HAL
open science

# Stretching Gossip with Live Streaming

Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Maxime Monod, Vivien Quéma

## To cite this version:

Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Maxime Monod, Vivien Quéma. Stretching Gossip with Live Streaming. DSN 2009, Jun 2009, Estoril, Portugal. inria-00436130

**HAL Id: inria-00436130**

**https://inria.hal.science/inria-00436130v1**

Submitted on 26 Nov 2009

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

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

# Stretching Gossip with Live Streaming

Davide Frey[§] Rachid Guerraoui[†] Anne-Marie Kermarrec[§]
Maxime Monod[†*] Vivien Quéma[‡]
[†]Ecole Polytechnique Fédérale de Lausanne
[§]INRIA Rennes-Bretagne Atlantique
[‡]CNRS

## Abstract

*Gossip-based information dissemination protocols are considered easy to deploy, scalable and resilient to network dynamics. They are also considered highly flexible, namely tunable at will to increase their robustness and adapt to churn. So far however, they have mainly been evaluated through simulation, very often assuming ideal settings.*

*Instead, in this paper, we report on an extensive study of gossip protocols, deployed on a 230 Planetlab node testbed, in the context of a challenging video streaming application in environments with constrained bandwidths. More precisely, we assess the impact of varying the well known knobs of gossip, fanout and refresh rate, in various upload-bandwidth distributions and churn. Our results show that in such challenging contexts, the performance of gossip protocols may be hampered by high fanout values. We also show that the more proactive a gossip protocol, the better it copes with churn. For instance, when 20% of the nodes simultaneously crash, 70% of the remaining nodes do not suffer any loss in stream quality, while the others only experience a performance decrease for an average of 5 seconds around the churn event.*

## 1. Introduction

Gossip-based protocols [1, 3, 4, 9] are well known for their robustness and ability to cope with network dynamics. A lot of efforts have thus been devoted to their theoretical analysis and evaluation [6, 11]. It was for instance shown that the number of communication partners (called *fanout*) in a system of size $n$, should be greater than a threshold value $ln(n)$ for the gossip dissemination to reach all nodes with high probability [7].

Gossip protocols have also been shown to be effective for challenging applications like live streaming [2]. Yet, most of the work evaluating gossip has considered ideal settings; e.g., unconstrained bandwidth, no (or uniform) message loss, global knowledge about the state of all nodes. Evaluations conducted through simulation [2, 4, 10] also assume that key parameters of gossip, such as fanout and gossip rate, can be arbitrarily tuned to improve robustness and to adequately adapt to network dynamics. Moreover, exceptions of gossip experiments in real settings assume infinite bandwidth [9], or consider applications with low bandwidth needs, such as membership maintenance [5].

In the presence of heterogeneous bandwidth and greedy applications, we have no evidence that gossip will fulfill its promises. The impact of the key parameters of gossip protocols in such contexts remains unexplored. For instance, the impact of varying the fanout, an obvious knob to tune the robustness of a gossip protocol, has never been determined.

The rate at which the gossip partners are changed reflects the *proactiveness* of a gossip protocol. The impact of varying this rate has never been studied either. At one extreme, one might consider a gossip protocol where nodes change their neighbors at every communication step (this is the scheme typically considered in theoretical studies). At the other extreme, nodes would never change their communication partners unless they notice malfunctions; this is typically the approach underlying mesh-based systems where the unstructured overlay used to create random or deterministic dissemination trees, once constructed, is kept as is until the mesh connectivity is reduced (e.g., a node has less than a given number of neighbors), or a node feels it is incorrectly fed [8]. A wide range of schemes can be considered between these two extremes.

In this paper, we report on extensive experimentation on gossip protocols in a realistic environment and in the context of greedy applications with respect to bandwidth, namely video streaming applications. We evaluate the impact of the key parameters of gossip protocols with various churn scenarios and bandwidth capabilities. Finally, we do not consider any repair mechanism or underlying overlay

*Maxime Monod has been partially funded by the Swiss National Science Foundation with grant 200021-113825.

structure, thus preserving the inherent simplicity of gossip.

