Heterogeneous Gossip

Davide Frey[2], Rachid Guerraoui[1], Anne-Marie Kermarrec[2], Boris Koldehofe[3], Martin Mogensen[4], Maxime Monod[1⋆], and Vivien Quéma[5]

[1] Ecole Polytechnique Fédérale de Lausanne
[2] INRIA Rennes-Bretagne Atlantique
[3] University of Stuttgart
[4] University of Aarhus
[5] CNRS

Abstract. Gossip-based information dissemination protocols are considered easy to deploy, scalable and resilient to network dynamics. Load-balancing is inherent in these protocols as the dissemination work is evenly spread among all nodes. Yet, large-scale distributed systems are usually heterogeneous with respect to network capabilities such as bandwidth. In practice, a blind load-balancing strategy might significantly hamper the performance of the gossip dissemination.
This paper presents HEAP, *HEterogeneity-Aware gossip Protocol*, where nodes dynamically adapt their contribution to the gossip dissemination according to their bandwidth capabilities. Using a continuous, itself gossip-based, approximation of relative bandwidth capabilities, HEAP dynamically leverages the most capable nodes by increasing their fanout, while decreasing by the same proportion that of less capable nodes. HEAP preserves the simple and proactive (churn adaptation) nature of gossip, while significantly improving its effectiveness. We extensively evaluate HEAP in the context of a video streaming application on a testbed of 270 PlanetLab nodes. Our results show that HEAP significantly improves the quality of the streaming over standard homogeneous gossip protocols, especially when the stream rate is close to the average available bandwidth.

1 Introduction

Gossip protocols are especially appealing in the context of large-scale dynamic systems. Initially introduced for maintaining replicated database systems [6], they are particularly useful for effective dissemination [1].

In the context of decentralized live streaming, for instance, gossip protocols [3, 18, 19] constitute an interesting alternative to classical mesh-based techniques for large-scale dynamic systems. While efficient under steady state, mesh-based solutions require sophisticated and sometimes expensive repair schemes to maintain possibly several dissemination paths in case of churn [17]. In the

⋆ Maxime Monod has been partially funded by the Swiss National Science Foundation with grant 20021-113825.

streaming context, churn might be caused by failures, overloads, leaves and joins (e.g., users switching TV channels).

In a gossip protocol, each node periodically forwards every packet identifier it received to a subset of nodes picked uniformly at random. The size of this subset is called the *fanout*. Nodes subsequently request the packet whenever necessary. As no particular structure needs to be maintained, there is no need for a recovery protocol in case of churn, which is considered the norm rather than the exception. Robustness stems from the *proactive* and *random* selection of communication partners. This proactiveness is a major difference with respect to mesh-based techniques, relying on a rather static neighborhood, which *react* to churn by having every node select new neighbors after noticing malfunctions [17, 35]. In a sense, gossip-based protocols build extreme forms of mesh-based overlay networks with a continuously changing set of neighbors, and an ultimate splitting procedure where each packet is potentially disseminated through continuously changing dissemination paths, as opposed to explicit substream creation leading to multi-trees [4, 17, 35].

Gossip in action. Consider a stream of 600 kbps produced by a single source and intended to be disseminated to 270 PlanetLab nodes in a decentralized manner. Our preliminary experiments revealed the difficulty of disseminating through a static tree without any reconstruction even among 30 nodes. The static nature of the tree exacerbates the loss rate of UDP packets particularly in the presence of heavily loaded nodes, which may see their upload capabilities change by 20% from one experiment to the other. One might consider sophisticated reactive mechanisms to cope with the network dynamics but these are particularly challenging in highly dynamic environments.

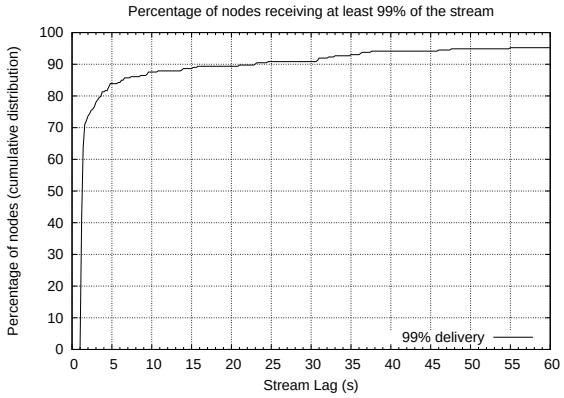


Fig. 1: Without constraining upload capabilities, a gossip with fanout 7 provides a stream of high quality and low lag to a large number of PlanetLab nodes.

Instead, we could obtain a good quality stream using a simple gossip protocol over all 270 PlanetLab nodes. Figure 1 reports on our experiments (which we detail later in the paper) by conveying a high *average delivery ratio* (the number of stream packets received over the total number of stream packets produced), and a low *stream lag* (the difference between the time the stream is produced at the source and the time it is viewed): 50% of the nodes receive 99% of the stream with a stream lag of 1.3 s, 75% of the nodes receive the same amount after 2.4 s and 90% after 21 s. The fanout considered here is 7. In a system of size n, and assuming a uniformly random peer selection, a fanout of $\ln(n)$ is the theoretical threshold between a non-connected and a well-connected communication graph. By overestimating $\ln(n)$, theory [15] and experiences [9] reveal that the graph gets fully connected with high probability.

However, this simple experiment, as well as the encouraging ones of [18, 19], rely on all nodes having uniform and high upload capabilities. Assuming nodes with limited and different upload capabilities (e.g., users having heterogeneous bandwidths), the situation is less favorable as shown in Figure 2. Several fanouts are tested given two upload capability distributions having the same average of 691 kbps. Dist1 contains three classes of nodes with 512 kbps, 768 kbps, and 3 Mbps of upload bandwidth (more details about the distributions are provided in Section 3), while dist2 is a uniform distribution.

