LAYSTREAM: composing standard gossip protocols for live video streaming

Miguel Matos*, Valerio Schiavoni†, Etienne Riviere†, Pascal Felber†, Rui Oliveira*
*HASLab - High-Assurance Software Lab, INESC TEC & U. Minho, Portugal. Email: {miguelmatos,rco}@di.uminho.pt
†University of Neuchâtel, Switzerland. Email: first.last@unine.ch

***Abstract*—Gossip-based live streaming is a popular topic, as attested by the vast literature on the subject. Despite the particular merits of each proposal, all need to implement and deal with common challenges such as membership management, topology construction and video packets dissemination. Well-principled gossip-based protocols have been proposed in the literature for each of these aspects. Our goal is to assess the feasibility of building a live streaming system, LAYSTREAM, as a composition of these existing protocols, to deploy the resulting system on real testbeds, and report on lessons learned in the process. Unlike previous evaluations conducted by simulations and considering each protocol independently, we use real deployments. We evaluate protocols both independently and as a layered composition, and unearth specific problems and challenges associated with deployment and composition. We discuss and present solutions for these, such as a novel topology construction mechanism able to cope with the specificities of a large-scale and delay-sensitive environment, but also with requirements from the upper layer. Our implementation and data are openly available to support experimental reproducibility.**

I. INTRODUCTION

Originally introduced for propagating updates in distributed databases [11], epidemic or gossip protocols have attracted increasing interest as the scale and dynamism of computer systems have drastically increased over the last decade. A wide variety of gossip protocols have been proposed, for purposes such as fault detection [28], membership management [31], aggregation [15], data dissemination [6], [12], [25], publish-subscribe [2], [33] and infrastructure management [27]. This interest stems from the robustness and scalability of the gossip model, but also from its simplicity. A node participating in a gossip protocol follows a periodic interaction pattern. It regularly selects a communication partner and exchanges some data with it. This allows each of the two nodes to update a local state, according to protocol-specific rules. Due to the randomized, local and redundant nature of interactions, gossip protocols are known to be very robust against faults and churn. They are therefore well suited to dynamic environments. Furthermore, each node only needs limited knowledge of the entire system. Gossip protocols thus generally scale well to very large numbers of nodes.

While operations at the level of a single node are generally simple to describe and implement, the evaluation or modeling of the behavior of a collection of nodes on a global scale is a challenging task, due to the scale but also to the emergent and probabilistic nature of gossip protocols. Furthermore, while one can compose gossip protocols to build richer distributed services [2], [29], the resulting stack is not straightforward to study and reason about. There is not only a limited number of examples of evaluation of individual gossip protocols deployed on real platforms but also their interplay in these real environments remains largely unstudied. The potential mismatch between the behavior predicted by theory or simulations and practical results has been pointed out before, further highlighting the challenges of deploying key protocols such as consensus in the real world [8], [24].

In this paper, we wish to make a step towards bridging the gap between theory and simulations on the one hand, and practice and real deployments on the other, by building, deploying and evaluating a demanding application, live video streaming, using gossip. Rather than contributing yet another protocol, and comparing with existing state-of-the-art live video-streaming systems, which would require to implement all features of membership management, topology construction and data dissemination, we instead wish to investigate if this application can be built based on a composition of existing protocols, each forming a gossip building block. We wish to evaluate their behaviors under real deployments, and well as evaluating the behavior of their composition. We implemented an instantiation of the peer sampling service for membership management [17], T-Man [16] for topology construction and Brisa [25] for data dissemination. Our results are based on the deployment of a prototype, named LAYSTREAM. It uses the VLC media player as the source and sink of video streams. Our results generally support the main conclusions from the literature but, unsurprisingly, unearthed issues and limitations stemming from the simplifying nature of simulations or from the lack of consideration for protocols interoperation. We discuss and present solutions for these issues.

