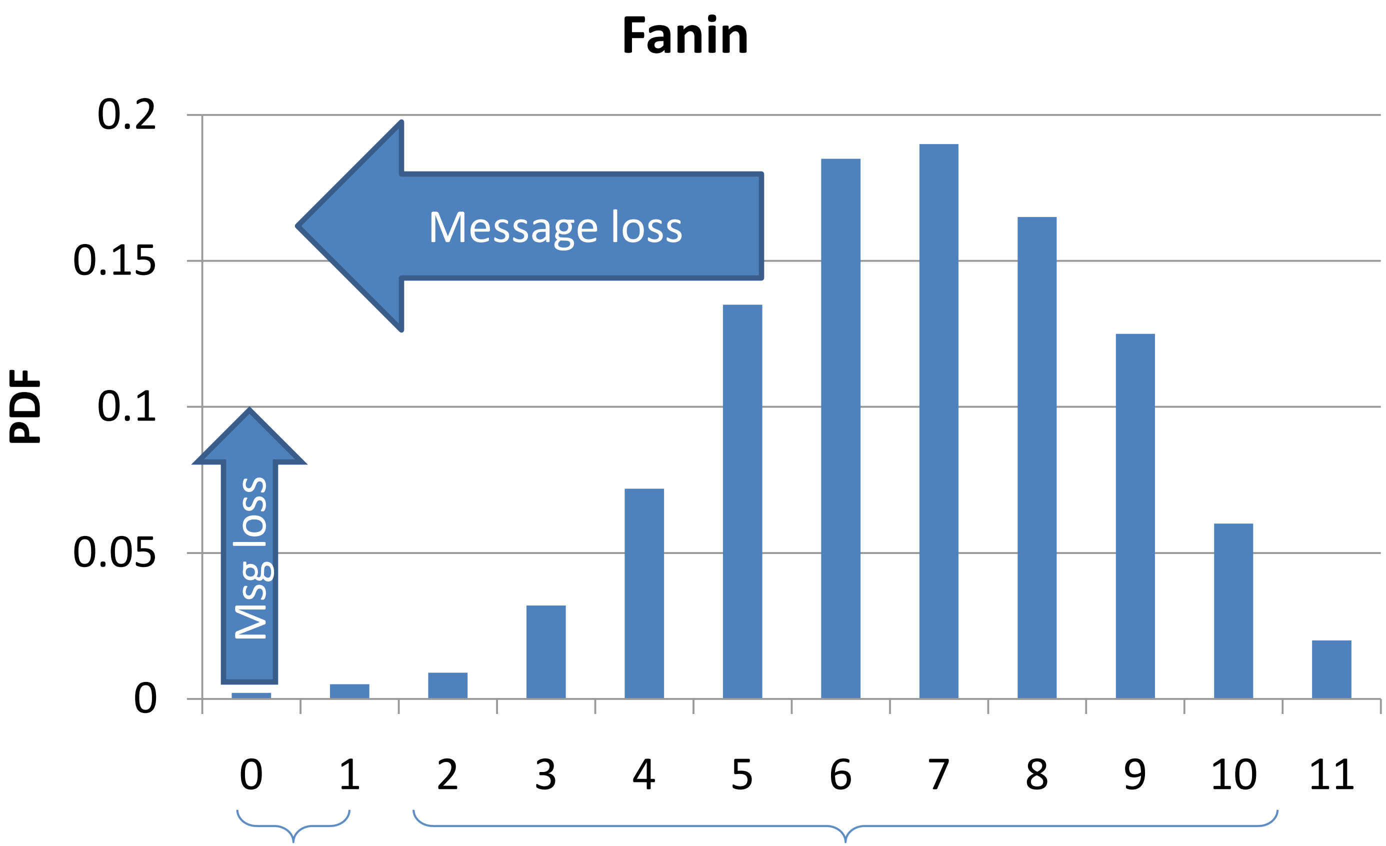
Contributions

Constrained environment	• Observations • **Gossip++**
Heterogeneous environment	• HEAP
Presence of freeriders	• LiFT

Gossip++

Observations:

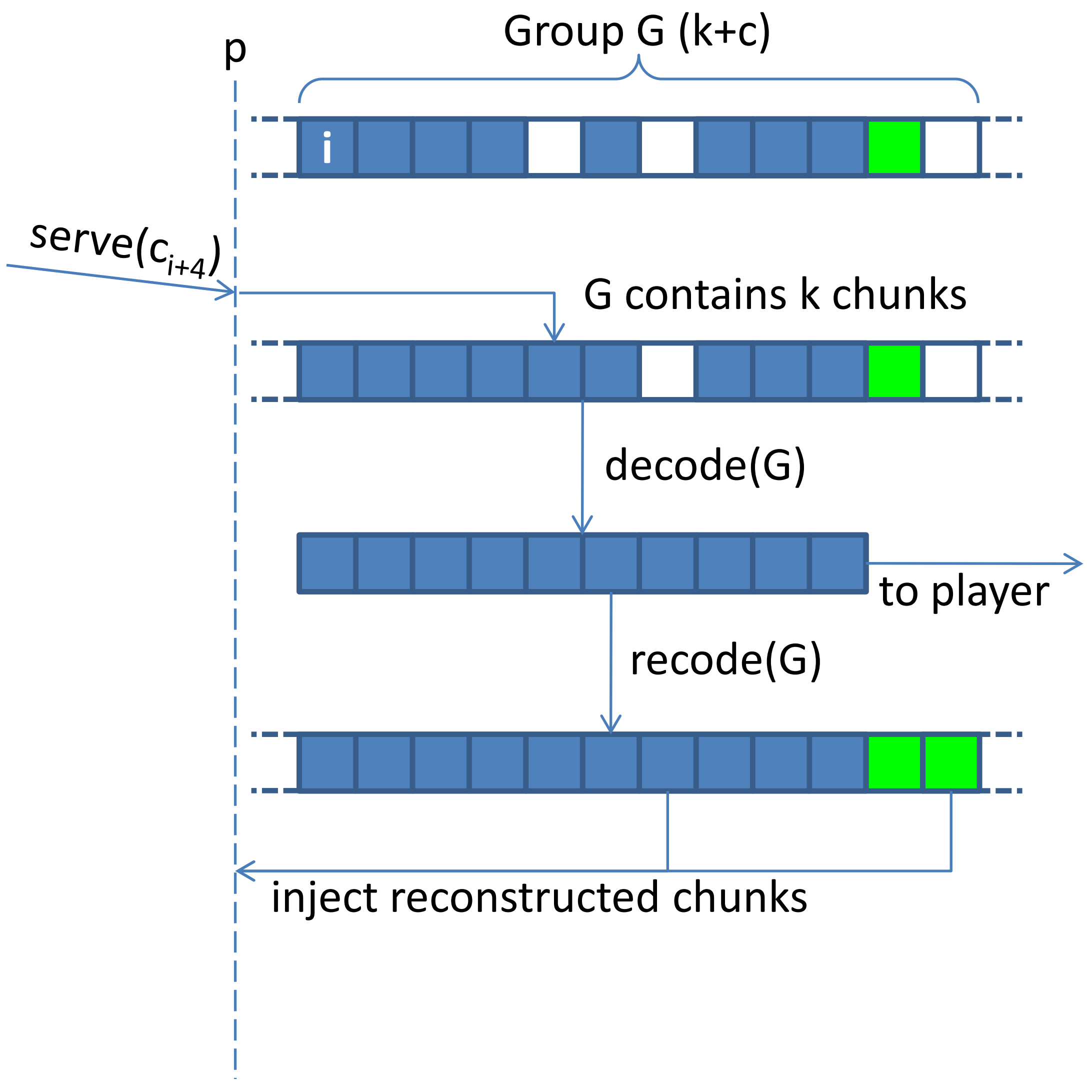
- Fanout has an optimal value/range
- Change partners every gossip period
  - Ultimate way of splitting the stream