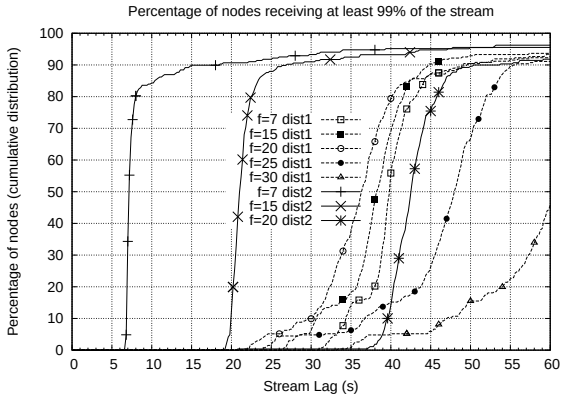


Fig. 2: When constraining the upload capability in a heterogeneous manner (with an average upload capability of 691kbps – dist1), the stream lag of all nodes significantly deteriorates. Adjusting the fanout (e.g., between 15 and 20) slightly improves the stream lag but a blind fanout increase (e.g., if it goes over 25) degrades performance. Moreover, the good fanout range in this case (fanouts of 15, 20 in dist1) reveals bad with a different distribution (uniform distribution - dist2) having the same average upload capability. With dist2, a fanout of 7 is optimal and much more effective than fanouts of 15 and 20.

A case for adaptation. A major reason for the mixed behavior of gossip in a heterogeneous setting is its homogeneous and load-balanced nature. All nodes are supposed to disseminate the same number of messages for they rely on the same fanout and dissemination period. However, this uniform distribution of load ignores the intrinsic heterogeneous nature of large-scale distributed systems where nodes may exhibit significant differences in their capabilities. Interestingly, and as conveyed by our experiments (and pointed out in [7]), a gossip protocol does indeed adapt to heterogeneity to a certain extent. Nodes with high bandwidth gossip rapidly, get thus pulled more often and can indeed sustain the overload to a certain extent. Nevertheless, as the bandwidth distribution gets tighter (closer to the stream rate) and more skewed (rich nodes get richer whereas poor nodes get poorer), there is a limit on the adaptation that traditional homogeneous gossip can achieve.

Heterogeneous gossip. Echoing [2, 7, 17, 27, 29, 30], we recognize the need to account for the heterogeneity between peers in order to achieve a more effective dissemination. This poses important technical challenges in the context of a gossip-based streaming application. First, an effective dissemination protocol needs to dynamically track and reflect the changes of available bandwidth over time. Second, the robustness of gossip protocols heavily relies on the proactive and uniform random selection of target peers: biasing this selection could impact the average quality of dissemination and the robustness to churn. Finally, gossip is simple and thus easy to deploy and maintain; sophisticated extensions that account for heterogeneity could improve the quality of the stream, but they would render the protocol more complex and thus less appealing.

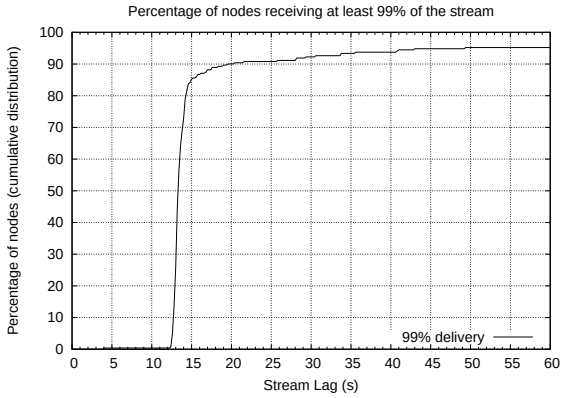


Fig. 3: With the same constrained distribution (dist1), HEAP significantly improves performance over a homogeneous gossip.

We propose a new gossip protocol, called *HEAP* (*HEterogeneity-Aware Gossip Protocol*), whose simple design follows from two observations. First, mathe-

matical results on epidemics and empirical evaluations of gossip protocols convey the fact that the robustness of the dissemination is ensured as long as the *average* of all fanouts is in the order of $\ln(n)$ [15] (assuming the source has at least a fanout of 1). This is crucial because the fanout is an obvious knob to adapt the contribution of a node and account for heterogeneity. A node with an increased (resp. decreased) fanout will send more (resp. less) information about the packets it can provide and in turn will be pulled more (resp. less) often. Second, using gossip dissemination, one can implement an aggregation protocol [13, 28] to continuously provide every node with a pretty accurate approximation of its relative bandwidth capability. Using such a protocol, HEAP dynamically leverages the most capable nodes by increasing their fanouts, while decreasing by the same proportion those of less capable nodes. HEAP preserves the simplicity and proactive (churn adaptation) nature of traditional (homogeneous) gossip, while significantly improving its effectiveness.

Applying HEAP in the PlanetLab context of Figure 2, i.e., assuming a heterogeneous bandwidth distribution exemplifying users using ADSL, we significantly improve streaming delay and quality (Figure 3). With an average fanout of 7, 50% of nodes receive 99% of the stream with 13.3 s lag, 75% with 14.1 s, and 90% with 19.5 s. More generally, we report on an exhaustive evaluation which shows that, when compared to a standard gossip, HEAP: *(i)* better matches the contributions of nodes to their bandwidth capabilities; *(ii)* enables a better usage of the overall bandwidth thus significantly improving the stream quality of all nodes and; *(iii)* significantly improves the resilience to churn.

Summary of contributions. We present HEAP, an information dissemination protocol that preserves the simplicity of standard gossip protocols, while significantly outperforming them with respect to the efficiency of streaming and resilience to churn. We also report on a full implementation of a P2P video streaming application using a proactive gossip protocol over a 270 PlanetLab node testbed with constrained and heterogeneous bandwidth distribution.

Roadmap. The rest of the paper is organized as follows. Section 2 gives some background on gossip-based content dissemination protocols and describes HEAP in detail. We report on the results of our experiments on PlanetLab in Section 3. Related work is covered in Section 4. Concluding remarks are given in Section 5.

2 HEAP

This section presents HEAP, *HEterogeneity-Aware Gossip Protocol*, a gossip protocol for collaborative content distribution in heterogeneous environments. We start this section by giving a short background on gossip-based content dissemination.

2.1 Background: gossip-based content dissemination

Consider a set of n nodes, and an event e to be disseminated in the system: e typically contains a series of application blocks (e.g., stream packets in a streaming

application), as well as control information. Gossip-based content dissemination generally follows a three-phase push-request-push protocol as depicted in Algorithm 1. The use of a three-phase mechanism is essential when dealing with high payloads in that it guarantees that a packet may never be delivered more than once to the same node, thus causing the average data rate induced by the protocol to be less than or equal to the stream rate.

