Gossiping Capabilities

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

Gossiping Capabilities

Davide Frey[†] Rachid Guerraoui[*] Anne-Marie Kermarrec[†]
Martin Mogensen[‡] Maxime Monod[*] Vivien Quéma[§]

[*]EPFL, Lausanne, Switzerland
[†]INRIA Rennes-Bretagne Atlantique, France
[‡]University of Aarhus, Denmark
[§]CNRS, Grenoble, France

Technical Report LPD-REPORT-2008-010
May 8, 2008
last revision May 22, 2008

Abstract

Gossip-based protocols are now acknowledged as a sound basis to implement collaborative high-bandwidth content dissemination: content location is disseminated through gossip, the actual contents being subsequently pulled.

In this paper, we present HEAP, *HEterogeneity-Aware Gossip Protocol*, where nodes dynamically adjust their contribution to gossip dissemination according to their capabilities. Using a continuous, itself gossip-based, approximation of relative capabilities, HEAP dynamically leverages the most capable nodes by (a) increasing their fanouts (while decreasing by the same proportion those of less capable nodes) and (b) employing them early in the dissemination chain. These, on the other hand, have an incentive to take on additional load as being first in the chain improves their perceived quality. A lightweight accountability mechanism is used to track selfish nodes that might declare a high capability in order to augment their perceived quality without contributing accordingly.

We evaluate HEAP in the context of a video streaming application on a 236 PlanetLab nodes testbed. Our results shows that HEAP improves the quality of the streaming by 25% over a standard gossip protocol without impacting the average load or availability of the system.

1 Introduction

Gossip-based dissemination. Gossip-based protocols provide a reliable way to disseminate information in large-scale dynamic systems [3, 6, 12]. For this reason, they have recently been acknowledged as a sound basis to locate content in peer-to-peer streaming applications [33, 49].[1] In a gossip protocol, each node periodically forwards every stream-packet identifier it receives to a subset of nodes picked uniformly at random. These nodes may subsequently pull the content whenever necessary.

In most gossip-based protocols we are aware of, the dissemination load is evenly spread among all participating nodes. Specifically, all nodes usually have the same number of gossip targets (fanout) and the same dissemination period, and therefore send exactly the same

[1]In contrast, even though trees are often seen as the ultimate efficient way of dissemination, they are not very robust in large-scale dynamic settings. For instance, a perfectly structured tree for disseminating a stream, over 30 runs on PlanetLab, in all runs, a significant number of nodes were not able to view the stream due to the static nature of trees exacerbating the loss rate of UDP packets.

number of messages. However, this uniform distribution of load ignores the heterogeneous nature of large-scale distributed systems where nodes may exhibit significant differences in their capabilities.

Such heterogeneity in capabilities may be the result of heterogeneity in computational power, i.e., small devices versus powerful machines, in network connections, parts of the network might be lossy and alter the communication between certain peers, or in actual host applications with some nodes running network-greedy applications while others have full bandwidth available. As an example, while the penetration of the Internet into homes has greatly favored the deployment of p2p collaborative systems, bandwidth capability of edge nodes connected through ADSL links remains extremely heterogeneous. This is particularly true for the upload capacity which is of utmost importance for collaborative systems. For instance, the average upload capacity of ADSL connections in Europe is 1077kbps, while it is 899kbps in North America and only 170kbps in Africa [1]. As we confirm later in the paper, a load-balancing strategy oblivious to such discrepencies may significantly hamper the reliability of the dissemination of, say, a South-American soccer game to Asia, Europe, and Africa.

As in [10, 49, 43, 31, 46, 47, 7], we recognize the need to account for the discrepencies between peers and we leverage them to achieve a more effective dissemination process. Doing this in the context of a gossip protocol presents important technical challenges. First, capabilities can clearly change over time and an effective dissemination protocol needs to dynamically track and reflect their changes. Second, the reliability of gossip protocols heavily relies on the uniformly random selection of target peers: biasing this selection and hurting uniformity could impact the average quality of gossip dissemination.

In short, we address these challenges using two observations about the behavior of gossip dissemination protocols. First, mathematical results on epidemics and empirical evaluations of gossip protocols [13, 39] convey the fact that the reliability of dissemination is ensured as long as the *average* of all fanouts is approximatively $\ln(n)$ (in a system of size n). This is crucial because the size of the fanout is an obvious knob to adapt the contribution of a node. 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 from more (resp. less) often. Hence, the observation about the ideal fanout being of size $\ln(n)$ on average opens a clear opportunity for adaptation. Second, and this is a general characteristic of large-scale dissemination protocols, the utility of messages is higher in the *first* gossip rounds. Basically, the capability of a node intervening in the first gossip rounds can have a dramatic impact on the overal reliability of dissemination [27]. This is particularly crucial because such a node is likely to frequently be pulled for new packets. Usually, the latter rounds simply impose more redundancy to ensure that all nodes are eventually covered.

Based on these observations, we propose a new gossip protocol, which we call *HEAP* (*HEterogeneity-Aware Gossip Protocol*). HEAP is specifically aimed for collaborative, high-bandwidth content distribution. It relies on a gossip-based aggregation protocol [22, 44] to have each node continously assess its relative capability in order to implement the following mechanisms.

1. **Fanout adaptation:** each node updates its fanout to match its relative capability. We perform this adaptation in such a way that, whenever some node decreases (resp. increases) its fanout by a proportion x, d nodes increase (resp. decrease) their fanout by a proportion x/d. This strategy is precisely devised to ensure the average fanout value persists over time without decreasing under the fatal $\ln(n)$ threshold and without unnecessarily increasing and impacting the overall load on the system.

2. **Biased gossip target selection:** each node biases the selection of gossip targets so

that the position of nodes in the dissemination process matches their capabilities. This is achieved by taking into account the hopcount of any given message so that the early stages of the dissemination process involve the most capable nodes. These nodes have an incentive to take on the resulting additional load as being first in the dissemination chain improves their perceived quality (as we show through our experiments in Section 5). To preserve enough uniformity in the random selection and ensure global reliability, we apply a sliding window technique where less capable nodes have better chances to be selected in subsequent rounds.

A companion sub-protocol, which we call LIFT *(LIghtweight Freerider Tracking*), is also provided to track selfish nodes that might declare a high capability in order to augment their perceived quality, without contributing accordingly. While interesting in its own right, LIFT is particularly useful in the context of HEAP. A node that declares a high capability will, in fact, end up among the first in the dissemination chain and thus benefit from a higher gossip reliability (Section 5). However, such a node is also expected to contribute more (larger fanout) and it may negatively impact the overall reliability if it decides not to do so (Section 5). Our LIFT protocol (a) requires nodes to record the history of the nodes they sent packets to and (b) involves an audit procedure in which a node can potentially be confronted with others in order to check if it behaves as it is supposed to. These characteristics allow LIFT to achieve particularly effective results with only negligible overhead.

We evaluated HEAP in the context of a video streaming application on a 236-PlanetLab-node testbed. We used upload capacity as a capability metric. Specifically, we considered nodes with asymmetric bandwidth and heterogeneous upload capacities. Upload bandwidth is in fact a crucial parameter for collaborative applications and may vary not only as a result of the limits imposed by the provider but also depending on the overall bandwidth consumption in a given geographical area.

Our results show that, HEAP improves stream quality by 25% with respect to standard *homogeneous* gossip, while generating a very similar load.[2] We also show that HEAP ensures a dynamic adaptation of the contribution of nodes that matches the distribution of capabilities across the system; we report on the negligible overhead of its underlying capability aggregation protocol; and we measure the impact of the capability of a node on the stream quality it obtains through HEAP as well as the amount of its contribution. Finally, we report on the negligible overhead of LIFT and measure the impact of a selfish node based on its degree of freeriding, as well as the relation between the number of LIFT audits performed, the degree of freeriding and the probability of detecting selfish nodes.