In short, our results show that gossip can be very effective in greedy and capability-constrained applications, but only within a small window of parameter ranges. First, the resulting stream quality is very sensitive to the fanout value. The power of gossip is unleashed when the fanout is slightly larger than $ln(n)$ but degrades drastically with higher fanout values as a result of higher contention. For example, with a bandwidth cap of 700 kbps for a stream rate of 600 kbps, gossip reaches its optimal performance with a fanout of 7 (230 nodes). Second, gossip is most effective when the set of communication partners is continuously changing, particularly in the presence of churn. When 20% of the nodes simultaneously crash, 70% of the remaining nodes do not suffer any loss in quality, while the remaining ones only experience a performance decrease for an average of 5 around the churn event. These numbers drop to 30% when communication partners are refreshed only every 2 gossip rounds.

Finally, our results show that allowing nodes to change their incoming communication partners explicitly by means of periodic requests to be fed by a new set of nodes does not provide any improvement over standard gossip techniques. This further confirms the strength of gossip protocols in their simplest form.

## 2. Gossip-based content dissemination

Consider a set of $n$ nodes, and an event $e$ to be disseminated in the system: $e$ typically consists in a payload and an id. Gossip-based content dissemination protocols generally follow a three-phase push-request-push pattern as depicted in Algorithm 1. The protocol follows an *infect-and-die* model [7]. Each node periodically contacts a fixed number, $f$ (fanout), of nodes chosen according to the selectNodes function and proposes them a set of event ids (of the events it has received) with a [PROPOSE] message (line 5 for the broadcaster and 6 for other nodes). Upon receipt of a [PROPOSE], nodes pull the content needed with a [REQUEST] message to the proposing peer. The peer being pulled then sends back the actual content (the event) in a [SERVE] message that contains the requested events. Such a protocol leverages the reliability of gossip to disseminate the event ids while avoiding redundancy in content dissemination, which would increase the consumed bandwidth.

## 3. Tailoring gossip-based dissemination

While the protocol is simple, its parameters may be tuned to improve efficiency, reliability, or resilience to churn. We explore the parameter space as follows.

**Fanout.** The fanout is defined as the number of communication partners each node contacts in each gossip operation.

**Algorithm 1** Standard gossip protocol

```
Initialization:
 1: f := ln(n) + c
 2: eventsToPropose := eventsDelivered := requestedEvents := ∅
 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(eventsToPropose)
 7: eventsToPropose := ∅                                  {Infect and die}
Phase 2 – Request events
upon receive [PROPOSE, eventsProposed] do
 8: wantedEvents := ∅
 9: for all e.id ∈ eventsProposed do
10:     if (e.id ∉ requestedEvents) then
11:         wantedEvents := wantedEvents ∪ e.id
12: requestedEvents := requestedEvents ∪ wantedEvents
13: reply [REQUEST, wantedEvents]
14: if (e requested less than K times) then
15:     start(RetTimer(retPeriod, eventsProposed))
Phase 3 – Push payload
upon receive [REQUEST, wantedEvents] do
16: askedEvents := ∅
17: for all e.id ∈ wantedEvents do
18:     askedEvents := askedEvents ∪ getEvent(e.id)
19: reply [SERVE, askedEvents]
upon receive [SERVE, events] do
20: for all e ∈ events do
21:     if (e ∉ eventsDelivered) then
22:         eventsToPropose := eventsToPropose ∪ e.id
23:         deliverEvent(e)
24: cancel(RetTimer(retPeriod, events))
Retransmission
upon (RetTimer(retPeriod, eventsProposed) mod retPeriod) = 0 do
25: receive [PROPOSE, eventsProposed]
Miscellaneous
function selectNodes(f) returns set of nodes is
26: return f uniformly random chosen nodes in the set of all nodes
function getEvent(event id) returns event is
27: return the event corresponding to the id
procedure deliverEvent(e) is
28: deliveredEvents := deliveredEvents ∪ e
29: deliver(e)
procedure gossip(event ids) is
30: communicationPartners := selectNodes(f)
31: for all p ∈ communicationPartners do
32:     send(p) [PROPOSE, event ids]
```

It has been theoretically shown that a fanout greater than $\ln(n)$ in an infect-and-die model [7] ensures a highly reliable dissemination. Theory also assumes that increasing the fanout results in an even more robust (as the probability to receive an event id increases) and faster dissemination (as the degree of the resulting dissemination tree increases). In practice, however, too high a fanout can negatively impact performance as heavily requested nodes may exceed their capabilities in bandwidth constrained environments.