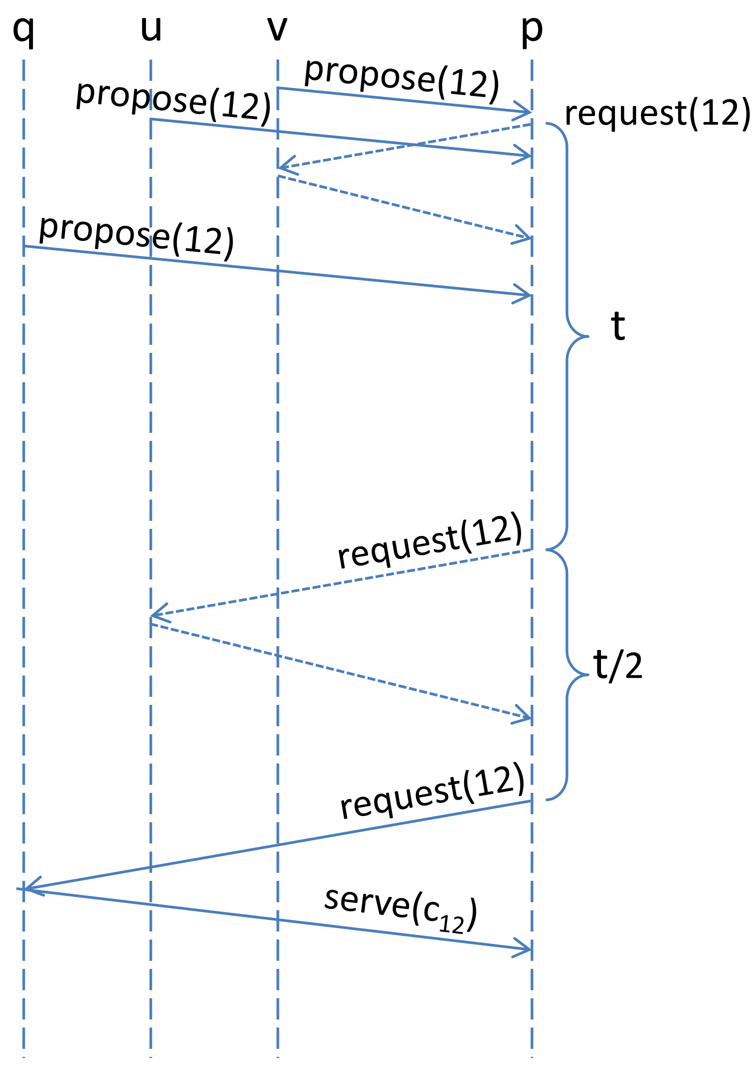
How to receive a chunk that is not even proposed?

How to exploit the many duplicates?

Gossip++: **Codec** & Claim

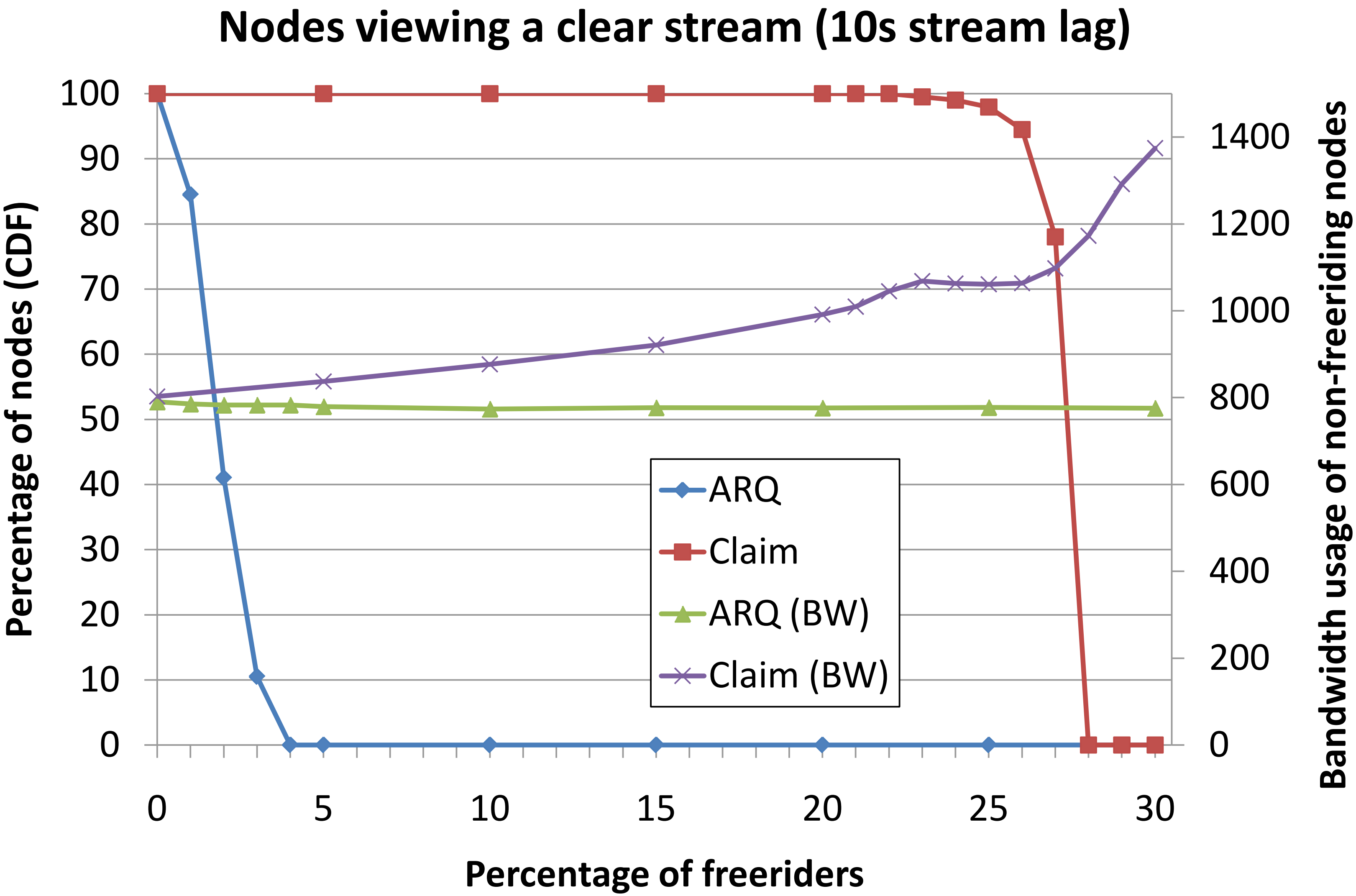


Gossip++: Codec & **Claim**



Gossip++ with freeriders

Grid’5000 (200)
1000 kbps cap
s = 714 kbps (5% FEC)