Roadmap. The rest of the paper is organized as follows. Section 2 gives some background on traditional gossip-based dissemination and provides an overview of HEAP. Sections 3 and 4 describe HEAP and LIFT in details. We present our streaming application and reports on the results of our experiments on PlanetLab in Section 5. Related work is covered in Section 6. Some concluding remarks are given in Section 7.

2 HEAP design overview

In this section, we present an overview of HEAP, our gossip-based protocol for collaborative high-bandwidth content distribution. We first provide some background on standard gossip-based protocols, then we discuss the rationale behind our approach. Finally, we present an

[2]Homogeneous means that all nodes contribute the same amount.

overview of the components of the HEAP and companion LIFT protocols. Section 3 elaborates on the details.

2.1 Background on gossip-based protocols

Gossip-based dissemination protocols. Most such protocols follow some variant of the skeleton illustrated in Algorithm 1, which HEAP itself extends. In the context of collaborative high-bandwidth content distribution, typically content location is pushed using a gossip protocol and the content is subsequently pulled. This reduces redundancy on large messages.

Consider a set of n nodes $\Pi = \{p_1, \ldots, p_n\}$, 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 and additional data for error correction.

The gossip protocol works as follows. Nodes periodically contact a fixed number f of nodes chosen according to the selectNodes() function and propose to them a set of event identifiers (ids) with a [Propose] message (lines 6-7 & 26-28 in Algorithm 1).

The selectNodes(f) function is typically returning a set of f nodes, chosen uniformly at random in Π. The fixed number f is called the *fanout* and is a key parameter of gossip-based dissemination protocols. It can be tuned to trade reliability against overhead. Typically, the larger the fanout, the more reliable the dissemination [5, 26, 30, 39]. As discussed in the introduction, a large fanout may impose a large overhead over the system and introduce unnecessary redundancy. A value of the fanout of $\ln(n)$ is typically considered ideal to ensure reliable dissemination with small resource consumption [13].

Each node maintains a list (eventsToPropose) containing the identifiers of the events it has delivered since its last gossip. A node may deliver an event either when publishing a new one (line 4 in Alogirthm 1), or when retrieving it (i.e., receiving a [Brief] message, lines 14-17).

There are basically two ways to gossip events: *(i)* when a new event e is produced, it is gossiped asynchronously, i.e., immediately sending a [Propose] message, which contains only a single event e, to f nodes (line 5 in Algorithm 1), and *(ii)* periodically whenever the GossipTimer triggers (line 6 in Algorithm 1), which period is given by the gossipPeriod parameter.

Upon reception of a gossip message containing event ids, a node pulls the content it has not yet retrieved by sending a [Request] message to the sending peer. A peer being pulled then sends back the actual content in a [Brief] message that contains the events which ids were requested in the received [Request] message.

Gossip-based membership protocol. Gossip protocols typically assume that each node is able to sample uniformly the system when renewing the f gossip targets. This is key to the reliability of the dissemination (function selectNodes() in Algorithm 1). In large scale infrastructures, where nodes maintain only a partial view of the system, this can be achieved through a peer-sampling protocol [4, 12, 23, 48]. Many variants exist which mostly differ in *(i)* the selection of the gossip target *(ii)* the data exchanged over gossip and *(iii)* the processing of this data.[3]

2.2 HEAP design rationale

The design of HEAP stems from two observations on gossip protocols. First, most gossip dissemination protocols assume a *homogeneous* fanout [3] even though theory reveals that the reliability of gossip dissemination is actually preserved as long as an $O(\ln(n))$ fanout value

[3]Note that gossip-based dissemination protocols also follow this canvas [3].

Algorithm 1 Basic gossip protocol

Initialization:
1: $f := \ln(n) + c + \mathcal{O}(1)$ {f *is the fanout*}
2: eventsToPropose := eventsDelivered := requestedEvents := $\varnothing$
3: **trigger** ⟨ **start**(GossipTimer(gossipPeriod)) ⟩

upon event ⟨ publish | e ⟩ **do** {*The application layer uses this primitive every time a new event e is produced and needs to be broadcast*}
4: deliverEvent(e)
5: gossip($\{e.id\}$, f)

upon event ⟨ fire | GossipTimer ⟩ **do**
6: gossip(eventsToPropose, f)
7: eventsToPropose := $\varnothing$ {*Infect and die*}

upon event ⟨ receive | [PROPOSE, eventsProposed] ⟩ **do**
8: wantedEvents := $\varnothing$
9: **for all** $e.id \in$ eventsProposed **do**
10: **if** ($e.id \notin$ requestedEvents) **then**
11: wantedEvents := wantedEvents $\cup$ $e.id$
12: requestedEvents := requestedEvents $\cup$ wantedEvents
13: **trigger** ⟨ reply | [REQUEST, wantedEvents ⟩

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

upon event ⟨ receive | [BRIEF, events] ⟩ **do**
18: **for all** $e \in$ events **do**
19: **if** ($e \notin$ eventsDelivered) **then**
20: eventsToPropose := eventsToPropose $\cup$ $e.id$
21: deliverEvent(e)

function selectNodes(f) **returns** set of nodes **is**
22: **return** f uniformly random chosen nodes in Π

function getEvent(event id) **returns** event **is**
23: **return** the event corresponding to the id

procedure deliverEvent(e) **is**
24: deliveredEvents := deliveredEvents $\cup$ e
25: **trigger** ⟨ deliver | e ⟩ {*The event e is given to the application layer*}

procedure gossip(event ids, fanout) **is**
26: communicationPartners := selectNodes(fanout)
27: **for all** p $\in$ communicationPartners **do**
28: **trigger** ⟨ send | p, [PROPOSE, event ids] ⟩

is ensured *on average* [29], regardless of the actual fanout distribution across nodes.[4] HEAP leverages this property to adjust the fanout distribution so that *(i)* the average fanout remains close to $\ln(n)$, and *(ii)* the fanout values match the capability distribution. In other words, starting from a homogeneous initial fanout, we thus *heterogeneize* the fanout among nodes. We call this the *fanout adaptation protocol*.

The second key observation relates to the importance of nodes which are involved early in the dissemination chain.[5] In (push-based) protocols, the number of nodes that receive the message grows exponentially during the first rounds of gossiping. Afterwards, the set of uninformed

[4]Obviously, in an uneven distribution of fanout values, the failure of a node having a large fanout may have a larger impact on dissemination.

[5]Note that this observation is also valid for tree-based approaches.

processes only shrinks by a constant factor. This conveys the fact that the *utility* of a message decreases with the number of hops it travels. This has been leveraged in [27], where the authors propose a push-based dissemination until $n/\log n$ nodes have the message with high probability (*exponential growth* phase). The protocol then switches to a pull (*quadratic shrinking*) phase as the probability to reach uninformed nodes dramatically decreases. This shows that the nodes gossiping messages early are of the utmost importance for an early spreading. In addition, in the context of high-bandwidth content distribution where nodes pull the actual content, such nodes are the first to advertise new data and will be pulled proportionally more often. This suggests that to achieve efficient content dissemination, such nodes should be capable of sustaining high load. HEAP leverages these observations and biases the selection of the gossip targets toward the most capable nodes at early stages: we call this component of HEAP the *biased gossip target selection*.