Our contributions are the following: i) we study the challenges faced when combining and deploying several gossip protocols on a real environment, ii) we propose solutions and protocol adaptations for the problems encountered and iii) we propose a novel topology construction mechanism suitable to a large-scale deployment of a delay-sensitive application. Our implementation and datasets are openly available for experimental reproducibility.[1]

The rest of this paper is organized as follows. Section II describes the protocol stack and the requirements of each layer. Section III presents the implementation of the stack and discusses the adaption made to existing protocols. Section IV presents our deployment and evaluation. Section V reviews related work and Section VI concludes.

[1] http://www.splay-project.org/laystream.

978-1-4799-6201-3/14/$31.00

II. Live Streaming Components and Requirements

We start by describing the stack of services required to support live video streaming. We use a bottom-up approach and focus on the guarantees offered by each of the layers, illustrated by Figure 1. The base layer offers point-to-point wired communication. We assume that any two nodes can directly communicate with one another. While this does not hold in general due to NATs and firewalls, there are techniques at the peer sampling layer for addressing this restriction [19].

The first layer provides a *peer sampling* service (PSS) [17] which guarantees *membership management* and *randomization*. It provides each node with a small and fixed size set of constantly changing contacts to other nodes, the *pss-view*. This pss-view should be as close as possible to a *random* sample of all nodes currently present in the system, thus effectively constructing a random graph. The expected properties are low clustering, low diameter, and balanced in-degrees [17]. The *quality* of that random graph is fundamental not only for performance reasons but, in some cases, also for correctness [6], [11], [15], [28], [33]. Namely, high clustering negatively impacts robustness as the graph will tend to be disconnected more easily, diameter impacts end-to-end latency and as such it should be as low as possible, and balanced degrees help evenly spread the management and dissemination effort. For instance, the topology construction service discussed next *requires* randomness and constantly changing views to ensure convergence and self-organization capabilities [34]. The PSS also guarantees connectivity and robustness, as failures are unlikely to result in all members in the pss-view failing simultaneously. Besides, the probability of a partition under the failure of a large fraction of the network is minimal. Finally, the PSS deals with membership management. It must efficiently accommodate new nodes joining the system and place them in the views of other nodes, as well as promptly remove departed nodes. Communication at the level of the PSS is sporadic and short lived. It thus uses UDP and tolerates the associated message loss.

The second layer provides a *topology construction* (TC) service [16], [34]. Its guarantees are dictated by the *dissemination* service requirements [25]. First, it needs to maintain a stable and bi-directional graph between nodes. Each node has a *tc-view*, maintained separately from the pss-view but containing nodes initially obtained from it. The TC service maintains persistent bidirectional TCP connections towards all nodes in this tc-view. It also acts as a failure detector, using periodic heartbeats to detect unresponsive nodes and replacing them by live ones obtained from the pss-view. Second, the graph formed by the persistent connections must remain connected. Note that the underlying directed PSS graph is connected with high probability thanks to its random graph nature [17]. One possibility to ensure connectivity of the dissemination graphs would be to leverage randomness in the same way, and randomly pick a subset of the nodes provided by the pss-view [20]. This is however inappropriate for live video streaming which is bandwidth-intensive and requires low transmission delays. Indeed, as packets will be transmitted over multiple links, using arbitrary links would create unnecessarily long delays and put pressure on the underlying network. Therefore, the third requirement for the TC layer is to meet the two previous requirements while linking nodes that are *close* according to a proximity metric such as the communication delay or the geographical locality.

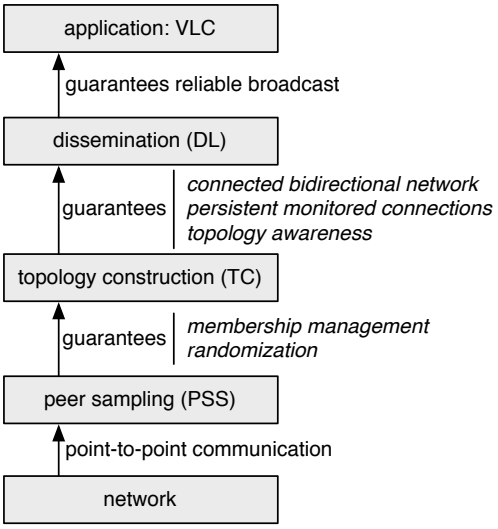

Fig. 1. Stack of services and guarantees.