Contributions

Constrained environment

- Observations
- Gossip++

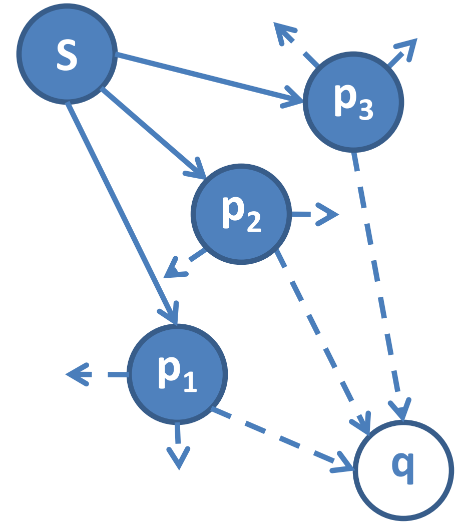
Heterogeneous environment

- HEAP

Presence of freeriders

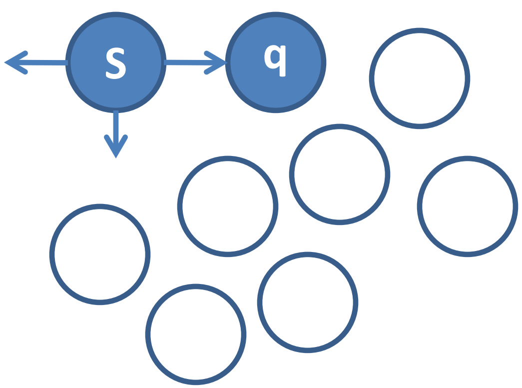
- LiFT

Gossip is load-balancing…



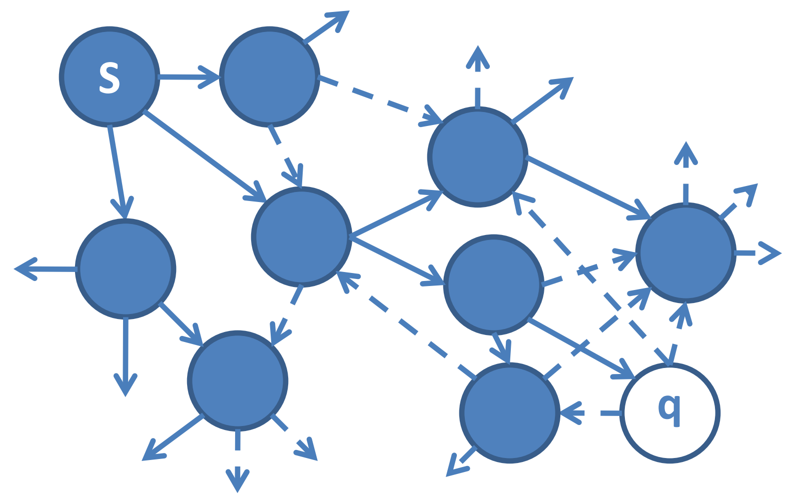
- Proposals arrive randomly
  - Nodes pull from first proposal

- Highly-dynamic



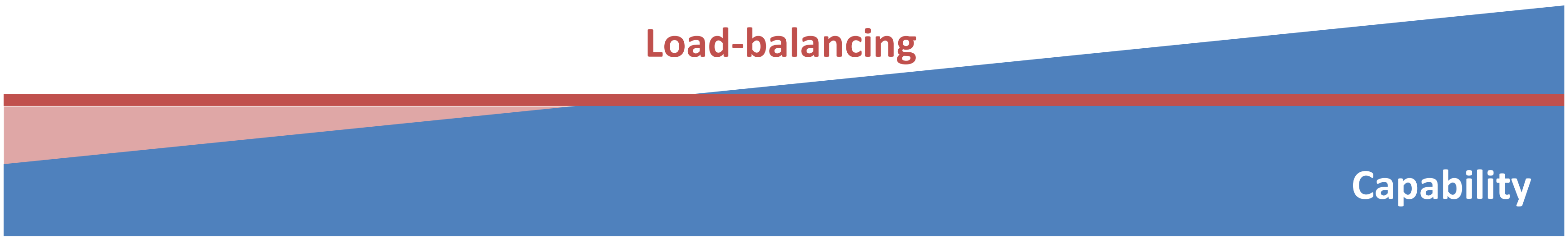
Node q will serve *f* nodes whp

…

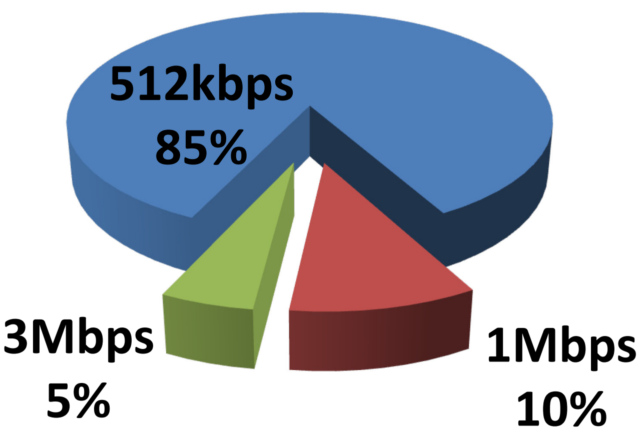


Node q will serve *f* nodes wlp

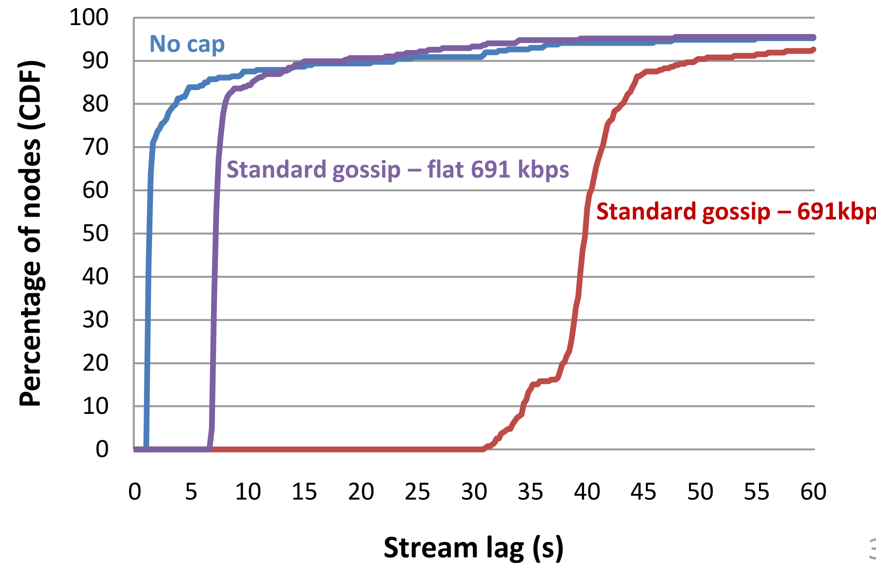
… but the world is heterogeneous!



3 classes (691kbps avg):



Percentage of nodes receiving at least 99% of the stream



How to cope with heterogeneity?

- **Goal:** contribute according to capability

- Propose more = serve more
  – Increase fanout…

vs

… and decrease it too!