In short, a HEAP node *(i)* takes into account its own capability (and its variations over time) by dynamically adjusting its own fanout, and thus directly impacting its own contribution, and *(ii)* accounts for capabilities of other peers by biasing the target peer selection for gossip dissemination, directly impacting the contribution of other peers.

Both these operations must be carried out with care so as not to hurt the reliability of the protocol. In the following, we describe how this is achieved in the HEAP protocol.

HEAP fanout adaptation. To get a sense of the underlying challenge, consider upload bandwidth as a measure of capability. A naive approach would simply consist for each node to periodically measure its capability and increase (resp. decrease) its fanout proportionally to the increase (resp. decrease) of its bandwidth - since the last measure. This approach has two major limitations. First, it could easily lead to a significant drop in the reliability of the system if all nodes see their upload capacity progressively diminish, even by a small proportion in each period. Eventually, all fanouts will end up significantly lower than the theoretical $\ln(n)$ threshold, impacting the reliability of the dissemination. Second, the unilateral decision of how to increase or decrease the fanout can lead to extreme situations of unfairness.

Instead in HEAP, nodes dynamically adjust their fanout according to the capability distribution of the nodes. To ensure that the contribution of a node matches it capability, each node needs to have an idea of its own position across nodes, capability-wise. To this end, we introduce a gossip-based capability aggregation protocol in which nodes periodically send their capability values to their neighbors. Periodically, each p_i measures its capability, it normalizes it according to its local approximation of the global capacity, and it computes and then adopts a new target fanout. A major challenge towards achieving this goal is to prevent these adaptations from jeopardizing the overall reliability of a gossip dissemination. Effectively, a homogeneous fanout across nodes obviously limits the impact of the failure of a subset of the nodes on the reliability of the dissemination. HEAP ensures that *(i)* the fanout average remains close to $\ln(n)$, *(ii)* the fanout distribution matches the capability distribution, while the average fanout is kept accross nodes.

HEAP biased gossip target selection. It is clear that nodes involved early in the dissemination chain are of the utmost importance. One of the originality of HEAP is to place the most capable nodes in the beginning of the gossip dissemination. This ensures both that capable nodes be prevalent in the gossip phase and that they be proportionally more requested to provide the content. Achieving such an adaptation is challenging for two reasons. First, biasing the gossip target selection is somewhat antagonist with the uniform random selection of gossip targets, which is key to the reliability of traditional gossip approaches. Second, placing such nodes early in the dissemination chain, not only improves the overall efficiency of the system

but also provides those nodes with an improved streaming quality. This gives an incentive to collaborate, but might as well give rise to freerider vocations. This latter issue is actually both more probable to happen and more harmful than in a standard gossip protocol.

To tackle this challenge, we propose a companion collaborative *LIghtweight Freerider Tracking protocol*, called LIFT. LIFT periodically audits randomly chosen nodes to verify if their behavior matches the expectations based on their capabilities. Basically, LIFT tracks down selfish nodes, that announce themselves as capable nodes by declaring a high capability to take advantage from the biased gossip target selection mechanism of HEAP. Such nodes clearly benefit from being placed early in the chain, but they significantly harm the whole system if they do not contribute as they should, that is by participating in the dissemination process according to their capabilities.

3 HEAP description

In this section, we provide the details of HEAP. As already explained, HEAP can be seen as the combination of *(i)* a standard gossip protocol as the one depicted in Algorithm 1, *(ii)* a fanout adaptation mechanism, *(iii)* a biased gossip-target selection, and *(iv)* a gossip-based capability-aggregation mechanism. In this section, we detail the three latter ones, the standard gossip protocol having been already presented in the previous section.

System model. We assume an unstructured overlay network of n nodes, built using a gossip-based peer sampling protocol; such protocols are known to achieve topologies close to those of random graphs. As this is not the purpose of the paper, we do not detail them here; the interested reader can refer to [24]. Each node in a peer-sampling protocol maintains a view of c nodes which represent a random sample of the network. The gossip partners are selected from this view. Nodes gossip their capability along with their IP addresses, so that each node knows the capability of its partners. We assume that each node has a large enough sample to make statistical inference on the population of the network.

Capability aggregation mechanism. The pseudocode of the capability aggregation protocol is depicted in Algorithm 2. Each node periodically (i.e., every capPeriod time units) gossips the latest capabilities it has received (including its own) to capFanout communication partners. Upon receipt of such information, the capability values are aggregated into $\mathsf{nodesCap}_{local}$ to provide each node with information about the capabilities of other nodes.

Algorithm 2 The capability aggregation gossip protocol.

Initialization:
1: capFanout $:\simeq \ln(n)$
2: cap_{local} := local node's capability
3: $\mathsf{nodesCap}_{local}$:= set of w capabilities
4: **trigger** ⟨ **start**(CapGossipTimer(capPeriod)) ⟩

upon event ⟨ fire | CapGossipTimer ⟩ **do**
5: values := $\mathsf{cap}_{local} \cup k$ most recently received capability values
6: communicationPartners := selectNodes(capFanout)
7: **for all** p ∈ communicationPartners **do**
8: **trigger** ⟨ send | p, [CapAgg, values] ⟩

upon event ⟨ receive | [CapAgg, values] ⟩ **do**
9: $\mathsf{nodesCap}_{local}$:= $\mathsf{nodesCap}_{local} \cup$ values

Fanout adaptation mechanism. The pseudocode of the fanout adaptation mechanism is instead depicted in Algorithm 3. Nodes use the capability values collected in $\mathsf{nodesCap}_{local}$ to compute their fanouts for gossip dissemination, f_{target}, as follows: $f_{target} = \frac{cap_{local}}{cap_{avg}} \cdot f_{init}$.

This allows each node to match its fanout with its capability ratio. There is however one exception. Nodes may also specify a maximum fanout value, f_{max}, chosen so that it is always possible, in the worst case (in terms of bandwidth and latency) to exchange information with every set of f_{max} nodes within less than the duration of a round (i.e., gossipPeriod). A node that has $f_{target} > f_{max}$ does not increase its fanout further than f_{max} but sends *compensation* messages to the *poorest* nodes it is aware of.

It is important to observe that we recompute a new target fanout periodically, at every adaption phase, based on the collected capability values as well as on the initial fanout $f_{init} \simeq \ln(n)$, and not based on the last computed target fanout. This prevents the average value of the fanout from eventually diverging from f_{init}.

Algorithm 3 The fanout adaptation mechanism.

Initialization:
1: $f := f_{init} :\simeq \ln(n)$
2: **trigger** ⟨ start(FanoutAdaptationTimer(faPeriod)) ⟩

upon event ⟨ fire | FanoutAdaptationTimer ⟩ **do**
3: $f_{target} := \frac{\mathsf{cap}_{local}}{\mathsf{cap}_{avg}} \cdot f_{init}$
4: $f := \min(f_{target}, f_{max})$ {*The fanout is adopted*}
5: **if** $(f_{target} > f_{max})$ **then**
6: $nbComp := f_{target} - f_{max}$
7: send $nbComp$ [CompensationOrder] to the less capable nodes

upon event ⟨ receive | [CompensationOrder] ⟩ **do**
8: **if** $(f \geq f_{max})$ **then**
9: forward | [CompensationOrder] to one of the poorest nodes
10: **else**
11: $f := f + 1$

function @override selectNodes(fanout) **returns** set of nodes **is**
12: **return** fanout random chosen nodes which capability $< cap_{local}$

Biased gossip target protocol. In standard gossip protocols, nodes are picked uniformly at random from the local membership view. On the other hand, HEAP selects gossip targets according to their capabilities. The objective is to favor the nodes with the highest capabilities during the first hops of the dissemination and to select nodes with lower and lower capabilities at later hops. HEAP also strives to maintain a reasonable level of uniformity in the selection of nodes to preserve the reliability property of gossip dissemination. Instead of ensuring uniformity at each hop, HEAP aims at ensuring it across all hops, while introducing a bias toward specific hops for each node depending on its capability value. Nodes may also be selected with a low probability at other hops.