The third layer provides a *dissemination* service [25]. Its design is straightforward thanks to the guarantees provided by the lower layers. A simple flooding strategy is guaranteed to reach all nodes, because of the connected and bidirectional nature of the underlying graph. Flooding is robust but particularly inefficient as it yields many duplicates. The role of the dissemination layer is thus to construct efficient dissemination structures (namely trees) and maintain them in face of faults reported by the TC layer.

On top of the stack lies the application itself, in our case the VLC player. It receives video packets from the dissemination layer as if they were originating directly from the source via a point-to-point UDP connection. We maintain a buffer of received packets and start the VLC client only when a configurable number of packets is present in the buffer, to accommodate fluctuations in reception delays due to churn and ensure a smooth video display.

III. Gossip-based Building Blocks

In this section we describe the instantiation of the layers that compose LayStream using standard gossip-based protocols proposed in the literature, discuss the associated design decisions and propose solutions for the issues uncovered.

A. Peer Sampling Service

The PSS layer is implemented using the framework described in [17], which allows to instantiate PSSs such as Cyclon [31] or Newscast [32]. Each node maintains as its pss-view a list of c descriptors to other nodes. A descriptor contains a `ip:port` pair and an `age` field indicating its freshness. Each node p updates its pss-view by means of *view exchanges*, initiated by an active task on one node and served by a passive task on another. The implementation of the PSS faces a trade-off between the objectives of randomness quality and membership management. Randomness quality is measured by the *in-degree distribution* and the *clustering factor*. The in-degree denotes the

number of occurrences of a node in the views of the other nodes. A balanced distribution of in-degrees yields good load balancing. The clustering factor indicates that the nodes in a view are also neighbors themselves. A random graph exhibits low clustering. A high value would make the graph vulnerable to massive failures and churn, and impair convergence for protocols using the PSS. On the side of membership management, the goal is to ensure that failed nodes get removed from other views as fast as possible (in terms of number of exchanges, as indicated by the age fields of the corresponding descriptors). This tradeoff is controlled by the *healing* and *swapping* parameters H and S, subject to $H + S \leq \frac{c}{2}$ [17]. A value of $H = \frac{c}{2}$ (as in Cyclon [31]) favors randomness quality, at the price of slowly discarding failed nodes, while a value of $S = \frac{c}{2}$ (as in Newscast [32]) favors the quick removal of failed nodes but leads to higher clustering and unbalanced in-degrees.

The active task, called periodically and at the same frequency on each node, first selects a target node q and constructs a list with $\frac{c}{2} - 1$ descriptors randomly selected among the $c - H$ newest entries, as well as the identifier of p with age 0. The list is then sent to q, which replies back with a sample of its own descriptors chosen in the same way. Node p integrates the received descriptors in its own view. To keep the view size constant to c elements, p first drops the H oldest elements and, as needed, also removes the first S descriptors sent to q, and then random descriptors. Node q proceeds similarly. Finally, p increments the `age` of all descriptors in its view.