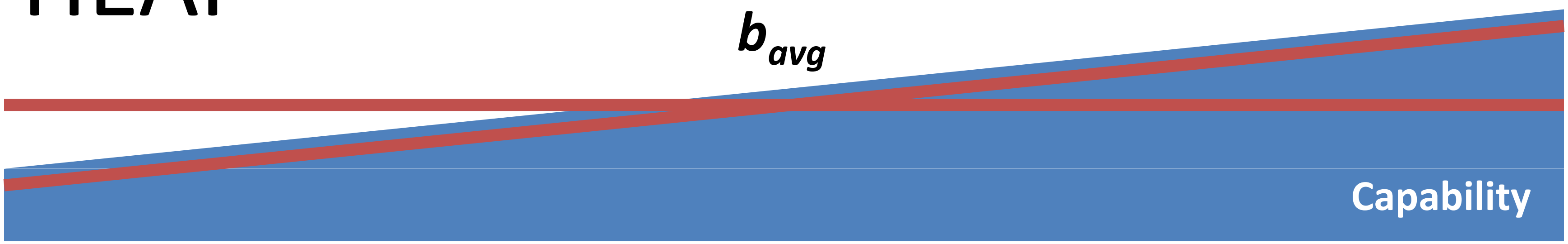
- Such that
  – average fanout (f_{avg}) ≥ initial fanout = ln(n) + c

Heterogeneous Gossip - HEAP

- q and r with bandwidths $b_q > b_r$
  - q should upload b_q / b_r times as much as r
- Who should increase/decrease its contribution?

  … and by how much?
- How to ensure reliability?
  - How to keep f_{avg} constant?

HEAP



- Total/average contribution is equal in both homogeneous and heterogeneous settings

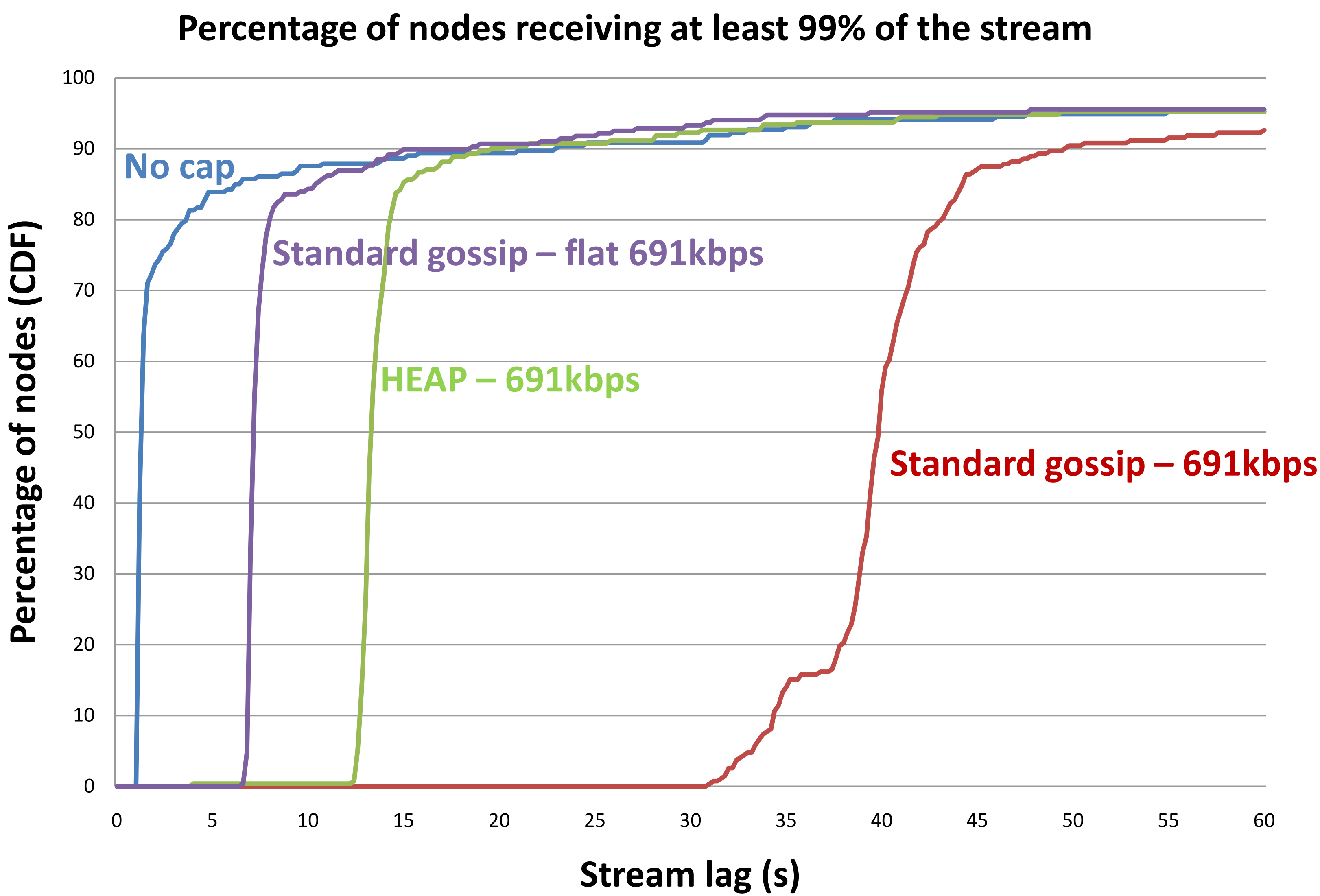
$$f_q = f_{\text{init}} \cdot b_q / \mathbf{b}_{avg}$$

…ensuring the average fanout is constant and equal to $f_{init} = \ln(\boldsymbol{n}) + c$

HEAP

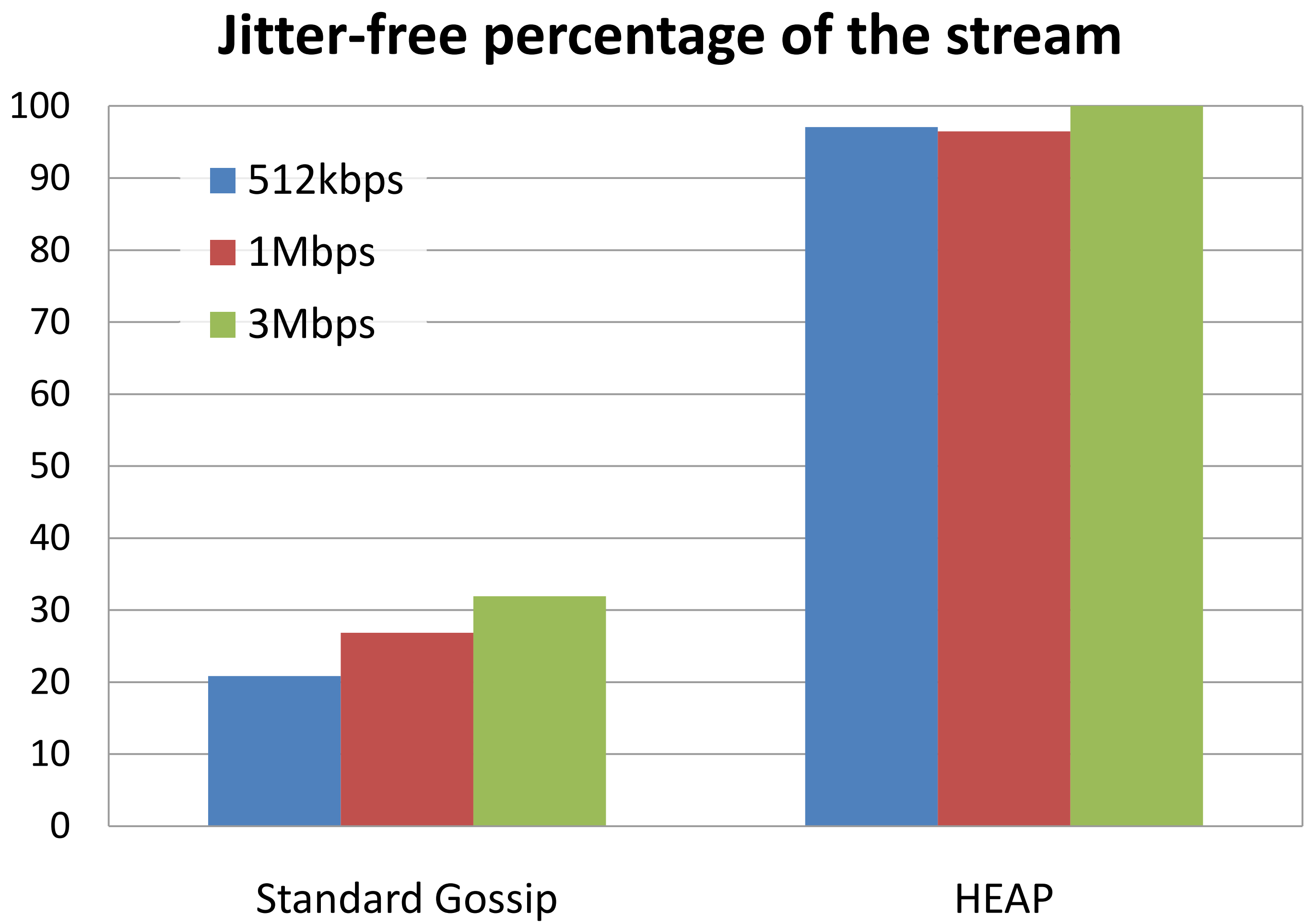
- Get b_{avg} with (gossip) aggregation
  - Advertize own and freshest received capabilities
  - Aggregation follows change in the capabilities
- Get n with (gossip) size estimation
  - Estimation follows change in the system
    - Join/leave
    - Crashes
    - …