In order to know at which hops each node should benefit from a bias in the peer selection, we divide the set of nodes into slices according to their capability values. To achieve this, we consider a message being disseminated in an ideal tree, with fanout f_{init}, built by adding nodes in a breadth-first fashion in descending order of their capability values. In such a tree, the f_{init} nodes with highest capability values receive the message at the first hop, the second f_{init}^2 receive it at the second, the subsequent f_{init}^3 at the third and so on: dissemination terminates in h_{tree} hops.

In principle, each node should slice its view according to the expected number of hops in this ideal tree. The first slice should therefore contain the top f_{init} nodes, the second should

contain the second f_{init}^2 nodes, and so on. In practice, however, each node only has a partial view of the network that is smaller than the size of the system. For this reason, it scales down the size of each slice according to the ratio between the size of the system and the size of its view,[6] while guaranteeing that each slice contains at least f_{target} nodes.

Nodes use slices during gossip-based dissemination to select their gossip targets based on the hop count of the messages they need to disseminate. Because, in general, each gossip message is a collection of event ids with different hop counts, the hopcount of a message is determined as the average hop count over all the events ids it contains.

The publisher of the stream initiates the dissemination by selecting its targets from $slice_0$, that is the slice containing the f_{target} nodes with the highest capability values. In subsequent hops, a node at hop i in the dissemination selects its gossip targets from the set of slices between $slice_0$ and $slice_i$. As shown in Section 5, this simple protocol biases the peer target selection whithout hampering too significantly the uniformity of the protocol, and thus without hurting the reliability of gossip-based dissemination.

4 LIFT

LIFT is a protocol aimed at tracking selfish nodes in a collaborative high-bandwidth content dissemination system. Although LIFT could be used in other settings, it is particularly interesting in the context of HEAP. Purely selfish nodes do not want to harm the system deliberately but only to take the most out of it while contributing as little as possible. However, as we explained previously, HEAP may both favor and be harmed by nodes with such selfish behaviors. Focusing on selfish behavior only, as opposed to BAR Gossip [33] or PeerReview [19] which deal with Byzantine behavior and provide strong guarantees, allows LIFT to operate without using any expensive key-based mechanism.

The LIFT protocol allows a node – the *audit node* – to periodically suspect other nodes and check if they are operating as they are supposed to, or if they exhibit selfish behaviors. Nodes having a high capability and lying about it punish themselves without impacting the performance in HEAP. Therefore, we focus on nodes declaring a high capability to be placed early in the dissemination chain. Such nodes may for instance be either complete freeriders and contribute nothing at all or liars (about their capability). Liars may contribute as much as they can, yet the discrepency with their expected contributions (based on their declared capabilities) may impact the overall efficiency of the system.

As pointed out, we focus here on the detection of selfish nodes. Properly punishing them requires further investigation and is left for future work. Each node logs *when* it sent a [BRIEF], *which* events were in the [BRIEF] and to *whom* it sent it. Additionally, each node must be able to confirm if it did receive a brief from a given node. This is done by keeping track of the requests sent. For presentation simplicity, we only consider one audit node and one audited (suspected) node. The algorithm for the audit node is shown in Algorithm 4 while the one for the audited nodes is in Algorithm 5.

The audit node suspects another node by requiring its short-term gossip history ([GMYH] message in Algorithm 4). The audited node replies with a history containing the identifiers of both the events it sent over the last period and of their recipients. The size of the reply message ([HIMH] message in Algorithm 4) is constrained to contain at most 10KB of data and at most 10s of past history to bound the associated overhead.

It may happen that the audited node does not seem to reply to an audit if: (a) one of the messages got lost ([GMYH] or [HIMH]) or (b) the audited node fakes message losses by

[6] We assume that each node has a large enough view to make this scaling possible.

not replying to the audit. Therefore, if the audit node does not receive any reply within a reasonable delay (e.g., dNoResp, typically 5s), it keeps sending up to nbOfNoResp other audit requests. After nbOfNoResp attempts, the audited node is considered non-responsive and its suspicion is confirmed.

Algorithm 4 LIFT - Audit node

Initialization:
1: suspectSet = Π
2: **trigger** ⟨ **start**(AuditTimer(auditPeriod)) ⟩

upon event ⟨ fire | AuditTimer ⟩ **do**
3: nodes := pick numberOfConcurrentAudits nodes in suspectSet
4: **for all** p in nodes **do**
5: **trigger** ⟨ send | p, [GMYH] ⟩
6: resuspect p in dNoResp seconds if no [HIMH] answer from p

upon event ⟨ receive | [HIMH, briefHistory] ⟩ **do**
7: **if** (briefHistory < 5s) **then**
8: resuspect in dSmallNoLog seconds
9: **else**
10: pick nbAckAsked random briefs b in briefHistory
11: **for all** b **do**
12: **trigger** ⟨ send | b.dest, [AckRqst, b.header] ⟩

upon event ⟨ receive | [Ack, header] ⟩ **do**
13: suspect := header.src
14: **if** (number of [Ack] received for suspect ≥ nbOfAckNeeded) **then**
15: **trigger** ⟨ trust suspect and resuspect in dResuspect seconds ⟩
16: **else if** (number of [NoAck] received for suspect ≥ nbOfNoAckNeeded) **then**
17: **trigger** ⟨ punish | suspect ⟩ {*The suspect is detected as a freerider and must thus be punished*}
18: **else**
19: **trigger** ⟨ resuspect suspect x times with same briefHistory if no decision can be made ⟩
20: **trigger** ⟨ send | header.src [GMYH] if no decision can be made after x tries with the same briefHistory ⟩

When the audit node receives a [HIMH], it randomly picks nbAckAsked supposedly sent [Brief] messages from the received history and selects their recipients as witnesses for the audited node. The audit node, then contacts the witnesses with a [AckRqst] to verify that they indeed received the [Brief] messages as declared, and then waits for them to respond with [Ack] or [NoAck].

Based on these replies, the audit node produces a verdict on the selfishness of the audited node. If the audit node receives nbAckNeeded [Ack] messages, it decides that the audited node is correct and reschedules it for suspicion in dResuspect seconds. On the other hand, if the audit node receives nbNoAckNeeded [NoAck] messages, it detects the audited node as a freerider. As pointed out earlier, punishing detected freeriders requires further investigation.

As [AckRqst], [Ack] and/or [NoAck] messages may get lost, the audit node, after a delay dAck, attempts again to give a verdict based on the [HIMH] received. After nbOfTriesWithSameHIMH attempts, it asks the audited node again for another [HIMH] and keeps running the above procedure until a decision can be reached. Note that the verification is done in priority with the same briefHistory to lower the traffic overhead and also to prevent the audited node from suddenly behaving perfectly correctly.

5 Evaluation

We evaluated HEAP on a streaming application with a single source and up to 236 PlanetLab nodes subscribed to the stream.