The protocol operates as follows. Each node periodically contacts a fixed number, f (fanout), of nodes chosen according to the selectNodes() function and proposes a set of event identifiers (ids) to them with a [PROPOSE] message (line 5 for the broadcaster and 6 for other nodes). A node receiving such a message pulls the content it has not yet retrieved by sending a [REQUEST] to the proposing peer. The peer being pulled sends back the actual content (the payload) in a [SERVE] message that contains the requested events. This procedure is then iterated according to an infect-and-die model [8]. Each node proposes each event id, exactly once, to f other peers, thus avoiding the need to deal with time-to-live.

Algorithm 1 Standard gossip protocol

Initialization:
1: $\overline{f} := \ln(n) + c$ $\quad \{\overline{f}$ *is the average fanout*$\}$
2: eToPropose := eDelivered := eRequested := $\varnothing$
3: **start**(GossipTimer(gossipPeriod))

Phase 1 – Push event ids

procedure publish(e) **is**
4: deliverEvent(e)
5: gossip($\{e.id\}$)

upon (GossipTimer mod gossipPeriod) $= 0$ **do**
6: gossip(eToPropose)
7: eToPropose := $\varnothing$ $\quad \{$*Infect and die*$\}$

Phase 2 – Request events

upon receive [PROPOSE, eProposed] **do**
8: wantedEvents := $\varnothing$
9: **for all** $e.id \in$ eProposed **do**
10: **if** ($e.id \notin$ eRequested) **then**
11: wantedEvents := wantedEvents $\cup$ $e.id$
12: eRequested := eRequested $\cup$ wantedEvents
13: **reply** [REQUEST, wantedEvents]

Phase 3 – Push payload

upon receive [REQUEST, wantedEvents] **do**
14: askedEvents := $\varnothing$
15: **for all** $e.id \in$ wantedEvents **do**
16: askedEvents := askedEvents $\cup$ event(e.id)
17: **reply** [SERVE, askedEvents]

upon receive [SERVE, events] **do**
18: **for all** $e \in$ events **do**
19: **if** ($e \notin$ eDelivered) **then**
20: eToPropose := eToPropose $\cup$ $e.id$
21: eDelivered := eDelivered $\cup$ e
22: **deliver**(e)

Miscellaneous

function selectNodes(f) **returns** set of nodes **is**
23: **return** f uniformly random nodes
procedure gossip(event ids) **is**
24: commPartners := selectNodes(getFanout())
25: **for all** p $\in$ commPartners **do**
26: **send**(p) [PROPOSE, event ids]
function getFanout() **returns** Integer **is**
27: **return** the fanout of gossip dissemination

As discussed in the introduction, standard gossip-based content dissemination works very well in unconstrained or otherwise homogeneous network environments, in which the load-balancing features of gossip provide the greatest benefit. Nevertheless, it becomes inefficient in constrained [9] and heterogeneous scenarios. In these, the standard homogeneous gossip described in Algorithm 1 stabilizes at a state in which low-capability nodes saturate their bandwidth, while high-capability ones are underutilized. This results in congested queues and increases the transmission delays introduced by low-capability nodes, impacting the overall performance experienced by all the nodes in the system.

2.2 Adapting contribution

Algorithm 2 HEAP protocol details

```
Initialization:
 1: capabilities := ∅
 2: b := own available bandwidth
 3: stdGossip.Initialization
 4: start(AggregationTimer(aggPeriod))

Fanout Adaptation

function getFanout() returns Integer is
 5: return b/b̄ · f̄

Retransmission

upon receive [Propose, eProposed] do
 6: stdGossip.receive [Propose, eProposed]
 7: start(RetTimer(retPeriod, eProposed))

upon receive [Serve, events] do
 8: stdGossip.receive [Serve, events]
 9: cancel(RetTimer(retPeriod, events))

upon (RetTimer mod retPeriod) = 0 do
10: receive [Propose, eProposed]

Aggregation Protocol

upon (AggregationTimer mod aggPeriod) = 0 do
11: commPartners := selectNodes(f̄)
12: for all p ∈ commPartners do
13:     fresh = 10 freshest values from capabilities
14:     send(p) [Aggregation, fresh]

upon receive [Aggregation, otherCap] do
15: merge otherCap into capabilities
16: update b̄ using capabilities
```

HEAP addresses the limitations of standard gossip by preventing congestion at low-capability nodes through the adaptation of each node's workload. Consider two nodes A and B with upload capabilities b_A and b_B. HEAP adapts the contribution of each node to its capability and thus causes the upload rate resulting from node A's [Serve] messages to be b_A/b_B times as large as that of node B.

Key to HEAP's adaptation mechanism is the fact that, in a non-congested setting, each [Propose] message has roughly the same probability, p, to be accepted (thereby generating a subsequent [Serve] message) regardless of the bandwidth capability of its sender[6]. HEAP exploits this fact to dynamically adapt the gossip fanouts of nodes so that their contribution to the stream delivery remains proportional to their available bandwidth. Specifically, because the average number of proposals accepted in each gossip round can be computed as $p \cdot f$, f being the fanout of the proposing node, we can derive that the fanout f_A of node A should be b_A/b_B times the fanout of node B.

$$f_A = \frac{b_A}{b_B} \cdot f_B \tag{1}$$

Preserving reliable dissemination. Interestingly, Equation (1) shows that determining the ratios between the fanouts of nodes is enough to predict their average contribution as the three phases of Algorithm 1 guarantee that the average upload rate[7] over all nodes is less than or equal to the stream rate. However, simply setting the fanouts of nodes to arbitrary values that satisfy Equation 1 may lead to undesired consequences. On the one hand, a low average fanout may hamper the ability of a gossip dissemination to reach all nodes. On the other hand, a large average fanout may unnecessarily increase the overhead resulting from the dissemination of [Propose] messages.

[6] In reality, proposals from low-capability nodes incur in higher transmission delays and thus have a slightly lower probability of acceptance, but this effect is negligible when dealing with small [Propose] messages in a non-congested setting.

[7] Not counting the overhead of [Propose] and other messages.