1) Implementation Issues: When implementing the PSS layer, we faced an issue that does not arise in simulations. The passive task at node p can reply to requests at virtually any time. If an ongoing exchange has been initiated by p and is being served by another node q, the modification of the view performed by p's passive task will break exchange semantics resulting in the duplication or loss of a descriptor. This can also lead to artificial clustering. Such situations should be avoided: while concurrent modifications seldom happen, they introduce shifts that persist throughout the lifetime of the system and worsen over time. For instance, a descriptor that gets duplicated results in an increased in-degree for the linked node, which leads in a higher chance of being contacted by others, and in turn in a higher chance of being subject to that shift again. A simple solution is to make the exchange atomic by locking the access to the view while the active task is pending. This requires careful design in the presence of churn and the use of appropriate timeouts. An alternative solution [30] is to simply reject incoming requests at the passive task when an exchange is pending. Because the latter option results in wasted communication steps and thus slower convergence, we opted for the first solution.

B. Topology Construction

The TC (topology construction) layer is implemented using a modified version of the T-Man protocol [16]. The goal of this layer is to create an overlay that matches the structural needs of the dissemination layer: links must be bidirectional and the network must be connected. Additionally, nodes should be linked to close nodes in terms of delays and geographical proximity, for reasons of performance and network utilization.

The basic operation of T-Man is similar to that of the PSS. However, the selection of the descriptors that are kept in each node's tc-view after the exchange is not random but depends on semantic information included in the descriptor. In our case, this semantics is the 2-dimensional geographical coordinate (latitude and longitude) of the node obtained from a *GeoIP* service (see Figure 2). The selection of which descriptors to keep is based on a *distance* function.

Fig. 2. Geographical distribution of nodes for a sample run.

Since there is no global knowledge, the topology self-organizes gradually as follows. Periodically, the active task at node p selects another node q from its tc-view, and sends it its own tc-view (or the best subset according to q). The passive task at node q also returns q's current tc-view. Both p and q merge their tc-views with the set of received descriptors, sort it according to the distance function, and keep the c closest entries. Furthermore, periodically (and on bootstrap), an exchange is initiated with a random node obtained from the pss-view. This guarantees that the graph will converge, but it may take long when using a carelessly designed distance function (e.g., if each node needs to encounter each other node in its pss-view). For fast convergence the distance function should be transitive: if node p is close to node q, and node q is close to node r, then p is also close to r. Our distance function is based on Euclidian distance between geographical coordinates and is therefore transitive. One trivial modification we also add to the original T-Man is to make all links bidirectional.

1) Implementation Issues: We uncovered two implementation issues in this layer regarding resource utilization and the properties of the resulting topology.

The T-Man protocol was evaluated in [16] by simulations only (note that this is also the case of the similar Vicinity [34] protocol). When deploying the protocol on a real testbed, we faced the problem of network resources limitations. The tc-view is highly dynamic, in particular after a node joins the system and until it evolves towards its stable configuration. Immediately establishing TCP connections when nodes are added to the tc-view is problematic both for resource utilization and load balancing, in particular because many of these connections will be short lived and discarded at the next gossip exchange. Still, TCP connections are desirable at the DL layer, as pointed out in Section III-C. However, TCP consumes more resources than UDP and requires maintaining connections on both ends of the links (e.g., by allocating few file descriptors). Some nodes might also be at the boundaries of clusters and pass through the views of many other nodes. Therefore, establishing temporary bidirectional communications with all these passing-by nodes

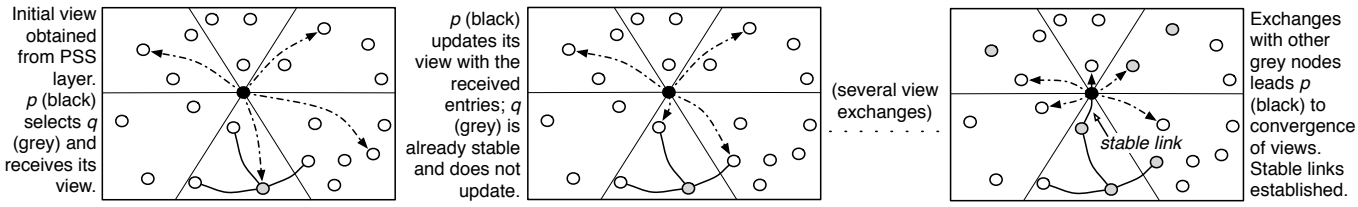

Fig. 3. Evolution of the view of node p towards Yao links for $k = 6$.

is a clear waste of resources. In extreme cases, this even leads to resource exhaustion at these boundary nodes.