In this section, we first describe the experimental setting as well as the considered applica-

Algorithm 5 LIFT - node.

upon event ⟨ receive | [GMYH] ⟩ **do** {*Give Me Your History*}
1: briefHistory := the last historySize briefs sent {*historySize is chosen such that the history* = $\min(10s, 10KB)$}
2: **trigger** ⟨ reply | [HIMH, briefHistory] ⟩ {*Here Is My History*}

upon event ⟨ receive | [AckRqst, header] ⟩ **do**
3: **if** self received a brief corresponding to header **then**
4: **trigger** ⟨ reply | [Ack, header] ⟩
5: **else**
6: **trigger** ⟨ reply | [NoAck, header] ⟩

tion. We then provide a detailed evaluation of the behavior of HEAP as well as a comparison with standard (homogeneous) gossip protocols with different fanouts. Unless stated otherwise, HEAP runs with an initial fanout of 6. We run homogeneous gossip with a fanout of 6, 8 and 10. We evaluated HEAP on a streaming application with a single source and up to 236 PlanetLab nodes subscribed to the stream. We also report on the accuracy of the LIFT protocol. We finally show that the capability aggregation and the fanout adaptation protocols provide accurate results in dynamic environments.

In short, we demonstrate that HEAP matches the distribution of the node dissemination load and the distribution of the node's perceived quality to the distribution of their capability, without impacting the reliability and without any noticeable overhead.

5.1 Experimental setting

Video streaming application. We evaluated HEAP in the context of a video streaming application. We based the experiments on the parameters given in [33]. We stream video at the rate of 234 Kbps so that a source initiates a broadcast of 30 events per second. Each event has a size of 1Kbyte and a brief contains a maximum of 10 of those events. The events have an associated time to live value (TTL) of 10 seconds during which they are valid. Once a TTL expires, the event is simply dropped and no longer gossiped. We use 5 streams in each experiment.

Experimental testbeds. We deployed the video streaming application on over 750 PlanetLab machines located all over the world on which we were able to run our streaming application on up to 236 nodes at the same time. The Planetlab deployment provides us with a real wide-area challenging environment.

PlanetLab and network capabilities. Because the PlanetLab nodes are situated mostly in research and education places, they benefit from a very good network quality, namely incoming and outgoing bandwidth (not to mention latency). As such, PlanetLab is not representative of a typical collaborative p2p system [41], in which most nodes would be end user typically behind ADSL connection, with an asymetric bandwidth and limited upload/download capacities. For mapping a scaled-down approximation of internet nodes onto the PlanetLab network, we decided to artificially limit the upload and download capacity of nodes so that they match the bandwidth usually available for home users, namely 56K, ISDN, (A)DSL and cable. To this end, we used internet usage statistics per continent [2] to have a realistic distribution of nodes across different continents and then limited the bandwidth according to the distribution of bandwidth in each of the different continents [1].

HEAP configuration. As previously mentioned, HEAP requires each node to be provided with a sufficiently large sample to be able to bias the target selection for instance. Given the limited scale provided by Planetlab testbed, we assumed a global membership. Also, we considered a static distribution of capabilities. Note that the accuracy of the aggregation protocol is provided through another experiment, with dynamically changing capabilities.

Evaluation metrics. We compare HEAP against an homogeneous gossip with various fanout along the following metrics: *(i)* the overall streaming quality of the system, *(ii)* the distribution of the streaming quality according to node capability, reflecting the ability of HEAP to optimize the dissemination by exploiting the capability of the most capable node, and *(iii)* the traffic generated for the dissemination of the stream. This shows that HEAP does not generate overhead over homogeneous and a better utilization of bandwidth among nodes.

We also report on the impact of selfish nodes in HEAP and show the ability of LIFT to detect such nodes. We provide figures on the precision and speed of selfish node detection, the percentage of false positives and the overhed of the protocol.

5.2 HEAP versus homogeneous gossip

Streaming quality. The streaming quality is computed as follows: we consider a sliding window, corresponding to 1s of streaming. For each position of the window over the whole stream, we compute the number of nodes that received at least 92% of the streamed packets. This value is assumed to be the lower bound to be able to watch a stream [33]. The values represented are the average over all windows.

Figure 1 depicts the distribution of the values of this metric over the nodes. The nodes are sorted by increasing capability and grouped in chunks of 10% nodes. Each value represents the percentage of nodes (averaged in the set of 10% nodes) that received at least 92% of the streamed packet using a sliding window size of 1s.

Several observations can be made on this figure. First, HEAP outperforms all homogeneous gossip configurations, including those with a higher average fanout than HEAP. The figure reveals that HEAP provides a better streaming quality for all nodes regardless of the capability. Many nodes with the highest capability get a good stream quality with HEAP. (This actually shows that HEAP provides an incentive for selfish nodes to be placed early in the dissemination, for instance.) In addition, HEAP dramatically improves the quality of low capability nodes. While only a few percent of such nodes get a good streaming quality with homogeneous gossip and a fanout of 6, HEAP achieves approximatively 55%. We also provide the quality achieved by the biased gossip target selection only and the fanout adaptation only. Even though the results are better than homogeneous gossip, they are not as performant as HEAP. This conveys the fact that HEAP exploits the most capable nodes to significantly improve the quality of both the most capable nodes and the low-capability nodes. Finally, we observe that the streaming quality metric matches the capability distribution.

Figure 2 shows the total traffic generated by the three configurations of homogeneous gossip (fanout of 6, 8 and 10), HEAP, and each of biased target selection and fanout adaptation only. We observe that HEAP outperforms all other configurations and provides 77% of nodes with a good stream quality, while any configuration of homogeneous gossip is lower than 60%. While we observe a significant improvement, the percentages remain lower than with HEAP. This illustrates the fact that the strength of HEAP resides in combining the fanout adaptation and the biased gossip target selection.

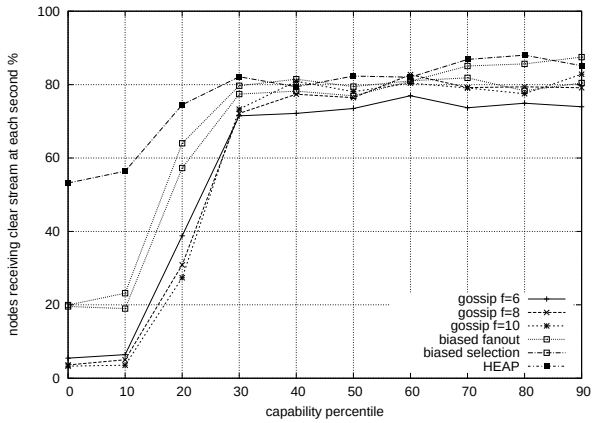

Figure 1: Quality vs capability: percentage of nodes receiving at least 92% of the stream for each segment of 10 capability percentiles.

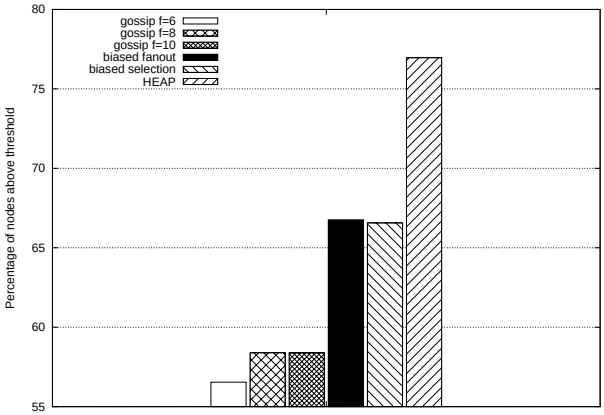

Figure 2: Overall quality: comparison of the total traffic

Contribution versus capability. Figure 3 shows the traffic according to the capability. We observe that in all homogeneous gossip configurations, the traffic is fairly homogeneous across all nodes except for the very low capability nodes. In biased gossip target configurations, we observe that the traffic matches the distribution of capability. This reflects a better bandwidth usage using HEAP. Again we observe that HEAP outperforms any of its two sub mechanisms used in isolation.