Stream lag reduction



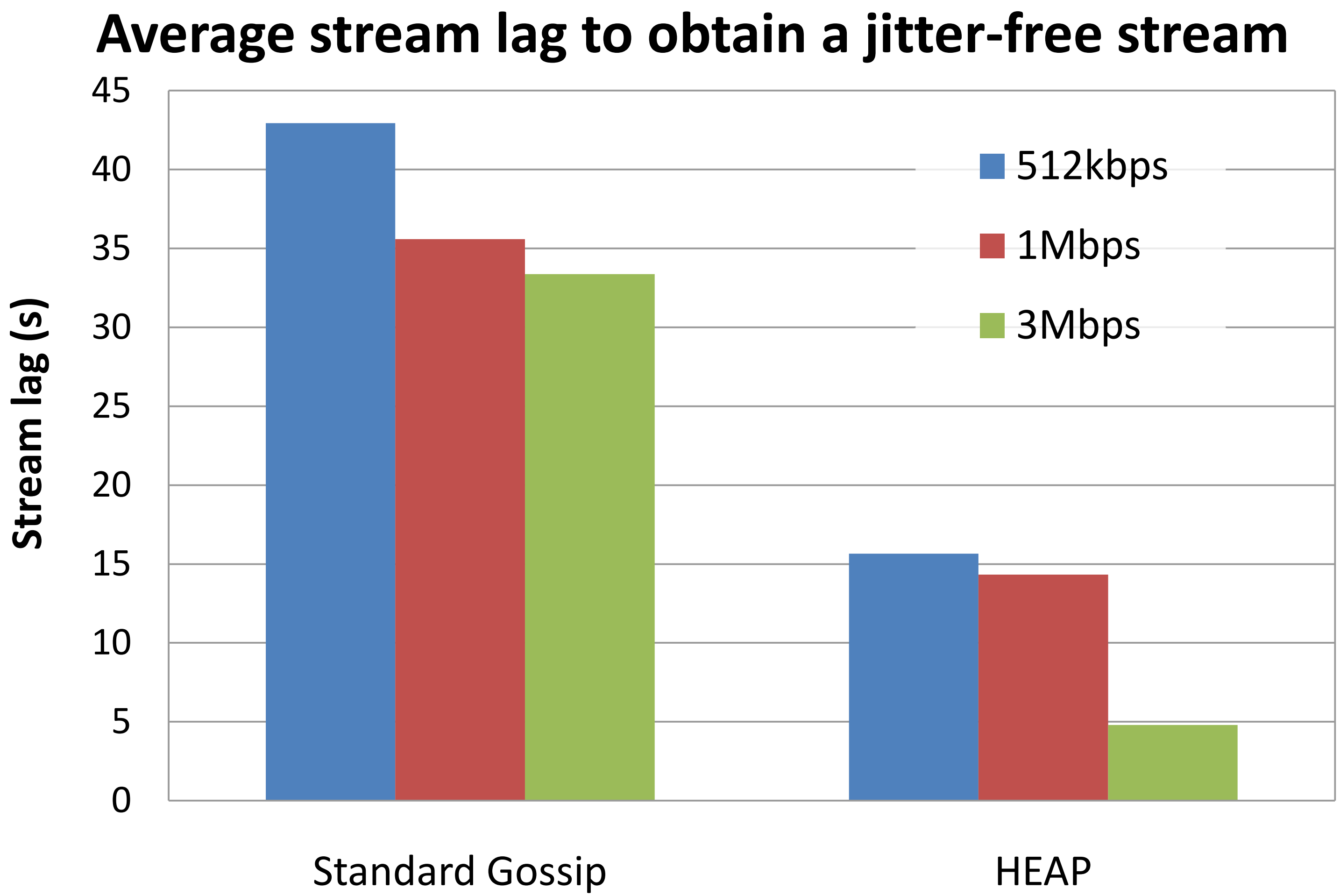
Quality improvement

- Stream lag of 10s

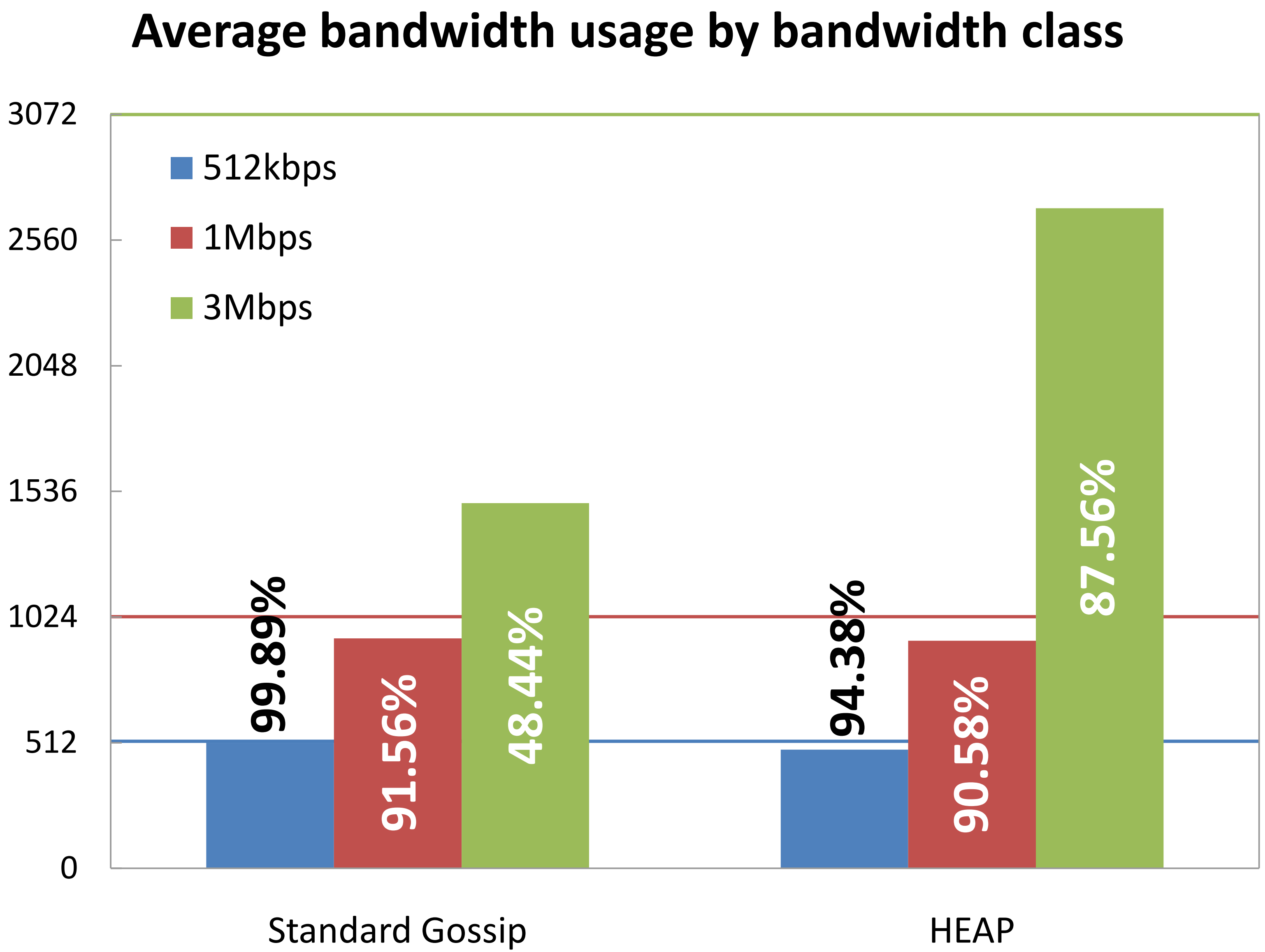


Stream lag

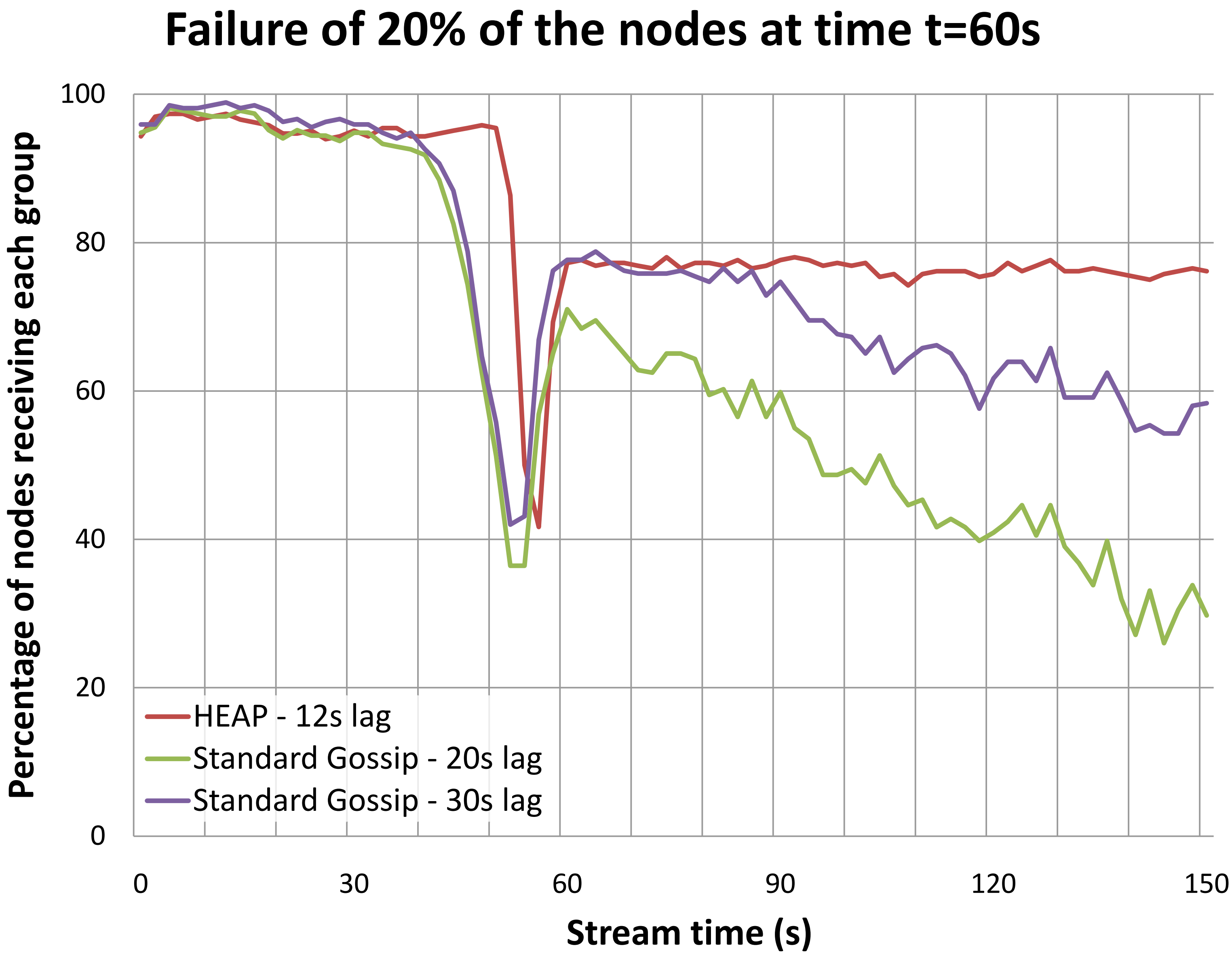
- For those who can have a jitter-free stream