We solve this issue by using UDP for all gossip exchanges. We only establish TCP connections when the corresponding entry in the tc-view has stabilized, which, based on experimental observations, we assume to happen after 5 consecutive gossip exchanges without modification. Because of this, only stabilized links are exposed to the dissemination layer. These stable links are then monitored for failures using heartbeats.

Our initial attempt to topology construction used the T-Man protocol applied to network coordinates from Vivaldi [10]. Vivaldi maps the physical location of nodes into a synthetic coordinate space. Each node is associated to a point in that space and distances between points reflect the latency between nodes. The coordinates of the nodes are updated based on delays observed on application-level messages. Vivaldi has a completely decentralized operation that is well adapted to our context. Each node simply chooses the c closest nodes according to the Euclidean distance between their respective coordinates.

However, this approach results in a disconnected topology. This is because nodes are mostly clustered in one large group in the US and another in Europe (Figure 2). As a result, nodes select as neighbors other nodes in the same cluster resulting in a partition. A trivial attempt to solve this problem was to add a set of additional randomly selected entries in the view, but it proved unsatisfactory. Indeed, using random links introduces delay penalties that are propagated to all nodes, and keeping the network connected, in particular under churn, requires a large number of such links, clearly diminishing the interest of using a delay-based node selection.

2) Spanning Graph Topology: Achieving the apparently conflicting goals of connectivity and low distance between nodes requires a careful adaptation of the T-Man protocol in order to guarantee system-wide structural properties that are not achieved by simply selecting the c closest nodes. Connectivity in a network with bidirectional links is equivalent to the ability of reaching all nodes from any node of the network, e.g., by building a *spanning graph*. A spanning graph contains for any source node an embedded *minimum spanning tree* made of all the vertices (nodes) and a subset of edges that guarantee connectivity but minimizes the sum of edges' costs. In our case, the cost of an edge is the geographical distance between two nodes and the spanning graph is a subgraph of the graph exposed by the PSS.

Several approaches have been proposed for constructing spanning graphs, some of which can be implemented using gossip protocols. Typical examples are Delaunay triangulations [5], but their main drawback is that the in-degree of nodes remains unbounded. Since we require bidirectional links, a node may end up maintaining a very large number of TCP connections. Instead, we took inspiration from ad-hoc wireless networks, where spanning graphs are necessary to establish routing protocols, and distances directly affect power requirements and energy consumption—two elements that must clearly be minimized. Wang and Li introduced Yao-Yao graphs in this context [36]. Yao graphs [37] are defined on a 2-dimensional Euclidean space (the space of geographical coordinates in our case). Each node n is associated with k equally-separated rays originating at n, that define k *sectors*. In each sector, n selects the closest node and connects to it with a directed edge. It has been shown that the Yao graph is a spanner for $k \geq 4$ [3]. The Yao-Yao graph builds on top of the Yao graph. For each sector, we discard all incoming links but the shortest one, and make this link bidirectional. The degree is thus bounded by k both for incoming and outgoing links, which will ensure good balancing of the load. It has been shown that the Yao-Yao graph is also a spanner graph [18].

To construct the Yao-Yao graph, T-Man must be modified to consider a separate entry in the view for each of the k sectors. All entries are bootstrapped using descriptors from the PSS if they are located in the corresponding sector. Descriptors received by gossip exchanges are considered only for the sector they lie in. Figure 3 presents an example of the evolution of the view of a node p based on gossip exchanges with its neighbors. While the illustration uses $k = 6$ for simplicity, we use $k = 8$ in our experiments.