Figure 4 shows the overall traffic generated by all the protocols. Note that most of the traffic is not represented on the figure. We observe that the protocol generated with the biased gossip target selection performs best. What is clear from the plot is that increasing the fanout in homogeneous gossip significantly increases the total traffic. The total traffic is increased in HEAP which is consistent with the fact that HEAP provides a better availability.

5.3 LIFT performance

We first expose the key parameters of LIFT and the inheritent limitations to the simplicity of the protocol.

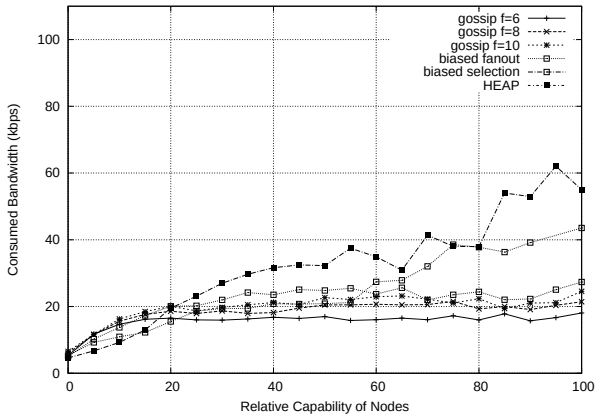

Figure 3: Bandwidth usage vs capabilities: average bandwidth utilization for each capability value

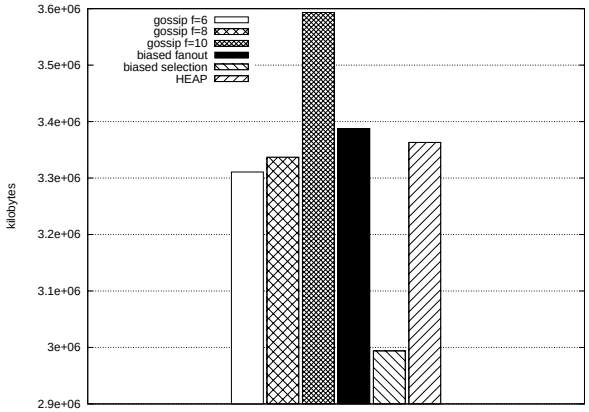

Figure 4: Total traffic generated by all the protocols

False positives. A false positive is when a node is detected freeriding when in fact it was behaving correctly. This can happen in two cases:

1. The suspected node tried to send all its payload ([Brief] message) and logged them as being sent correctly but $x \geq$ nbAckNeeded of them got lost in the communication channels. These x messages are thus sent with the history of briefs ([HIMH] messa ge) sent to the audit node. If the audit node asks for confirmation for those exact x briefs, the node will be detected freeriding and in fact was not.

2. Assume the suspected node s sent a brief b to a node r and the delay between the time it was sent (thus logged) and the reception is given as $d_{s,r}$. If the time taken for the audit node to ask for a briefHistory ([GMYH] message), receive it and contact r is smaller than $d_{s,r}$ then r will reply it did not receive b (which is on the way). The probability that this happens once is very small and the probability that this happens nbAckNeeded times is even smaller.

Non detection. A case of non detection is when a freerider can continue misbehaving without being detected by the audit node. This can happen when the freerider nodes sends $x \geq$

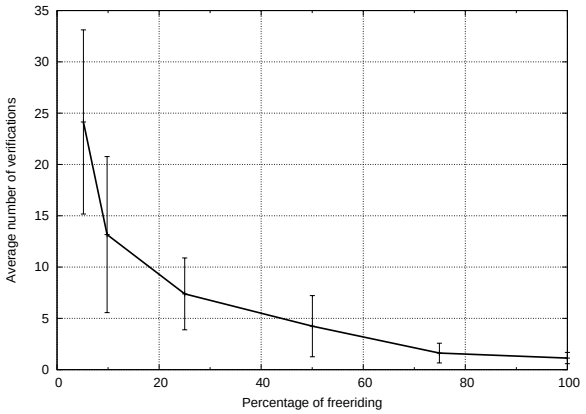

Figure 5: Freeriding percentage of a node vs the average number of verification for detection.

nbAckNeeded briefs but obviously not to all those it was supposed to in its past history. If the history is sent to the audit node and the audit node gets nbAckNeeded or more confirmations from the witnesses, the freerider has done enough work not to be detected. The probability of detection trivially increases as x decreases.

5.3.1 Detection Quality

We have experimented LIFT with nbAckAsked=3, nbAckNeeded=2 and nbNoAckAsked=3, meaning that for 3 [Ack]/[NoAck] replies, if 2 of them are positive, the audited node is disculped, if 3 replies are negative, the audited node is detected as a freerider.

Detection Precision. The number of false positives we have detected with these parameters is 15 out of 3777 different node verifications (each node getting verified 45 times on average), thus accounting for a false positive detection percentage as low as 0.397%.

The number of freeriders that we could not detect is 16, thus accounting for a non detection percentage as low as 0.424%.

Detection Speed. We have experienced different kinds of freerider, spanning from the very selfish one that never sends a brief to the very mild freerider that sends 95% of the time what it should. We thus define the percentage of freeriding of a node as the ratio between the number of briefs the node did not send over the number of briefs the node was supposed to send. With the given parameters Figure 5 shows that LIFT has no problem detecting fully freeriding nodes. The number ith of verifications needed to detect nodes grows as the freeriding percentage decreases, and even nodes with a freeriding percentage of 5% (i.e., that behave honestly 95% of the time) get detected, but with 24 verifications on average.

5.3.2 Overhead

For one verification, that is sending a [GMYH], receiving its [HIMH] response, sending nbAckAsked [AckRqst] and receiving nbAckAsked [Ack]/[NoAck], the total traffic usage is bounded by $12269+\mathsf{nbACKasked} \cdot \max(1166, 767)$ bytes.

The sizes of each individual LIFT messages are given in Table 1. The [Ack] messages are significantly larger than the [NoAck] ones, because they additionally contain the [Request] corresponding to the [Brief] that was supposedly sent. The reason to return the [Request] is

that even in case s sent a given brief to r, we want to make sure s did not intentionally only send a subset of the events requested by r – instead of sending only a limited number of briefs, a selfish node could send all briefs, but containing only a limited amount of the events they should really contain.

[GMYH]	[HIMH]	[AckRqst]	[Ack]	[NoAck]
230	12039	690	1166	767

Table 1: Average size of the different LIFT messages (bytes).

5.4 Structural Properties

Accuracy of the capability aggregation protocol. Although we used a static capability distribution to evaluate HEAP, we provide below the accuracy of the capacity aggregation protocol that we run separately. The capability aggregation protocol provides each node with the ability to estimate the gloabl capability of the system. Figure 6 depicts the output of the capability aggregation protocol on one of the 236 PlanetLab nodes. Capability data is gossiped every second. The capability estimate and the real capability wealth are depicted respectively by the plain and dashed lines. Error bars are also plotted and represent the range of errors among node estimations. The absolute error is represented by the light plain line and is computed as the difference between the real capability and the average of the estimated values. The maximum and average error are respectively 14.39% and 3.27% (Figure 6). We thus observe that the estimation on each node is quite accurate.