HEAP strives to avoid these two extremes by relying on theoretical results showing that the reliability of gossip dissemination is actually preserved as long as a fanout value of $\overline{f} = \ln(n) + c$, n being the size of the network, is ensured *on average* [15], regardless of the actual fanout distribution across nodes. To achieve this, HEAP exploits a simple gossip-based aggregation protocol (see Algorithm 2) which provides an estimate of the average upload capability $\overline{b}$ of network nodes. A similar protocol can be used to continuously approximate the size of the system [13], but, for simplicity, we consider here that the initial fanout is computed knowing the system size in advance. The aggregation protocol works by having each node periodically gossip its own capability and the freshest received capabilities. We assume a node's capability is either *(i)* a maximal capability given by the user at the application level (as the maximal outgoing bandwidth the user wants to give to the streaming application) or *(ii)* computed, when joining, by a simple heuristic to discover the nodes upload capability, e.g., starting with a very low-capability while trying to upload as much as possible in order to reach its maximal capability as proposed in [34]. Each node aggregates the received values and computes an estimate of the overall average capability. Based on this estimate, each node, p_i, regulates its fanout, f_{p_i}, according to the ratio between its own and the average capability, i.e., $f_{p_i} = \overline{f} \cdot b_{p_i}/\overline{b}$.

3 Evaluation

We report in this section on our evaluation of HEAP in the context of a video streaming application on a testbed of ∼270 PlanetLab nodes. This includes a head-to-head comparison with a standard gossip protocol. In short, we show that, when compared to a standard gossip protocol: *(i)* HEAP adapts the actual load of each node to its bandwidth capability (Section 3.3), *(ii)* HEAP consistently improves the streaming quality of all nodes (Section 3.4), *(iii)* HEAP improves the stream lag from 40% to 60% over standard gossip (Section 3.5), *(iv)* HEAP resists to extreme churn situations where standard gossip collapses (Section 3.6). Before diving into describing these results in more details, we first describe our experimental setup.

3.1 Experimental setup

Video streaming application. We generate stream packets of 1316 bytes at a stream rate of 551 kbps on average. Every *window* is composed of 9 *FEC-coded packets* and 101 buffered stream packets resulting in an effective rate of 600 kbps.

Gossiping parameters. The gossiping period of each node is set to 200 ms, which leads to grouping an average of 11.26 packet ids per [Propose]. The fanout is set to 7 for all nodes in the standard gossip protocol, while in HEAP, the average fanout is 7 across all nodes. The aggregation protocol gossips the 10 freshest local capabilities every 200 ms, costing around 1 KB/s and is thus completely marginal compared to the stream rate.

Message retransmission and bandwidth throttling. Given the random nature of its gossip-based dissemination process, HEAP does not attempt to establish stable TCP connections, but rather combines UDP datagrams with a retransmission mechanism. To further reduce message losses, HEAP also exploits a bandwidth throttling mechanism. This guarantees that nodes never attempt to send bursts of data that exceed their available bandwidth. Excess packets resulting from bursts are queued at the application level, and sent as soon as there is enough available bandwidth. To guarantee a fair comparison in our evaluation, we also integrated both retransmission and bandwidth throttling into the standard gossip protocol.

PlanetLab and network capabilities. PlanetLab nodes, located mostly in research and educational institutions, benefit from high bandwidth capabilities. As such, PlanetLab is not representative of a typical collaborative peer-to-peer system [26], in which most nodes would be sitting behind ADSL connection, with an asymmetric bandwidth and limited upload/download capabilities. We thus artificially limit the upload capability of nodes so that they match the bandwidth usually available for home users. We focus on upload as it is a well-known fact that download capabilities are much higher than upload ones. As we rely on UDP, we implemented, at the application level, an upload rate limiter that queues packets which are about to cross the bandwidth limit. In practice, nodes never exceed their given upload capability, but some nodes (between 5% and 7%), contribute way less than their capability, because of high CPU load and/or high bandwidth demand by other PlanetLab experiments. In other words, the average used capability of nodes is always less than or equal to their given upload limit.

We consider three different distributions of upload capabilities, depicted in Table 1 and inspired from the distributions used in [35]. The *capability supply ratio* (CSR, as defined in [35]) is the ratio of the average upload bandwidth over the stream rate. We only consider settings in which the global available bandwidth is enough to sustain the stream rate. Yet the lower the capability ratio, the closer we stand to that limit. The ms-691 distribution was referred to as dist1 in Section 1.

			Fraction of nodes		
Name	CSR	Average	2 Mbps	768 kbps	256 kbps
ref-691	1.15	691 kbps	0.1	0.5	0.4
ref-724	1.20	724 kbps	0.15	0.39	0.46
Name	CSR	Average	3 Mbps	1 Mbps	512 kbps
ms-691	1.15	691 kbps	0.05	0.1	0.85

Table 1: The reference distributions ref-691 and ref-724, and the more skewed distribution ms-691.

Each distribution is split into three classes of nodes. The skewness of an upload distribution is characterized by the various percentages of each class of nodes: in the most skewed distribution we consider (ms-691), most nodes are in the *poorest* category and only 15% of nodes have an upload capability higher than the stream rate.

3.2 Evaluation metrics

In the following, we first show that HEAP adapts the contribution of nodes according to their upload capability, and then we show that HEAP provides users with a good stream. We consider two metrics. The first is the *stream lag* and is defined as the difference between the time the stream was published by the source and the time it is actually delivered to the player on the nodes.[8] The second is the *stream quality*, which represents the percentage of the stream that is viewable. A FEC-encoded window is *jittered* as soon as it does not contain enough packets (i.e., at least 101) to be fully decoded. A X% jittered stream therefore means that X% of all the windows were jittered. Note that a jittered window does not mean that the window is entirely lost. Because we use systematic coding, a node may still receive 100 out of the 101 original stream packets, resulting in a 99% delivery ratio in a given window. We therefore also assess the quality of the jittered windows by giving the average delivery ratio in all jittered windows.

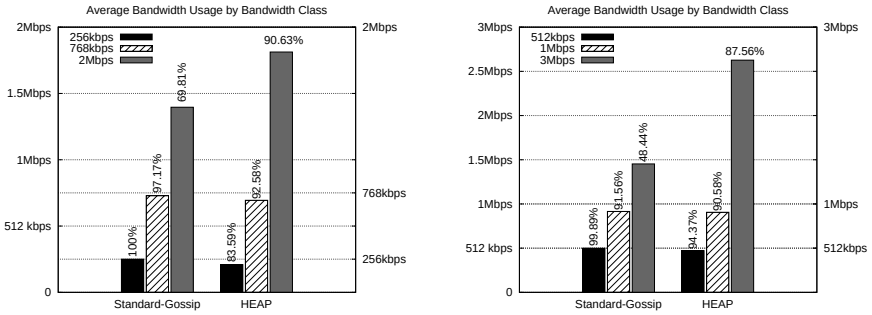
3.3 Adaptation to heterogeneous upload capabilities

We considered all three configurations. In ref-691, ref-724 and ms-691, resp. 60%, 54% and 15% of the nodes have an available bandwidth higher than the one required on average for the stream rate. As we observed similar results in ref-691 and ref-724, we only report on ref-691 in Figure 4a. Results on ms-691 are reported on Figure 4b.