C. Dissemination Layer

The overlay created by the TC layer is a spanning graph with bidirectional links. It follows that a flooding operation (where each node forwards the first occurrence of an incoming message to all its neighbors) is guaranteed to reach all nodes. Albeit extremely robust, such a mechanism is highly inefficient and not adapted to bandwidth-intensive video streaming. Each node receives each message from multiple nodes, up to c in the worst case.

The goal of the dissemination layer (DL) is to provide an efficient dissemination service. This service constructs efficient dissemination trees embedded in the spanning graph provided by the TC layer and inheriting from its robustness. The links that are not currently active as part of a dissemination tree can be rapidly used as backup links upon failures—as these are maintained and monitored as persistent TCP connections. We use a subset of the Brisa gossip protocol [25] for the implementation of this service.

Initially, all links exposed by the TC service are *active*. Upon receiving a message from a neighbor, node p propagates

it to all active links. The dissemination of the first message thus corresponds to flooding. Thereafter, a deactivation mechanism allows selecting a single parent for each node, effectively forming a tree: if a node p receives the first copy of a message from q, it simply deactivates all links but the one from q. Note that deactivating the link is only performed at the level of the dissemination layer: this link is still maintained as a persistent TCP connection at the TC layer. The failure of the parent node will be detected by the TC layer and trigger an up-call to the dissemination layer. The node then simply re-activates all its links and selects as a parent the first node that sends the next new message. The DL thus builds a tree by using the TC links without relying on complex cycle detection mechanisms but just on the detection of duplicate message receptions (a simplification of Brisa's *first-come first-picked* strategy [25]).

The use of a single tree is efficient in terms of duplicates but results in high load differences between the nodes. Some nodes may contribute a high upload bandwidth while others are leaves and do not contribute. We address this issue by constructing several trees concurrently using the same activation/deactivation mechanism. The only change needed is for control messages to carry an identifier that specifies the tree to which they belong. The source then splits the video packets onto the available trees, allowing for parallelization of the data dissemination and a more balanced distribution of load, similarly to SplitStream [7].

1) Implementation Issues: The Brisa [25] tree construction mechanism is tightly tied to a particular PSS protocol [20] unaware of the underlying topology and thus unable to exploit it. As a consequence, the version of Brisa in [25] proposes parent selection strategies aiming at reducing delays or network costs, but which may result in cycles that have to be prevented using a cycle detection mechanism. By decoupling the tree construction mechanism from the rest of the protocol and putting it atop the topology-aware connected overlay provided by the TC, the tree construction becomes simpler and more flexible as the logic of which nodes are best to build the tree according to some criteria is now pushed down to the TC and no complex cycle detection mechanism is required.

IV. Evaluation and Lessons Learned

We start by evaluation each layer of LayStream independently to validate their guarantees. Then, we evaluate the performance of the complete stack under a live video dissemination workload. The evaluation is done using Splay [21] on a cluster of 14 bi-quad-core Xeon machines (112 cores in total, 224 with SMT), each with 8 GB of RAM and interconnected using a switched 1 Gbps network. We deploy up to 300 nodes on this cluster. While we are aware that the experiments would be more interesting if we had thousands of machines instead of a few hundreds, unfortunately there is no such large scale testbed available for research purposes.

A. Peer Sampling Service

We first evaluate the two fundamental guarantees of the PSS layer: randomness quality and membership management. The former corresponds to the clustering degree and in-degree distribution, and to the resulting balance in bandwidth usage. The latter relates to the time taken to remove stale entries from the nodes' view after a failure. Our experiments reproduce some of the results obtained by simulation in [17], [31], [32], [34] but using a real implementation. We instantiate four protocols with a view size of $c = 20$. *Blind*, with $H = 0$ and $S = 0$ performs random selections for view selections at each exchange. *Swapper* corresponds to Cyclon [31] and uses $H = 5$ and $S = 0$. It favors randomness quality and in particular seeks to reduce clustering and load imbalance. *Healer* corresponds to Newscast [32], with $H = 0$ and $S = 5$ and favors membership management over randomness. *Hybrid* combines the two objectives by setting $H = 2$ and $S = 3$.