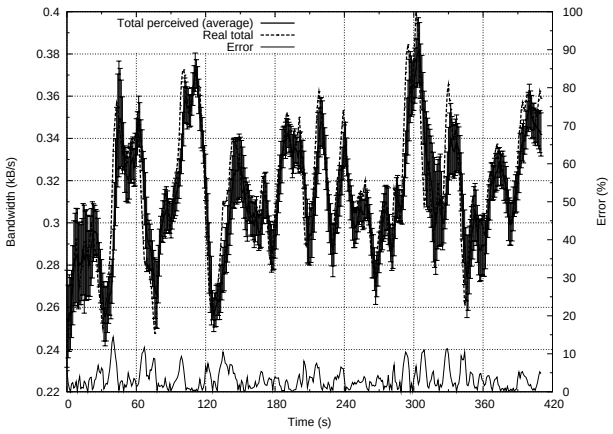

Figure 6: PlanetLab: Accuracy of the capability aggregation protocol

Accuracy of the fanout adaptation. Likewise, we run an experiment with capability changing over time. Figure 7 shows that the average fanout value over time is maintained. Error bars are represented on the graph and show that the average value remains within reasonable bounds compared to the mean fanout set to 4 in this experiment. The figure shows that the maximum fanout remains within the bounds set by the compensation protocol (set to 10 in this experiment). We observe that the fanout adaptation protocol suceeds to keep the average fanout value close to the initial fanout.

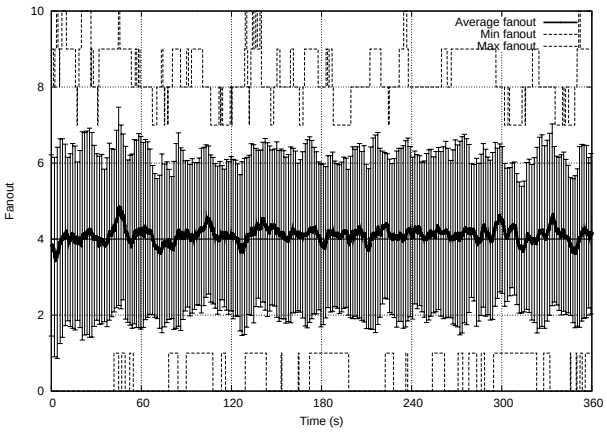

Figure 7: PlanetLab: Evolution of the average fanout value

6 Related work

This section discusses related work both in the area of gossip-based protocols and peer to peer streaming systems. We focus more specifically on approaches that account for heterogeneity.

Gossip-based protocols. Gossip based protocols are scalable and they exhibit high resilience to failures and good probabilistic reliability guarantees [14]. They provide a general-purpose appealing solution for a wide range of distributed applications [6] such as replicated databases [11], aggregation (cf. [22, 17, 28]), failure detection [44, 45], mobile event notification [18, 34], and overlay construction [21, 38, 8]. In addition, specific gossip protocols were devised to support random peer selection [4, 12, 23, 15, 48], or to cope with severe types of failures and attacks [20, 33, 35, 37, 9].

As we pointed out in the introduction, these protocols involve all nodes evenly in the dissemination task [36], which is often undesirable in highly heterogeneous systems such as the Internet. Our objective is thus to tackle this limitation and have nodes contribute to the system according to their capabilities.

Some amount of heterogeneity is present in gossip protocols that do not assume a prior knowledge of the size of the system, e.g., [15], and where nodes may start with different fanouts. However, these protocols consider such heterogeneity as a temporary situation and typically seek to converge towards a stable state where all nodes have the same fanout of $\ln(n)$. In contrast, HEAP seeks to assign different fanouts to different nodes in order to achieve better resource utilization.

Certain gossip-based dissemination protocols do have adaptive features. Some propose ideas that resemble our fanout adaptation scheme, while others exploit techniques similar to our biased gossip-target selection scheme. However, to the best of our knowledge, none of them combines both.

The gossip protocol proposed in [40] adapts the emission rate for a known buffer size depending on the number of participating nodes. In Smart Gossip [31], nodes of a wireless network gossip with different probabilities to account for the heterogeneity of the topology. The goal, however, is to reduce the overall number of messages sent, rather than adapt to individual capabilities. The protocol of [16] aims at probabilistically maximizing reliability in a spanning tree depending on the characteristics of the network. This approach is related to ours as nodes exchange messages to gather a knowledge of the topology. The aim of providing an optimized tree is fairly different though. In Gravitational Gossip [25], the fanin of nodes may be adjusted

based on the quality of reception they expect. This is achieved by biasing the node selection such that certain nodes have better chances to be selected for gossip than others. While somehow similar in spirit to our gossip-target selection, the technique is static and focuses on the fanin. SwapLinks [47] builds a peer-to-peer unstructured overlay network by statistically controlling the degree of each node depending on its capacity. This approach is related to our fanout adaptation mechanism as nodes with higher degree are selected more often and contribute more to the algorithm. Yet, non uniform peer-selection may hurt the reliability of gossip-based dissemination. In addition, this approach does not rely on the aggregated capacity of nodes across the system.

Heterogeneous-aware p2p streaming. Following the wide penetration of Internet access into homes, peer-to-peer streaming systems have received an increasing interest as ways to leverage the computing power available at the edge of the network. While single-tree approaches have dominated at first, they turn out to be highly vulnerable to failures as a failed node harms its entire subtree. Multi-tree schemes have been proposed such as Splitstream [10], Chunkyspread [46] and Coolstreaming [32, 49]. They enhance reliability by ensuring that part of the stream is disseminated through diverse paths, a characteristic that comes for free in gossip-based protocols where the neighbors of each node keep changing over time [24].

Splitstream accounts for heterogeneous capacities by enabling nodes to limit the number of their descendants in a tree. Yet, heterogeneity is not taken into account dynamically and each node takes local decisions without any normalization over the distribution of capacities across nodes. The work in [7] and [43] proposes 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 and ultimately in the system's performance, particularly in that experienced by high contributors. In addition the protocols aggregates global information about the implication of nodes across trees by exchanging messages along tree branches in a way that relates to our capacity aggregation protocol.

Chunkyspread [46] is a multi-tree Splitstream-like system accounting for heterogeneity using the SwapLinks protocol [47]. Nodes contribute in proportion to their capacity and/or willingness to collaborate. This is reflected by heterogeneous degrees across nodes. Yet there is no specific scheme to discover and aggregate information about node capacities which is simply assumed to be known by the application. A follow-up protocol [42] uses a gossip-based aggregation mechanism to compute some global knowledge of the system through gossiping in order to set the parameters of the Swaplinks protocol. This again plays a similar role as our capacity aggregation protocol. While improving upon single tree approaches, the fairly static nature and the lack of redundancy of these tree-based approaches makes them still vulnerable to churn.

As opposed to tree-based systems, Coolstreaming [49] is data-driven. Content location information is gossiped between neighbors (based on a membership protocol providing each node with a random sample of the network), while the actual content is pulled as in swarming systems. However, dissemination paths in Coolstreaming are static: nodes explicitly choose their partners and do not change them until they have to be replaced, while our approach exhibits a more dynamic nature. The analysis of real Coolstreaming traces [32] has revealed huge heterogeneity in terms of contribution as a result of its partner selection mechanism. Yet nodes have no information available about the load distribution across nodes. The similarity with HEAP is thus limited to the selection of targets based on capability.

7 Concluding Remarks

We propose HEAP, a new gossip-based protocol for collaborative high-bandwidth content distribution. HEAP adapts the dissemination load of the nodes to account for their heterogeneity. Experimental results on a video streaming application demonstrate the significant improvement of HEAP over a standard homogeneous gossip protocol.