Figure 4a depicts the breakdown of the contributions among the three classes of nodes. For example, the striped bar for standard gossip means that nodes having an upload capability of 768 kbps use 97.17 % of their available bandwidth. It is interesting to observe that nodes contribute somewhat proportionally to their upload capabilities even in standard gossip. This is because of the correlation between upload capability and latency: packet ids sent by high-capability nodes are received before those sent by lower-capability ones. Consequently, the former are requested first and serve the stream to more nodes than the latter. In addition, nodes with low capabilities are overloaded faster and therefore naturally serve fewer nodes (either because they are slower or because they are subject to more packet drops). Yet, despite this natural self-adaptation, we observe that high-capability nodes are underutilized in standard gossip. To the contrary,

[8] A different and complementary notion, *startup delay*, is the time a node takes to buffer the received packets until they are sent to the video player. Note that in a gossip protocol like HEAP the startup delay of all nodes is similar because of the unstructured and dynamic nature of gossip.

HEAP homogeneously balances the load on all nodes by correctly adapting their gossip fanouts: all nodes approximately consume 90% of their bandwidth. This highlights how the bandwidth consumption of standard gossip and HEAP on Figure 4a are caused by opposite reasons: congestion of low-capability nodes in standard gossip and fanout adaptation, which prevents congestion, in HEAP.



(a) ref-691 (b) ms-691

Fig. 4: Bandwidth consumption.

Figure 4b conveys the limits of the self-adaptation properties of standard gossip with an upload distribution in which only 15% of the nodes have an upload capability higher than the stream rate (ms-691). We observe that with standard gossip, the 5% nodes with high capabilities only use 48.44% of their bandwidth because their limited fanout does not allow them to serve more nodes. In HEAP, on the other hand, the 5% high-capability nodes can serve with up to 87.56% of their bandwidth, lowering the congestion of the low-capability nodes and providing much better performance than standard gossip in terms of quality as we show in next section.

3.4 Stream quality

Our next experiment compares the percentages of jitter-free windows received by nodes in the three considered scenarios. Results are depicted in Figures 5, 6a and 6b. For instance, the black bar in Figure 5 for standard gossip indicates that nodes with low capabilities in ref-691 have only 18% of the windows that are not jittered (considering packets received with a stream lag of up to 10 s.). The same figure also shows that HEAP significantly improves this value, with low-capability nodes receiving more than 90% of jitter-free windows. This reflects the fact that HEAP allows high-capability nodes to assist low-capability ones.

Results in Figure 6a are even more dramatic: high-capability nodes receive less than 33% of jitter-free windows in standard gossip, whereas all nodes receive more than 95% of jitter-free windows with HEAP.

Figure 6b clearly conveys the collaborative nature of HEAP when the global available bandwidth is higher (ref-724). The whole system benefits from the fact that nodes contribute according to their upload capability. For instance, the number of jitter-free windows that low-capability nodes obtain increases from 47% for standard gossip to 93% for HEAP. These results are complemented by Table 2, which presents the average delivery ratio in the jittered windows for both protocols, for each class of nodes in the three considered distributions. Again, results show that HEAP is able to provide good performance to nodes regardless of their capability classes. It should be noted, however, that the table provides results only for the windows that are jittered, which are a lot more in standard gossip than in HEAP. This explains the seemingly bad performance of HEAP in a few cases such as for high-bandwidth nodes in ref-724.

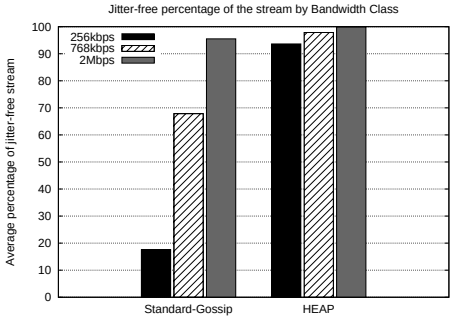
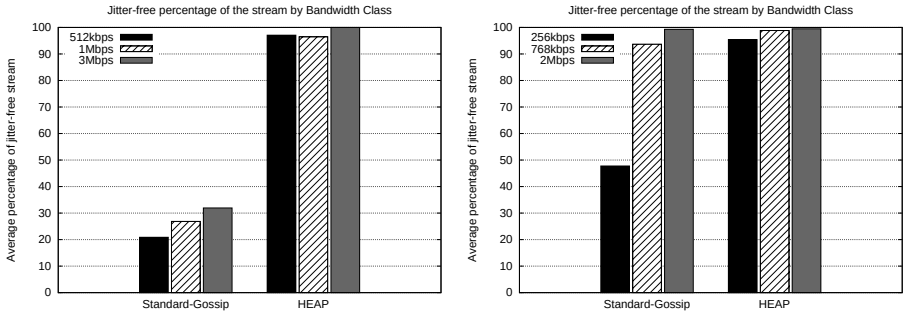


Fig. 5: Stream Quality (ref-691).

	Standard gossip			HEAP		
upload capability	256 kbps	768 kbps	2 Mbps	256 kbps	768 kbps	2 Mbps
ref-691	63.4%	87.1%	89.3%	80.4%	77.1%	89.8%
ref-724	75.6%	88.6%	89.6%	87.9%	87.7%	64.4%
upload capability	512 kbps	1 Mbps	3 Mbps	512 kbps	1 Mbps	3 Mbps
ms-691	42.8%	56.5%	64.5%	83.7%	80.7%	90.9%

Table 2: Average delivery rates in windows that cannot be fully decoded.

Figure 7 conveys the cumulative distribution of the nodes that view the stream as a function of the percentage of jitter. For instance, the point ($x = 0.1$, $y = 85$) on the HEAP - 10 s lag curve indicates that 85% of the nodes experience



(a) ms-691 (b) ref-724

Fig. 6: Stream quality by capability class.

a jitter that is less than or equal to 10%. Note that in this figure, we do not differentiate between capability classes. We consider standard gossip and HEAP in two settings: offline and with 10 s lag. We present offline results in order to show that, with standard gossip, nodes eventually receive the stream. However, with a 10 s lag, standard gossip achieves very poor performance: most windows are jittered. In contrast, HEAP achieves very good performance even with a 10 s lag.