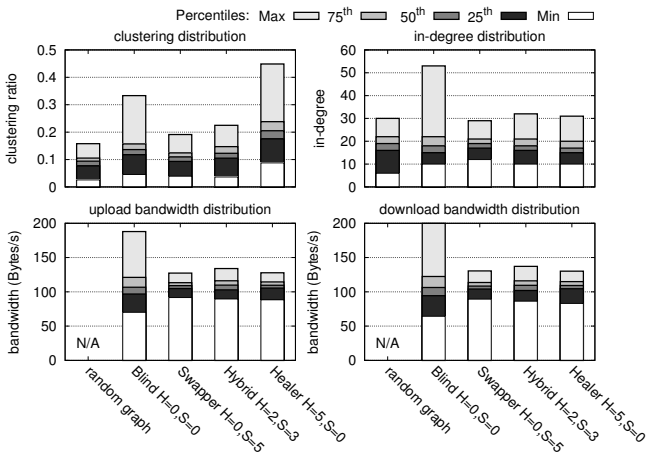

Fig. 4. PSS: clustering and in-degree distributions (top), upload and download bandwidth usage distributions (bottom).

Figure 4 (top) presents the randomness quality of a snapshot of the graph obtained after 5 minutes. The results are presented for each variant as well as for a reference random graph generated offline with the same number of nodes. We present the clustering distribution (left) and in-degree distribution (right). We use a representation based on stacked percentiles throughout this section. The white bar at the bottom represents the minimum value, the pale grey on top the maximal value. Intermediate shades of grey represent the 25^{th}, 50^{th}–the median–, and 75^{th} percentiles. For instance, the median clustering for *Healer* is 0.2 meaning that 50% of the nodes have 20% or less of all of possible pairs of their neighbors that are neighbors themselves.

We observe that, as expected, *Blind* performs badly for both aspects: clustering is high and in-degrees are skewed, with some nodes being in more than 50 views (while the distribution should be as narrow as possible around the average in-degree of 20). *Swapper* gives the best results in terms of clustering and in-degrees, for which it gives similar, respectively better, results than the random graph. While *Healer* performs well in terms of in-degrees, it yields a high clustering, which will result in slower convergence at the TC layer and leads to higher chances of partitioning. We could not reproduce the skewed in-degree distribution for Healer presented in [17]. This is due to the smaller size of our deployments compared to what can be obtained in synchronous simulations. The results indicate that *Swapper* is the best choice in terms of randomness guarantees, while *Hybrid* seems to be a good compromise that approximates the characteristics of a random graph.

Next, we evaluate the overhead imposed by the PSS by

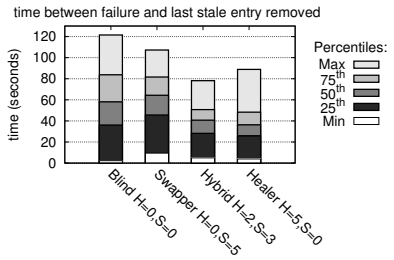

Fig. 5. PSS: time to remove failed nodes after half of the network fails.

observing the upload and download bandwidth. Because the PSS is a low level service it should be frugal in terms of bandwidth. The active task is invoked every 10 seconds. The distribution of upload and download bandwidth is presented in Figure 4 (bottom). All variants exhibit well balanced distributions, with the exception of *Blind*. This is due to the high imbalance in in-degrees distribution, resulting in some nodes being contacted much more or much less than average.