**Proactiveness.** We define proactiveness as the *rate* at which a node modifies its set of communication partners. We explore two ways of modifying this set.

First, the node may locally refresh its set of communication partners and change the output of selectNodes every

$X$ calls. In short, when $X = 1$ the gossip partners of the node change at every call to selectNodes (i.e., every gossip period), whereas $X = \infty$ means that the communication partners of a node never change.

Second, every $Y$ gossip periods, the node may contact $f$ random partners asking to be inserted in their views. When $Y = 1$, a node $A$ sends a *feed-me* message to $f$ random partners every gossip period asking them to feed it. Each of the random $f$ partners replaces a random node from its current set of $f$ partners with $A$.

# 4. Evaluation

We evaluate the impact of the fanout and proactiveness of a gossip protocol by deploying a streaming application based on Algorithm 1 over a set of 230 PlanetLab nodes. Our implementation is based on UDP and incorporates retransmission to recover lost stream packets (lines 14, 15, 25 in Algorithm 1).

**Bandwidth constraints.** PlanetLab nodes benefit from high bandwidth capabilities and therefore are not representative of peers with limited capabilities. We thus artificially constrain the upload bandwidths of nodes with three different caps: 700 kbps, 1000 kbps and 2000 kbps. To limit message loss resulting from bandwidth bursts, our bandwidth limiter also implements a bandwidth throttling mechanism.

**Streaming Configuration.** A source node generates a stream of 600 kbps and proposes it to 7 nodes in all experiments. To provide further tolerance to message loss (combined with retransmission), the source groups packets in windows of 110 packets, including 9 *FEC coded packets*. The gossip period is set to 200 ms.

**Evaluation metrics.** We assess the performance of the streaming protocol along two metrics: *(i)* the *stream lag*, defined as the difference between the time at which the stream is published by the source and the time at which it is actually delivered to the player on the nodes; and *(ii)* the *stream quality*, which represents the percentage of the stream that is viewable. A window is *jittered* if it does not contain enough packets (i.e., strictly less than 101) to fully reconstruct the window. A stream with a maximum of 1% jitter means that at least 99% of all the windows are complete.[1]

Stream lag and quality are correlated notions: the longer the acceptable lag, the better the quality. We consider stream qualities corresponding to several lag values as well as to an infinite lag, which represents the performance obtainable by a user that downloads the stream for playing at a later stage, e.g., offline viewing. Each experiment was run multiple times. We plotted the most representative one.

[1] An incomplete window does not mean that the window is unusable. Using systematic coding, a node receiving 100 out of the 101 original packets, experiences a 99% delivery in that window.

## 4.1. Impact of varying the fanout

We start our analysis by measuring how varying the fanout impacts each of the two metrics, with a proactiveness degree of 1 ($X = 1$). Results are depicted in Figures 1–3. Figure 1 shows the percentage of nodes that can view the stream with less than 1% jitter for various stream lags in a setting where all nodes have their upload capabilities capped at 700 kbps.

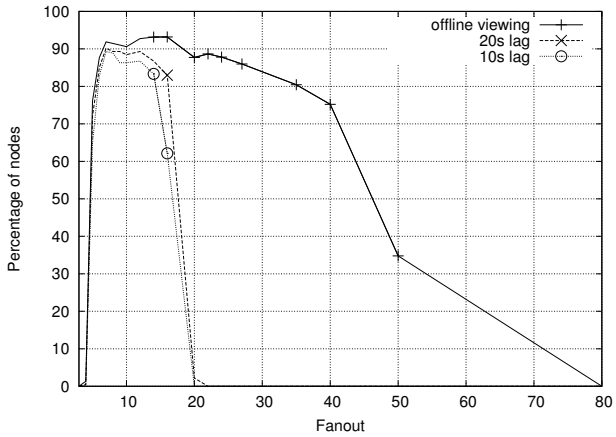


Figure 1: Percentage of nodes viewing the stream with less than 1% of jitter (upload capped at 700 kbps).