We considered upload capacity as the main factor for heterogeneity, which is crucial in the context of streaming. Other factors could reveal important in other applications. We believe HEAP could easily be adapted to such factors encapsulated within the underlying aggregation protocol. Also, we considered the choice of the fanout and the gossip targets as the way to adjust the load of the nodes. One might also explore adapting the frequency of the dissemination or the memory of the nodes devoted to the dissemination.

Acknowledgements

The authors would like to thank Ken Birman, Pascal Felber, Dahlia Malkhi and Jean-Philippe Martin for useful comments, Fabian Kaelin and Song Pak, undergraduates students at EPFL for early work on the gossip simulations and on the PlanetLab implementation. Special thanks to Boris Koldehofe for many fruitful discussions about this project. Maxime Monod has been partially funded by the Swiss National Science Foundation with grant 20021-113825.

References

[1] `http://www.speedtest.net`.

[2] `http://www.internetworldstats.com`.

[3] *Gossip-based computer networking*, volume 41. ACM, New York, NY, USA, October 2007.

[4] A. Allavena, A. Demers, and J. E. Hopcroft. Correctness of a gossip based membership protocol. In *Proceedings of the 24th ACM symposium on Principles of distributed computing (PODC)*, pages 292–301. ACM Press, 2005.

[5] N. T. J. Bailey. *The Mathematical Theory of Infectious Diseases and its Applications*. Griffin, 2nd edition, 1975.

[6] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, May 1999.

[7] M. Bishop, S. Rao, and K. Sripanidulchai. Considering priority in overlay multicast protocols under heterogeneous environments. In *Infocom'06*, Barcelona, Spain, April 2006.

[8] F. Bonnet, A.-M. Kermarrec, and M. Raynal. Small world networks: From theoretical bounds to practical systems. In *OPODIS, 11th International conference on principles of distributed systems*, Guadeloupe, France, December 2007.

[9] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer. Brahms: Byzantine resilient random membership sampling. In *27th ACM Symp. on Principles of Distributed Computing (PODC'08)*, 2008. To appear.

[10] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: high-bandwidth multicast in cooperative environments. In *Proceedings of the 19th ACM symposium on Operating systems principles (SOSP '03)*, pages 298–313. ACM Press, 2003.

[11] 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 *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–12. ACM Press, 1987.

[12] P. T. Eugster, R. Guerraoui, S. B. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(4):341 – 374, 2003.

[13] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. Epidemic information dissemination in distributed systems. *IEEE Computer*, 37(5):60–67, 2004.

[14] P. T. Eugster, R. Guerraoui, and P. Kouznetsov. Δ-reliable broadcast: A probabilistic measure of broadcast reliability. In *24th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 636–643. IEEE Computer Society, 2004.

[15] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans. Comput.*, 52(2):139–149, 2003.

[16] B. Garbinato, F. Pedone, and R. Schmidt. An adaptive algorithm for efficient message diffusion in unreliable environments. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN)*, page 507. IEEE Computer Society, 2004.

[17] I. Gupta, R. van Renesse, and K. P. Birman. Scalable fault-tolerant aggregation in large process groups. In *Proceedings of the 2nd International Conference on Dependable Systems and Networks (DSN)*, pages 433–442, July 2001.

[18] Z. J. Haas, J. Y. Halpern, and L. Li. Gossip-based ad hoc routing. *IEEE/ACM Transactions on Networking*, 14(3):479–491, 2006.

[19] A. Haeberlen, P. Kuznetsov, and P. Druschel. PeerReview: Practical accountability for distributed systems. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP'07)*, Oct 2007.

[20] M. Haridasan and R. van Renesse. Defense against intrusion in a live streaming multicast system. In *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 185–192. IEEE Computer Society, 2006.

[21] M. Jelasity and Ö. Babaoglu. T-man: Gossip-based overlay topology management. In *Proceedings of the Third International Workshop on Engineering Self-Organising Systems (ESOA), Revised Selected Papers*, pages 1–15. Springer, 2006.

[22] M. Jelasity, A. Montresor, and Ö. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, 2005.

[23] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems*, 25(3), August 2007.

[24] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3):8, 2007.

[25] K. Jenkins, K. M. Hopkinson, and K. Birman. A gossip protocol for subgroup multicast. In *ICDCS Workshops*, pages 25–30, 2001.

[26] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 565–574. IEEE Computer Society Press, Nov. 2000.

[27] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumour spreading. In *IEEE Proc. 41st Ann. Symp. Foundations of Computer Science (FOCS)*, 2000.

[28] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS '03)*, pages 482–491, Washington, DC, USA, 2003.

[29] A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):248–258, March 2003.

[30] B. Koldehofe. Simple gossiping with balls and bins. *Studia Informatica Universalis*, 3(1):43–60, 2004.

[31] P. Kyasanur, R. R. Choudhury, and I. Gupta. Smart gossip: An adaptive gossip-based broadcasting service for sensor networks. In *Proceedings of Mobile Adhoc and Sensor Systems (MASS)*, pages 91–100. IEEE Computer Society, 2006.

[32] 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 *IEEE Infocom'08*, Phoenix, AZ, USA, April 15-17 2008.

[33] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR gossip. In *Proceedings of the 2006 USENIX conference on Operating Systems Design and Implementation (OSDI)*, Nov. 2006.

[34] J. Luo, P. T. Eugster, and J.-P. Hubaux. Route driven gossip: Probabilistic reliable multicast in ad hoc networks. In *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM*, 2003.

[35] D. Malkhi, E. Pavlov, and Y. Sella. Optimal unconditional information diffusion. In *Proceedings of the 15th International Conference on Distributed Computing DISC '01*, pages 63–77. Springer, 2001.

[36] R. Melamed and I. Keidar. Araneola: A scalable reliable multicast system for dynamic environments. In *3rd IEEE Int'l Symp. on Network Computing and Applications (IEEE NCA'04)*, 2004.

[37] Y. M. Minsky and F. B. Schneider. Tolerating malicious gossip. *Distributed Computing*, 16(1):49–68, 2003.

[38] A. Montresor, M. Jelasity, and Ö. Babaoglu. Chord on demand. In *In Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005)*, Konstanz, Germany, August 2005.

[39] B. Pittel. On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, Feb. 1987.

[40] L. Rodrigues, S. Handurukande, J. Pereira, R. Guerraoui, and A. Kermarrec. Adaptive gossip-based broadcast. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, San Francisco, CA, June 2003.

[41] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using planetlab for network research: myths, realities, and best practices. *SIGOPS Oper. Syst. Rev.*, 40(1):17–24, 2006.

[42] T. Steele, V. Vishnumurthy, and P. Francis. A parameter-free load balancing mechanism for p2p networks. In *The 7th International Workshop on Peer to peer Systems*, February, Tampa Bay, Florida 2008.

[43] Y.-W. Sung, M. Bishop, and S. Rao. Enabling contribution awareness in an overlay broadcasting system. *SIGCOMM Comput. Commun. Rev.*, 36(4):411–422, 2006.

[44] R. van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, 2003.

[45] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proceedings of the first IFIP International Conference on Distributed Systems Platform and Open Distributed Processing (Middleware)*, pages 55–70, 1998.

[46] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous unstructured tree-based peer to peer multicast. In *Fourteenth IEEE International Conference on Network Protocols*, November 2006.

[47] V. Vishnumurthy and P. Francis. On heterogeneous overlay construction and random node selection in unstructured p2p networks. In *Infocom'06*, Barcelona, Spain, April 2006.

[48] S. Voulgaris, D. Gavidial, and M. van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.

[49] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. Coolstreaming/donet: A data-driven overlay network for efficient live media streaming. In *IEEE Infocom'05*, Miami, FL, USA,, March 2005.