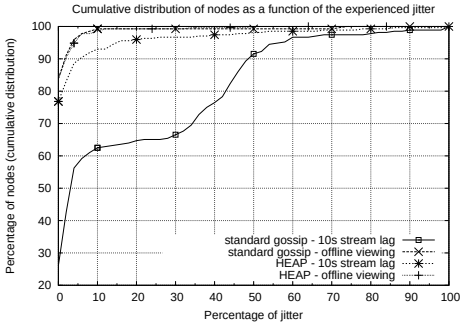
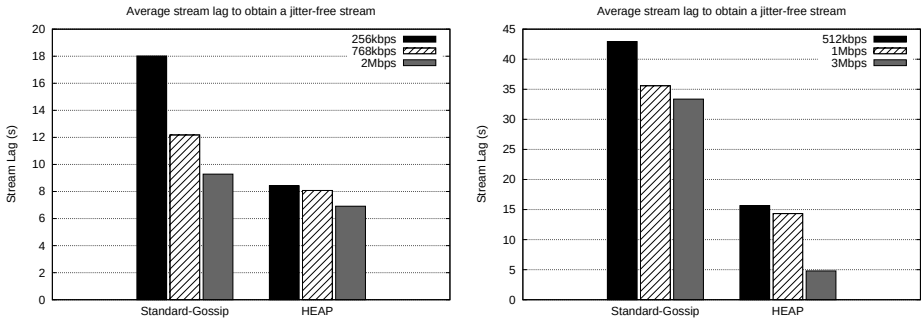


Fig. 7: Cumulative distribution of experienced jitter (ref-691). With HEAP and a stream lag of 10 s, 93% of the nodes experience less than 10% jitter.

3.5 Stream lag

Next, we compare the stream lag required by HEAP and standard gossip to obtain a non-jittered stream. We report the results for ref-691 and ms-691 on Figures 8a and 8b, respectively. In both cases, HEAP drastically reduces the

stream lag for all capability classes. Moreover, as shown in Figure 8b, the positive effect of HEAP significantly increases with the skewness of the distribution.



(a) ref-691 (b) ms-691

Fig. 8: Stream lag by capability class.

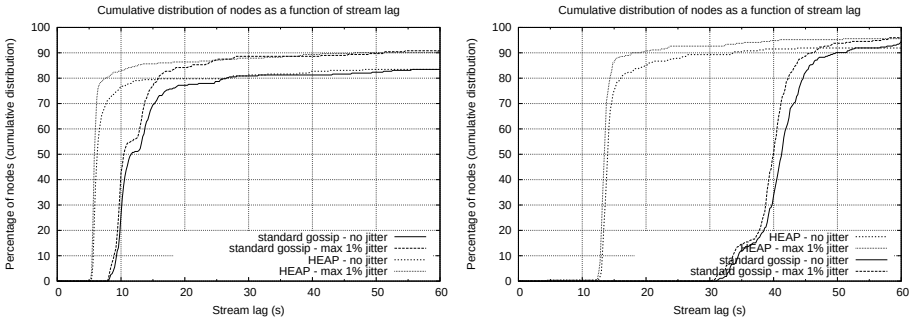
Figures 9a and 9b depict the cumulative distribution of nodes viewing the stream as a function of the stream lag, without distinguishing capability classes. We compare standard gossip and HEAP in two configurations: without jitter and with less than 1% of jitter. Sporadically, some PlanetLab nodes seem temporarily frozen, due to high CPU load and/or suffer excessive network problems explaining why neither protocol is able to deliver the stream to 100% of the nodes.[9] Still, both plots show that HEAP consistently outperforms standard gossip. For instance, in ref-691, HEAP requires 12 s to deliver the stream to 80% of the nodes without jitter, whereas standard gossip requires 26.6 s.

Table 3 complements these results by showing the percentage of nodes that can view a jitter-free stream for each bandwidth class and for the three described distributions. In brief, the table shows that the percentage of nodes receiving a clear stream increases as bandwidth capability increases for both protocols. However, HEAP is able to improve the performance experienced by poorer nodes without any significant decrease in the stream quality perceived by high-bandwidth nodes.

3.6 Resilience to catastrophic failures

Finally, we assess HEAP's resilience to churn in two catastrophic-failure scenarios where 20% and 50% respectively of the nodes fail simultaneously 60 s after the beginning of the experiment. The experiments are based on the ref-691 bandwidth distribution, while the percentage of failing nodes is taken uniformly

[9] Note that when running simulations without messages loss, 100% of the nodes received the full stream.



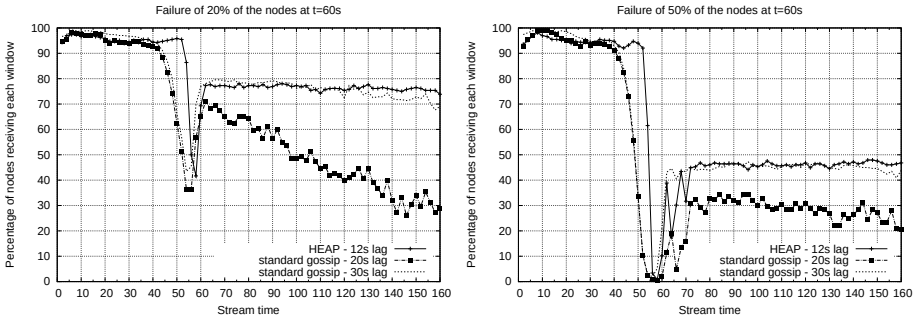
(a) ref-691 (b) ms-691

Fig. 9: Cumulative distribution of stream lag values.

	Standard gossip			HEAP		
bandwidth	256 kbps	768 kbps	2 Mbps	256 kbps	768 kbps	2 Mbps
ref-691 (10 s lag)	0	29.80	86.67	65.93	79.61	96.55
ref-724 (10 s lag)	0	67.52	97.73	61.95	74.34	93.02
bandwidth	512 kbps	1 Mbps	3 Mbps	512 kbps	1 Mbps	3 Mbps
ms-691 (20 s lag)	0	0	0	84.58	89.66	85.71

Table 3: Percentage of nodes receiving a jitter-free stream by capability class.

at random from the set of all nodes, i.e., keeping the average capability supply ratio unchanged. In addition, we configure the system so that surviving nodes learn about the failure an average of 10 s after it happened.



(a) 20% of nodes crashing. (b) 50% of nodes crashing.