**Optimal fanout range.** The plot clearly highlights an optimal range of fanout values (from 7 to 15 in Figure 1) that gives the best performance independently of the lag value considered. Lower fanout values are insufficient to achieve effective dissemination, while larger values generate higher network traffic and congestion, thus decreasing the obtainable stream quality.

The plot also shows that while the lines corresponding to finite lag values have a bell shape without any flat region, the one corresponding to offline viewing does not drop dramatically until a fanout value of 40. The bandwidth throttling mechanism is in fact able to recover from the congestion generated by large fanout values once the source has stopped generating new packets. For fanouts above 40, on the other hand, such recovery does not occur.

**Critical lag value.** A different view on the same set of data is provided by Figure 2. For each value $t$, the plot shows the percentage of nodes that can view at least 99% of

the stream with a lag shorter than $t$. A fanout in the optimal range (e.g., 7) causes almost all nodes to receive a high quality stream after a critical lag value ($t = 5$ s for a fanout of 7). Moderately larger fanout values cause this critical value to increase ($t = 22$ s for a fanout of 20), while for fanouts above 35, no critical value is present. Rather congestion causes significant performance degradation. With a fanout of 40, only 20% of the nodes can view the stream with a lag shorter than 60 s, and a lag of 90 s is necessary to reach 75% of the nodes.

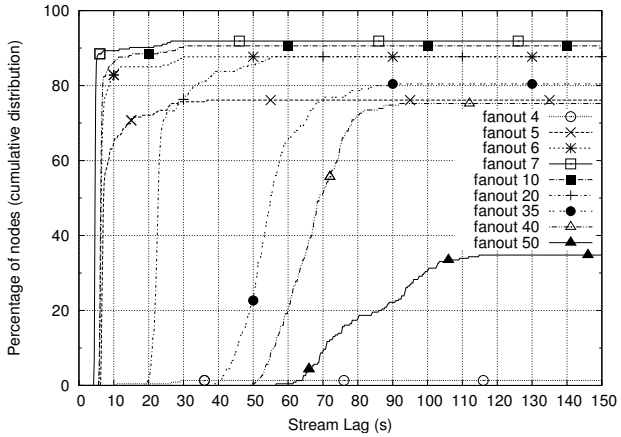


Figure 2: Cumulative distribution of stream lag with various fanouts (upload capped at 700 kbps).

**Behavior with less tight distributions.** The presence of an optimal fanout value is clearly a result of operation under limited bandwidth. Figure 3 complements the picture by showing how fanout affects performance under less critical conditions: 1000 kbps and 2000 kbps of capped bandwidth. As available bandwidth increases, the range of good fanout values clearly becomes larger and larger and tends to move to the right. With an available bandwidth of 1000 kbps, which is more than 1.67 times the stream rate, it is still possible to identify a clear region outside of which performance degrades significantly. With 2000 kbps of available bandwidth, both the 10 s-lag and the offline performance figures appear to remain high even with very large fanout values. This behavior may be better understood by examining how bandwidth is actually used by the PlanetLab nodes involved in the dissemination.

**Bandwidth usage with different fanouts values.** Figure 4 shows the distribution of bandwidth utilization over all the nodes involved in the experiments sorted from the one contributing the most to the one contributing the least. The plot immediately highlights an interesting property: even though all the considered scenarios have a homogeneous bandwidth cap, the distribution of utilized bandwidth is highly heterogeneous. This behavior is a direct result of the three-phase protocol employed for disseminating large content.