Proportional contribution



20% nodes crashing



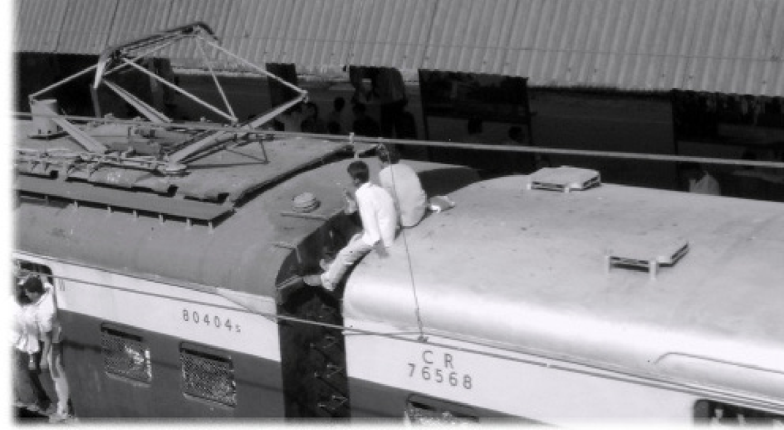
Contributions

Constrained environment	• Observations • Gossip++
Heterogeneous environment	• HEAP
Presence of freeriders	• LiFT