Fig. 10: Resilience in the presence of catastrophic failures.

Figure 10a depicts, for each encoded window in the stream, the percentage of nodes that are able to decode it completely, i.e., without any jitter. The plot highlights once more the significant improvements provided by HEAP over standard gossip-based content dissemination. The solid line showing HEAP with a 12 s lag shows that the percentage of nodes decoding each window is always close to 100% (or to the 80% of nodes remaining after the failure) except for the stream packets generated immediately before the failure. The reason for the temporary drop in performance is that the failure of a node causes the disappearance of all the packets that it has delivered but not yet forwarded. Clearly, windows generated after the failure are instead correctly decoded by almost all remaining nodes. The plot also shows two additional lines depicting the significantly worse performance achieved by standard gossip-based dissemination.

The number of nodes receiving the stream with a 20 s lag in standard gossip is, in fact, much lower than that of those receiving it with only 12 s of lag in HEAP. Only after 30 s of lag is standard gossip able to reach a performance that is comparable to that of HEAP after 12 s. The figure also highlights that the number of packets lost during the failure is higher in standard gossip than in HEAP (the width of the drop is larger). The reason is that in standard gossip upload queues tend to grow larger than in HEAP. Thus packets that are lost as a result of nodes that crash span a longer time interval in standard gossip than they do in HEAP. Finally, the continuous decrease in the 20 s-lag line for standard gossip shows that the delay experienced by packets in standard gossip increases as time elapses: this is a clear symptom of congestion that is instead not present in HEAP.

Figure 10b provides similar information for a scenario in which 50% of the nodes fail simultaneously. HEAP is still able to provide the stream to the remaining nodes with a lag of less than 12 s. Conversely, standard gossip achieves mediocre performance after as many as 20 s of lag.

4 Related work

When contrasting HEAP with related work, we distinguish two classes of content-dissemination protocols: *(i)* proactive protocols that continuously change the dissemination topology, namely gossip-based dissemination schemes, and *(ii)* reactive protocols which only change the dissemination topology (possibly in a random manner) in case of malfunctions (e.g., churn). This latter set includes tree and mesh-based protocols.

4.1 Proactive protocols

Several proactive protocols have incorporated some adaptation features, but none does so by dynamically adapting the fanout of the nodes according to their relative (bandwidth) capabilities. The protocol of [10] aims at increasing the reliability of a spanning tree, by having each node in the tree dynamically

adapt its number of children using global knowledge about the reliability of nodes and network links.

In Smart Gossip [16], nodes of a wireless network may decide not to gossip depending on the number of nodes in their surrounding. In CREW [7], a three-phase gossip protocol (similar to that of Section 2.1) is used to disseminate large content in the context of file sharing. Nodes locally decide, when their bandwidth is exhausted, to stop offering data.

In Gravitational Gossip [14], the fanin of nodes (i.e., the number of times a node is chosen as a gossip target) may be adjusted based on the quality of reception they expect. This is achieved by biasing the node selection such that some nodes have a higher probability to be selected for gossip than others. The technique is however static and focuses on the incoming traffic that nodes receive. Because of the three-phase nature of HEAP, nodes have a payload fanin of 1.

4.2 Reactive protocols

Some tree- and mesh-based protocols do have nodes dynamically adapt their neighborhood sets. However, such adaptation is only achieved after churn or malfunctions, and as such it is not proactive as in HEAP, or in any gossip dissemination protocol.

Multi-tree schemes such as Splitstream [4] and Chunkyspread [29] split streams over diverse paths to enhance their reliability. This comes for free in gossip protocols where the neighbors of a node continuously change. In a sense, a gossip dissemination protocol dynamically provides different dissemination paths for each stream packet, providing the ultimate splitting scheme. Chunkyspread accounts for heterogeneity using the SwapLinks protocol [30]. Each node contributes in proportion to its capacity and/or willingness to collaborate. This is reflected by heterogeneous numbers of children across the nodes in the tree.

The approaches of [2, 27] propose a set of heuristics that account for bandwidth heterogeneity (and node uptimes) in tree-based multicast protocols. This leads to significant improvements in bandwidth usage. These protocols aggregate global information about the implication of nodes across trees, by exchanging messages along tree branches, in a way that relates to our capability aggregation protocol.

Mesh-based systems [5, 17, 20, 22–24, 35] are appealing alternatives to tree-based ones. They are similar to gossip in the sense that their topology is unstructured. Some of those, namely the latest version of Coolstreaming [17] and GridMedia [35] dynamically build multi-trees on top of the unstructured overlay when nodes perceive they are stably served by their neighbors. Typically, every node has a *view* of its neighbors, from which it picks new partners if it observes malfunctions. In the extreme case, a node has to seek for more or different communication partners if none of its neighbors is operating properly. Not surprisingly, it was shown in [17,20] that increasing the view size has a very positive effect on the streaming quality and is more robust in case of churn. Gossip protocols like HEAP are extreme cases of these phenomena because the views they rely on keep continuously changing.

Finally, [31] addresses the problem of building an optimized mesh in terms of network proximity and latency, in the presence of *guarded* peers, i.e., peers that are behind a NAT or firewall. This work led to mixing application level multicast with IP multicast whenever possible [34]. The core of this research is now commercially used in [33] but little is known on the dissemination protocol. At the time the prototype was used for research, some nodes were fed by super peers deployed on PlanetLab and it is reasonable to think that those super peers are now replaced by dedicated servers in the commercial product. It is for instance known that the dissemination protocol of PPLive [25] substantially relies on a set of super peers and thus does not represent a purely decentralized solution [12].

5 Concluding remarks

This paper presents HEAP, a new gossip protocol which adapts the dissemination load of the nodes to account for their heterogeneity. HEAP preserves the simplicity and proactive (churn adaptation) nature of traditional homogeneous gossip, while significantly improving its effectiveness. Experimental results with a video streaming application on PlanetLab convey the improvement of HEAP over a standard homogeneous gossip protocol with respect to stream quality, bandwidth usage and resilience to churn. When the stream rate is close to the average available bandwidth, the improvement is even more significant.

A natural way to further improve the quality of gossiping is to bias the neighbor selection towards rich nodes in the early steps of dissemination. Our early experiments reveal that this can be beneficial at the first step of the dissemination (i.e., from the source) but reveals not trivial if performed in later steps.