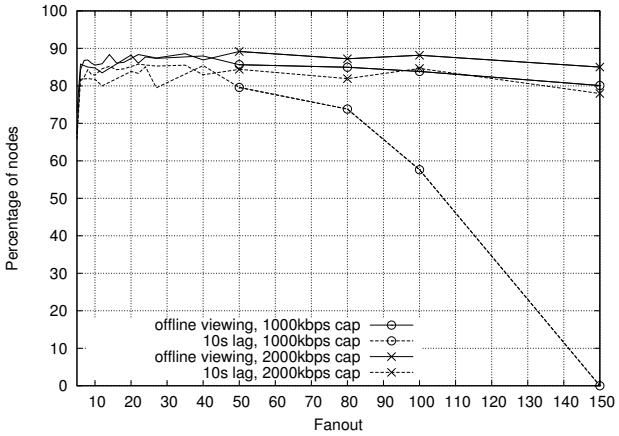


Figure 3: Percentage of nodes viewing the stream with less than 1% of jitter with upload caps of 1000 kbps and 2000 kbps, and different fanout values.

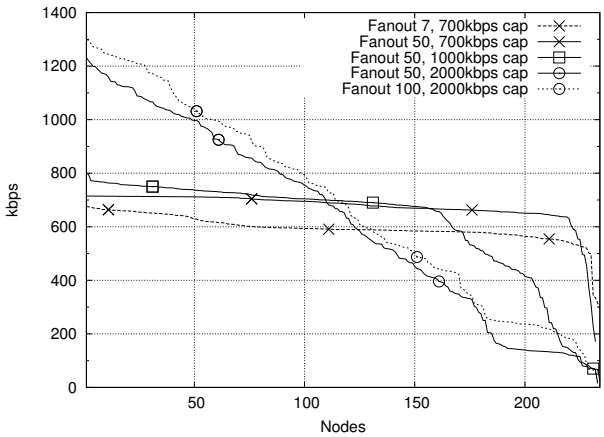


Figure 4: Distribution of bandwidth usage among nodes with different fanout values and upload caps.

According to Algorithm 1, all nodes contribute to the gossip dissemination by sending their proposal messages to the same number of nodes. However, the contribution of a node in terms of serve messages depends on the probability that its proposal messages are accepted by other nodes. In general, nodes with low-latency and reliable connections (i.e., *good* nodes) have higher probabilities to see their proposals accepted. This is confirmed by Figure 4. The plot also shows that the heterogeneity in bandwidth utilization increases with the amount of available bandwidth. For example the lines for the 700 kbps bandwidth cap show an

almost homogeneous distribution apart from a small set of *bad* nodes. This is because the higher latencies exhibited by good nodes when their bandwidth utilization is close to the limit causes other nodes to work more, thus equalizing bandwidth consumption. On the other hand, if we observe the lines corresponding to a fanout of 50 in the 1000 kbps and 2000 kbps scenarios and to a fanout of 100 in the 2000 kbps scenario, we see that good nodes have enough spare capacity to operate without saturating their bandwidths. As a result, the contribution of nodes remains highly heterogeneous.

## 4.2. Proactiveness

Next, we present our analysis of gossip proactiveness by showing how refreshing the set of communication partners affects application performance. Figure 5 presents the results obtained by varying the *view refresh rate*, $X$, in a scenario with a 700 kbps bandwidth cap. The plot shows three lines corresponding to stream lags of 10 s and 20 s as well as to offline viewing. In all cases, results confirm that the best performance is obtained by varying the set of gossip partners at every communication round. If on the other hand, the set of communication partners remains constant for long periods of time, a small set of nodes end up having the responsibility of feeding large numbers of nodes for as long as they keep being selected early in the dissemination process. This means that their upload rates remain constantly higher than their allowed bandwidth limits, ultimately resulting in high levels of congestion, huge latencies, and message loss.

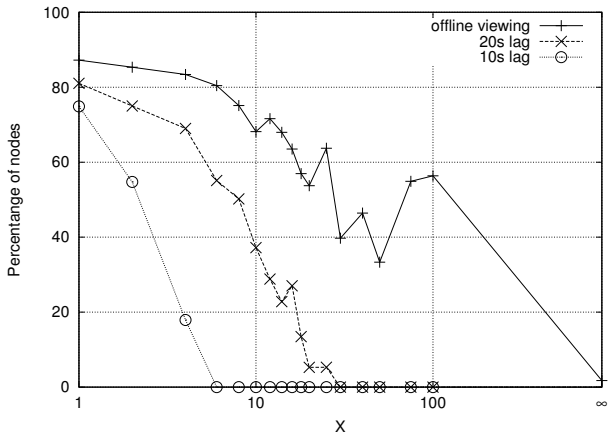


Figure 5: Percentage of nodes viewing the stream with at most 1% jitter as a function of the refresh rate $X$.