Freeriders

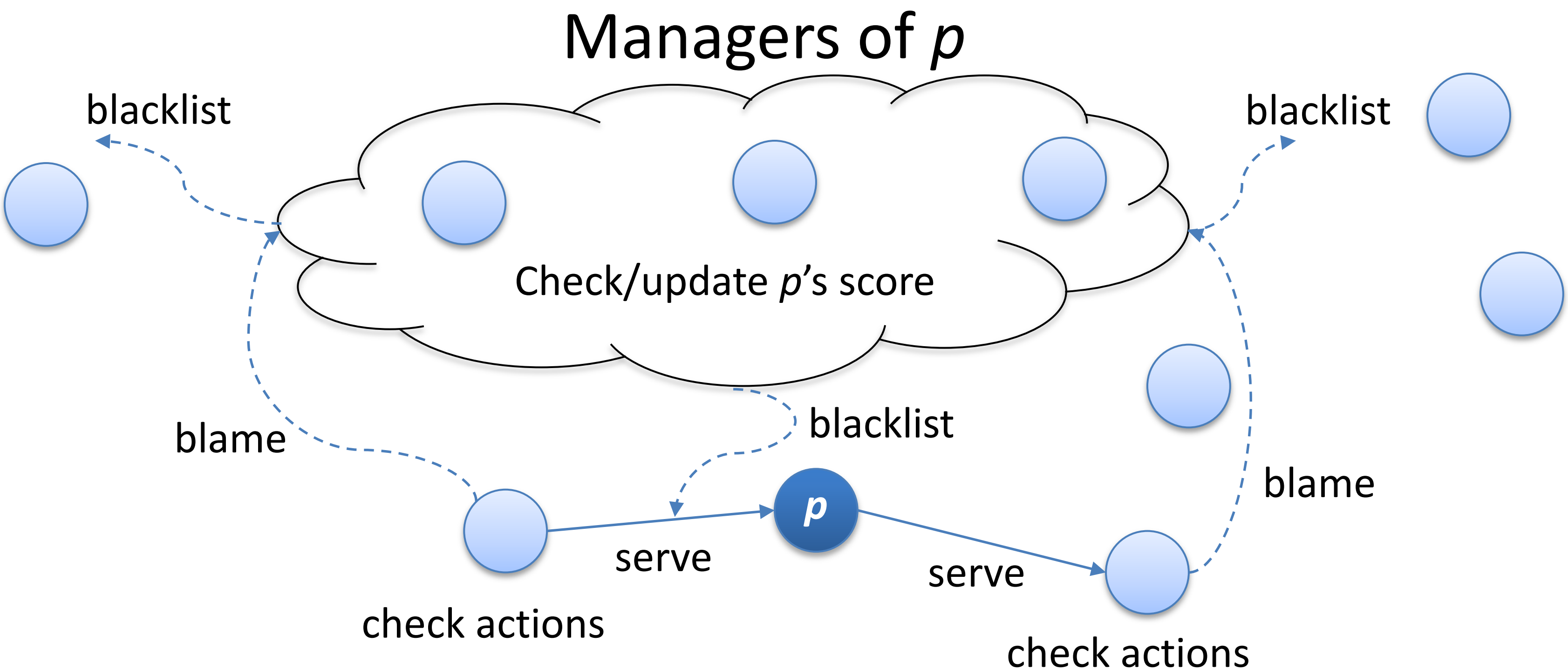
- Selfish participants
  - Maximize benefit
  - Minimize contribution
  - Avoid to be detected
  - May collude
- Attacks on Tit-for-Tat protocols
  - Opportunistic unchoking
    - Can get without giving anything in return
      - (e.g., Large-view exploit, etc)

Freeriding Gossip



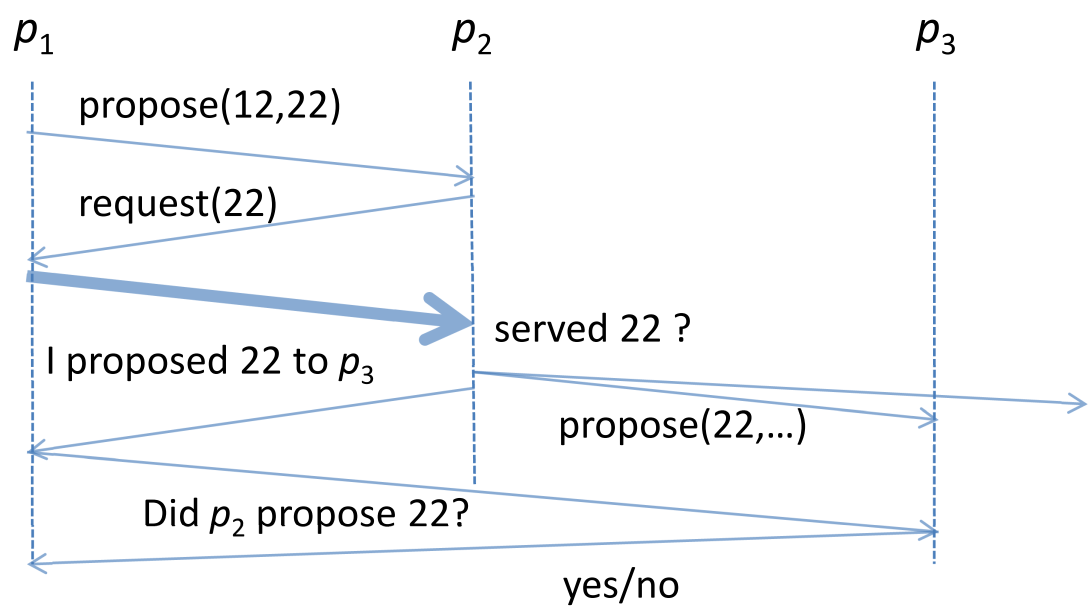
- Reduce fanout
- Propose less chunks than received
- Serve less chunks than requested
- Bias partner selection (colluders)

Architecture



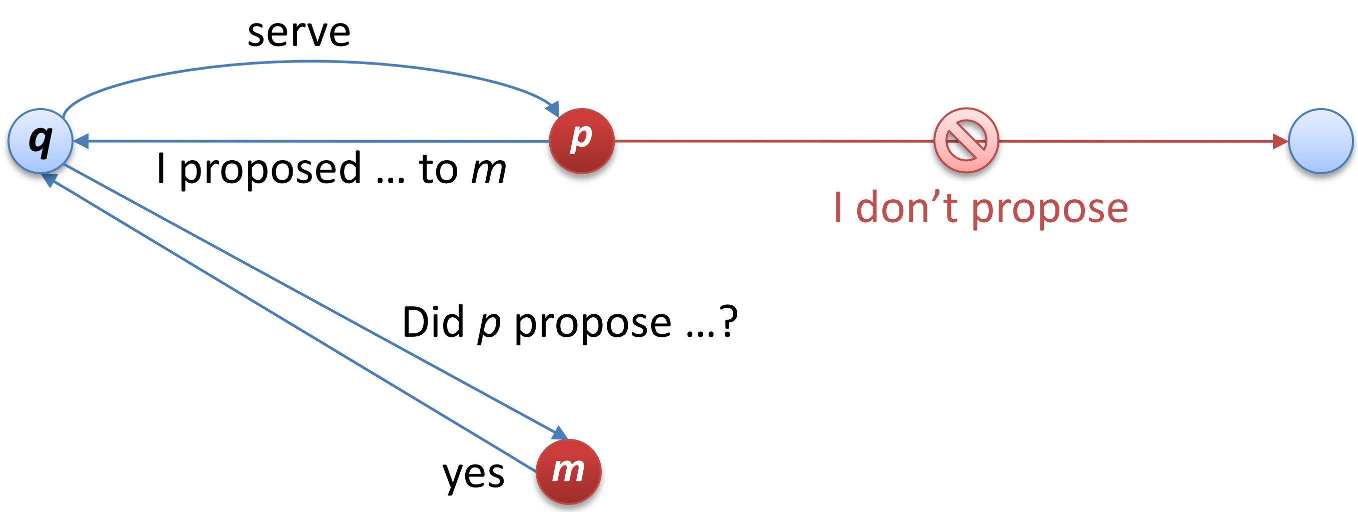
LiFT: direct verifications

- Direct check
  - Requested chunks are served
- Cross-checking
  - Served chunks are proposed