We considered bandwidth as the main heterogeneity factor, as it is indeed crucial in the context of streaming. Other factors might reveal important in other applications (e.g., node interests, available CPU). We believe HEAP could easily be adapted to such factors by modifying the underlying aggregation protocol accordingly. Also, we considered the choice of the fanout as the way to adjust the load of the nodes. One might also explore the dynamic adaptation of the gossip targets, the frequency of the dissemination or the memory size devoted to the dissemination.

There are some limitations to adaptation and these provide interesting research tracks to pursue. While adapting to heterogeneity, a natural behavior is to elevate certain wealthy nodes to the rank of temporary superpeers, which could potentially have an impact in case of failures. Moreover, an attacker targeting highly capable nodes could degrade the overall performance of the protocol. Likewise, the very fact that nodes advertise their capabilities may trigger freeriding vocations, where nodes would pretend to be poor in order not to contribute to the dissemination. We are working towards a freerider-tracking protocol for gossip in order to detect and punish freeriding behaviors [11].

Finally, since gossip targets are periodically changing and because sent messages are very small, it is quite natural to transfer them via UDP. Nevertheless,

doing so can have a negative impact on other applications competing for bandwidth. In other words, our protocol is not *TCP-friendly* as it might simply take priority over other applications, similar to most commercial voice-over-IP protocols. Making protocols using multiple incoming streams TCP-friendly was quite difficult [21,32] assuming the serving nodes were static. Doing the same for ever changing neighbors such as in a gossip is therefore a problem on its own and needs further research.

Acknowledgements

The authors would like to thank Ken Birman, Pascal Felber, Ali Ghodsi and Dahlia Malkhi for useful comments.

References

1. K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal Multicast. *TOCS*, 17(2):41–88, 1999.
2. M. Bishop, S. Rao, and K. Sripanidulchai. Considering Priority in Overlay Multicast Protocols under Heterogeneous Environments. In *Proc. of INFOCOM*, 2006.
3. T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg. Epidemic Live Streaming: Optimal Performance Trade-Offs. In *Proc. of SIGMETRICS*, 2008.
4. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *Proc. of SOSP*, 2003.
5. Y.-H. Chu, S. Rao, and H. Zhang. A Case for End System Multicast. *JSAC*, 20(8):1456–1471, 2000.
6. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. In *Proc. of PODC*, 1987.
7. M. Deshpande, B. Xing, I. Lazardis, B. Hore, N. Venkatasubramanian, and S. Mehrotra. CREW: A Gossip-based Flash-Dissemination System. In *Proc. of ICDCS*, 2006.
8. P. Eugster, R. Guerraoui, S. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight Probabilistic Broadcast. *TOCS*, 21(4):341–374, 2003.
9. D. Frey, R. Guerraoui, A.-M. Kermarrec, M. Monod, and V. Quéma. Stretching Gossip with Live Streaming. In *Proc. of DSN*, 2009.
10. B. Garbinato, F. Pedone, and R. Schmidt. An Adaptive Algorithm for Efficient Message Diffusion in Unreliable Environments. In *Proc. of DSN*, 2004.
11. R. Guerraoui, K. Huguenin, A.-M. Kermarrec, and M. Monod. On Tracking Freeriders in Gossip Protocols. In *Proc. of P2P (to appear)*, 2009.
12. X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross. A Measurement Study of a Large-Scale P2P IPTV System. *TMM*, 9(8):1672–1687, 2007.
13. M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-Based Aggregation in Large Dynamic Networks. *TOCS*, 23(3):219–252, 2005.
14. K. Jenkins, K. Hopkinson, and K. Birman. A Gossip Protocol for Subgroup Multicast. In *Proc. of ICDCS Workshops*, 2001.
15. A.-M. Kermarrec, L. Massoulié, and A. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. *TPDS*, 14(3):248–258, 2003.

16. P. Kyasanur, R. R. Choudhury, and I. Gupta. Smart Gossip: An Adaptive Gossip-based Broadcasting Service for Sensor Networks. In *Proc. of MASS*, 2006.
17. B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang. Inside the New Coolstreaming: Principles, Measurements and Performance Implications. In *Proc. of INFOCOM*, 2008.
18. H. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robinson, L. Alvisi, and M. Dahlin. FlightPath: Obedience vs Choice in Cooperative Services. In *Proc. of OSDI*, 2008.
19. H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR Gossip. In *Proc. of OSDI*, 2006.
20. C. Liang, Y. Guo, and Y. Liu. Is Random Scheduling Sufficient in P2P Video Streaming? In *Proc. of ICDCS*, 2008.
21. L. Ma and W. Ooi. Congestion Control in Distributed Media Streaming. In *Proc. of INFOCOM*, 2007.
22. N. Magharei and R. Rejaie. PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming. In *Proc. of INFOCOM*, 2007.
23. V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr. Chainsaw: Eliminating Trees from Overlay Multicast. In *Proc. of IPTPS*, 2005.
24. F. Picconi and L. Massoulié. Is There a Future for Mesh-Based live Video Streaming? In *Proc. of P2P*, 2008.
25. PPLive. `http://www.pplive.com`.
26. N. Spring, L. Peterson, A. Bavier, and V. Pai. Using Planetlab for Network Research: Myths, Realities, and Best Practices. *OSR*, 40(1):17–24, 2006.
27. Y.-W. Sung, M. Bishop, and S. Rao. Enabling Contribution Awareness in an Overlay Broadcasting System. *CCR*, 36(4):411–422, 2006.
28. R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *TOCS*, 21(2):164–206, 2003.
29. V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous Unstructured Tree-Based Peer to Peer Multicast. In *Proc. of ICNP*, 2006.
30. V. Vishnumurthy and P. Francis. On Heterogeneous Overlay Construction and Random Node Selection in Unstructured P2P Networks. In *Proc. of INFOCOM*, 2006.
31. W. Wang, C. Jin, and S. Jamin. Network Overlay Construction under Limited End-to-End Reachability. In *Proc. of INFOCOM*, 2005.
32. J. Widmer and M. Handley. Extending Equation-based Congestion Control to Multicast Applications. In *Proc. of SIGCOMM*, 2001.
33. Zattoo. `http://www.zattoo.com`.
34. B. Zhang, W. Wang, S. Jamin, D. Massey, and L. Zhang. Universal IP multicast delivery. *Computer Networks*, 50(6):781–806, 2006.
35. M. Zhang, Q. Zhang, L. Sun, and S. Yang. Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better? *JSAC*, 25(9):1678–1694, 2007.