In accordance with these observations, Figure 5 shows that the slope at which the curves decrease with $X$ is most negative for a lag of 10s. This is because longer values of lag allow the bandwidth throttling mechanism more time to recover from the bursts generated by a constant set of communication partners. Nonetheless, a completely static dissemination mesh invariably yields bad performance even for offline viewing as the load becomes then concentrated on a very small set of nodes for the entire experiment.

**Requesting nodes to update their views.** A second way to modify the proactive behavior of the considered streaming protocol is for nodes to periodically request a new set of nodes to feed them, i.e., every $Y$ dissemination rounds. In Figure 6 we show the results obtained with different values of $Y$. Results show that this technique remains inferior to the simpler approach of choosing a *view refresh rate* of $X = 1$, as discussed above. The additional messages used by this approach may in fact be lost or delayed while the node is congested, resulting in a larger $Y$ than planned.

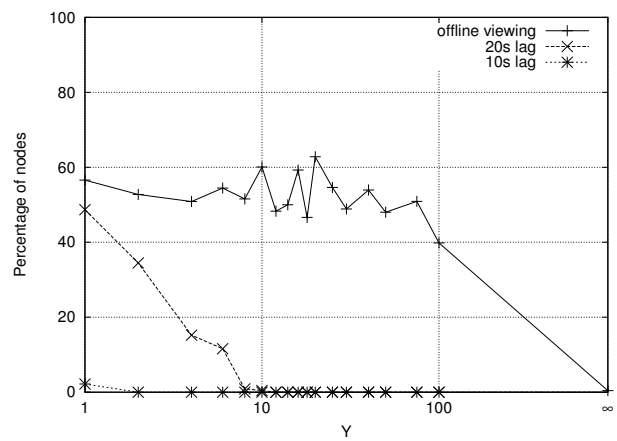


Figure 6: Percentage of nodes viewing the stream with at most 1% jitter as a function of the request rate $Y$.

## 4.3. Performance in the presence of churn

Finally, we evaluate the impact of proactiveness in the presence of churn with $Y = \infty$. Results are depicted in Figures 7 and 8. The experiments consist in randomly picking a percentage of nodes and make them fail simultaneously. We compare the baseline results (e.g., when no nodes fail) with those with increasing percentages, from 10% to 80%, of failing nodes. Figure 7 shows the percentage of remaining nodes experiencing less than 1% jitter after the catastrophic failure, for values of $X$ of 1, 2, 20 and $\infty$.

Figure 7 clearly shows that a completely dynamic mesh offers the best performance in terms of ability to withstand churn. With 35% churn, a proactiveness of $X = 1$ is able to deliver an unaffected stream to 60% of the remaining nodes, while the percentage drops to 32% for $X = 2$ and a stream lag of 20 seconds. The results obtained with large values of

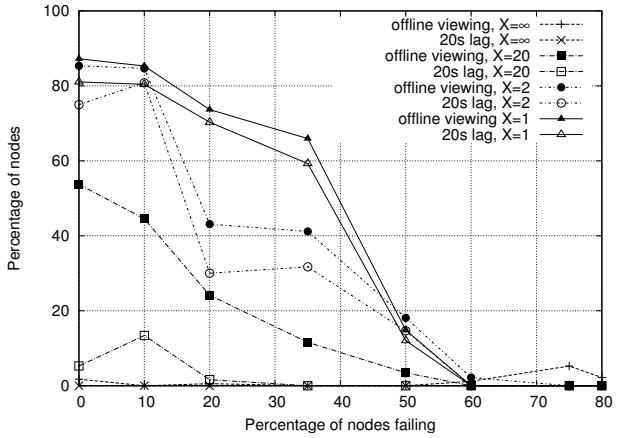


Figure 7: Percentage of surviving nodes experiencing less than 1% jitter for different values of $X$.