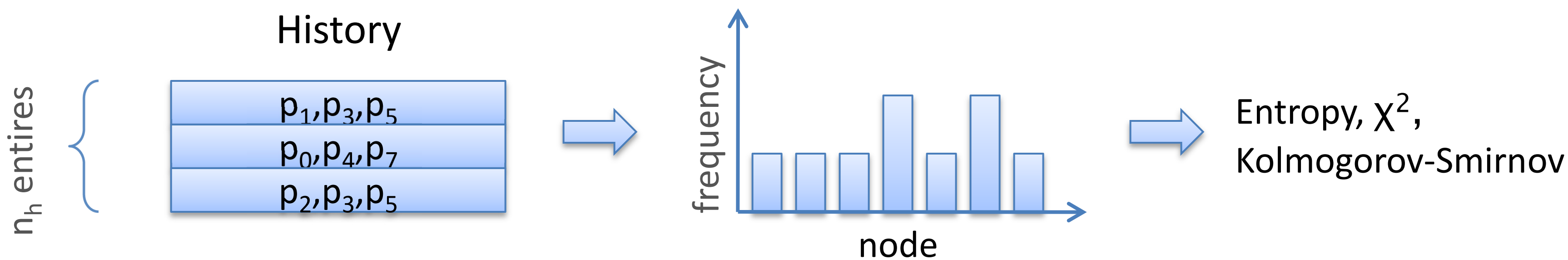
Attacking the direct verifications

- Colluder-in-the-middle



LiFT: *a posteriori* verifications

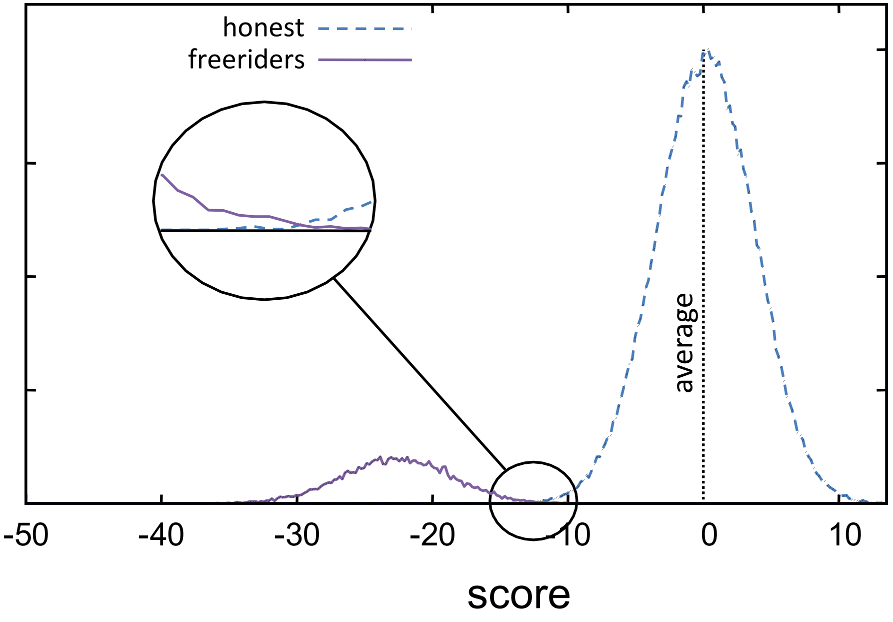
- Statistical check
  - Partners chosen at random



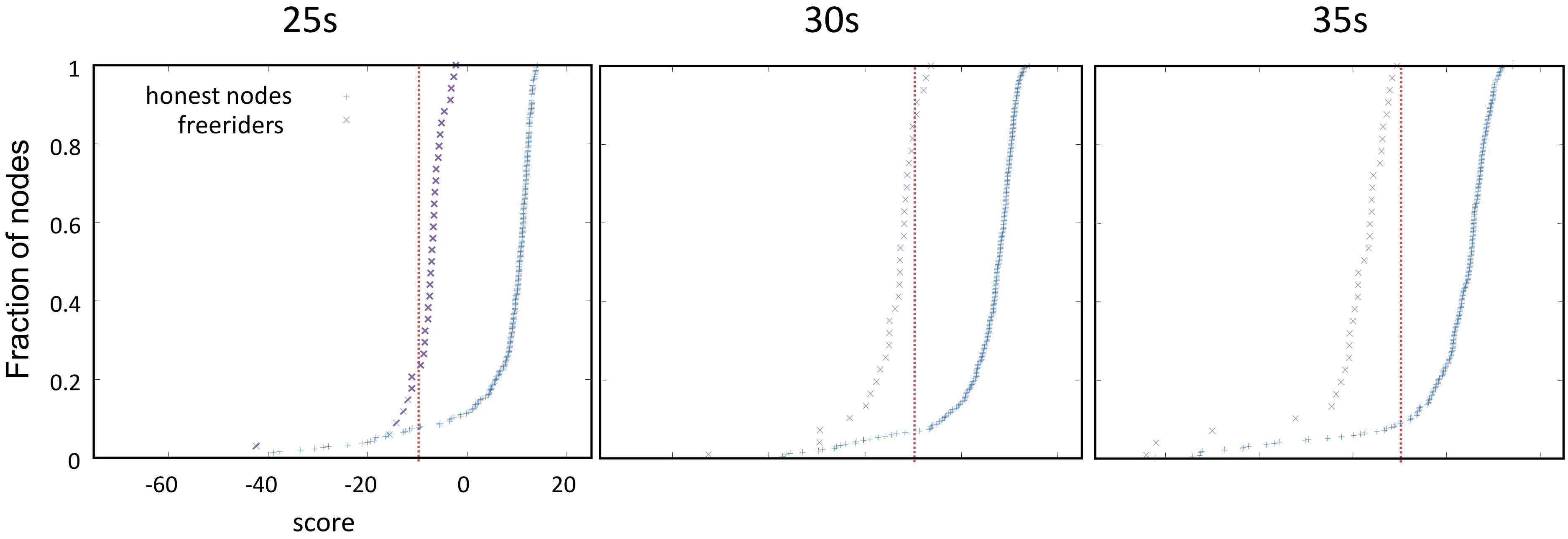
- Verification on both fanin/fanout histories

LiFT: scores

- Absolute scores
  - Need to be compensated
  - Message loss = wrongful blames



LiFT: evaluation



	Cross-checking and blaming overhead		
p_{cc}	0	0.5	1
674 kbps	1.07%	4.53%	8.01%
1082kbps	0.69%	3.51%	5.04%
2036 kbps	0.38%	1.69%	2.76%

Summary of results

Live Streaming with Gossip? **Yes**

Constrained environment

Observations

- Optimal fanout value/range
- Optimal proactiveness

Gossip++

- Codec + Claim
- Tolerance to freeriders

Heterogeneous environment

HEAP

- Bandwidth measurement
- Fanout adaptation

- Preserved simplicity
- Preserved reliability
- Preserved Efficiency

Presence of freeriders

LiFT

- Lightweight
- Simple
- Secures asymmetric exchanges

Open problems

Gossip++
- Chunk priorities
- Dissemination tree prediction

HEAP
- Do more with b_{avg}
- Different adaptation
  - p_{acc}

LiFT
- Punishing vs revoking
- Heterogeneous environment
- Oracle & spies

Thank you