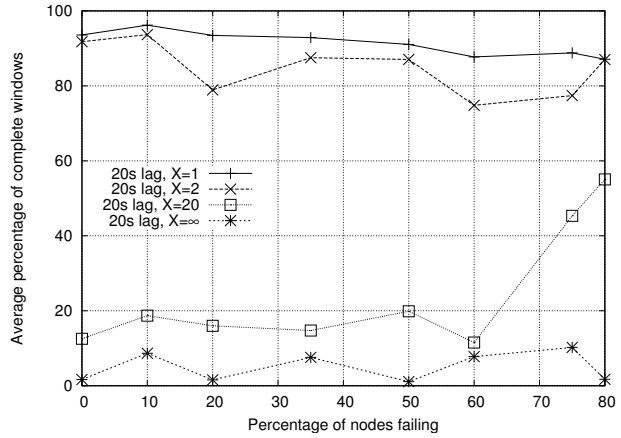


Figure 8: Average percentage of complete windows for surviving nodes.

$X$ and in particular when $X = \infty$ show very high degrees of variability from experiment to experiment. The resulting static or semi-static random graph may become completely unable to disseminate the stream if failing nodes are close to the source or it may appear extremely resilient to churn if failing nodes are located at the edge of the network. On average, performance is therefore in favor of a completely dynamic graph ($X = 1$).

It should be noted that Figure 7 only shows how many nodes manage to remain completely unaware of the churn event. To characterize the extent of the performance decrease experienced by all surviving nodes, Figure 8 shows the average percentage of decoded windows over the total number of windows streamed by the source. With $X = 1$, the protocol is almost unaffected by churn, and nodes correctly receive over 90% of the windows for all churn percentages lower than 80%. In addition, almost all the missing windows turn out to be concentrated in a time frame of 5 s to 10 s around the churn event when 20% and 80% of nodes fail (not shown in the plot).

## 5. Concluding remarks

The evaluations of gossip protocols for dissemination usually back up the associated theoretical claims: gossip can be tuned to achieve reliable dissemination in the presence of churn. In this paper, we challenged these results, obtained mostly through simulations in close-to-ideal settings, and evaluated a gossip-based live streaming application in a real deployment over 230 PlanetLab nodes.

Our results show that message loss and limited bandwidth significantly restrict the range of parameter values in which gossip can successfully operate. First, the fanout cannot be increased arbitrarily to improve reliability and latency, but must remain small enough to prevent bandwidth saturation. Second, the set of communication partners should be changed frequently, at every communication round, in order to minimize congestion and provide an effective response to churn.

## References


[1] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal Multicast. *TOCS*, 17(2):41–88, 1999.

[2] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg. Epidemic Live Streaming: Optimal Performance Trade-Offs. In *SIGMETRICS*, 2008.

[3] 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 *PODC*, 1987.

[4] M. Deshpande, B. Xing, I. Lazardis, B. Hore, N. Venkatasubramanian, and S. Mehrotra. CREW: A Gossip-based Flash-Dissemination System. In *ICDCS*, 2006.

[5] N. Drost, E. Ogston, R. van Nieuwpoort, and H. Bal. ARRG: Real-World Gossiping. In *HPDC*, 2007.

[6] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized Rumor Spreading. In *FOCS*, 2000.

[7] A.-M. Kermarrec, L. Massoulié, and A. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. *TPDS*, 14(3):248–258, 2003.

[8] 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 *INFOCOM*, 2008.

[9] H. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robinson, L. Alvisi, and M. Dahlin. FlightPath: Obedience vs. Choice in Cooperative Services. In *OSDI*, 2008.

[10] L. Massoulié, A. Twigg, C. Gkantsidis, and P. Rodriguez. Randomized Decentralized Broadcasting Algorithms. In *INFOCOM*, 2007.

[11] S. Sanghavi, B. Hajek, and L. Massoulié. Gossiping with Multiple Messages. *TIT*, 53(12):4640–4654, 2007.