Finally, we study the quality of the membership management by evaluating the time required for discarding failed nodes from the views of all nodes. Until then, the TC layer may try to connect to the failed nodes, resulting in wasted communication. We use a catastrophic scenario where half of the nodes (100 out of 200) fail after a stabilization period of 5 minutes. We measure the views of all nodes, reported after each exchange, and in particular the last time failed nodes appears in a view. Results in Figure 5 indicate that, as expected, *Blind* performs the worst. Indeed, this strategy does not use the `age` field in the descriptors and thus stale descriptors are discarded only due to random selections after a large number of exchanges.

Healer performs the best (in terms of median time), and in particular better than *Swapper* which is on par with the conclusions in [17]. This illustrates the compromise made when setting the H and S parameters, which will have an influence on the service given to upper layers. Again here, *Hybrid* performs well as even a small value of H allows discarding old descriptors quickly enough to grant rapid reaction to membership change [17]. As a result, we select the *Hybrid* strategy for the remaining of our experiments.

B. Topology Construction

We now evaluate the TC layer using the modified version of T-Man [16] described in Section III-B. We evaluate both the Vivaldi synthetic coordinates with some links and the Yao-Yao graph construction based on geographical distances. The view size is $c = 10$ for the former and $c = k = 8$ sectors (view entries) for the latter. The active task period is 10 seconds.

Vivaldi coordinates were generated offline using 5 dimensions (as recommended by [10]) based on the geographical coordinates obtained from PlanetLab. We use an Euclidean distance function for $p \times c$ entries of the view, and random links for the remaining $(1-p) \times c$ entries. Selecting random links should allow overcoming the clustering and produce connected topologies. The pseudo-random distance between nodes p and q is given by `abs(hash(`p`)-hash(`q`))`. The reason for this is to decouple the distance from any geographical information while making it deterministic (as given by the hash

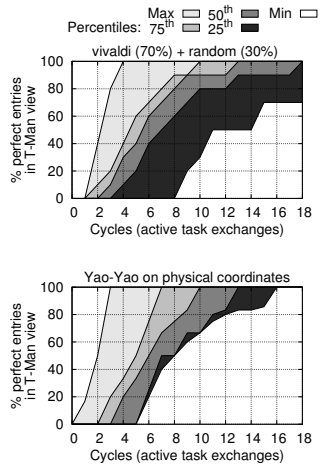

Fig. 6. TC: convergence time (in cycles).

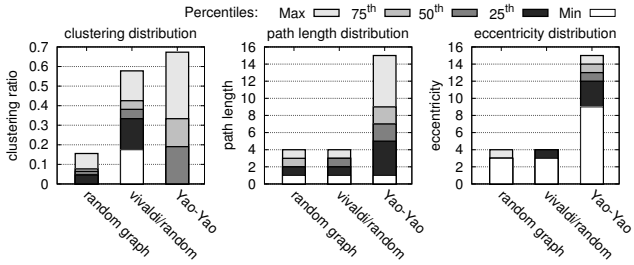

Fig. 7. TC: constructed overlay properties for the two selection strategies and baseline random graph.

function), allowing each node to sort every other node and reach a stable view that can be exposed to the DL. It also allows deterministic bidirectional links, another requirement of the DL. In this geographical distribution, we were only able to obtain connected topologies when at least $1 - p = 30\%$ of the nodes in the view where selected using the pseudo-random criteria. For the Yao-Yao graph, we use the Euclidean geographical distance of Figure 2. Yao-Yao graphs are always connected.

We present the distribution of convergence time, measured in the number of active task cycles, in Figure 6. Convergence measures the time taken for each node to connect to the ideal nodes (pre-computed offline). The reported time corresponds to the first inclusion of these nodes in the tc-views, not their exposition to the DL. Interestingly, the Vivaldi/random distance takes longer to converge than the Yao-Yao graph. The reason for this stems from the need to use the pseudo-random criteria. In fact, the TC works best when the transitivity among neighbors holds which is not the case with a pseudo-random selection. Unfortunately, this is an inherent limitation for metrics based exclusively on the physical distance which can be addressed by planar graphs such as Yao-Yao that take into account the *direction* of neighbors.

Next, we study the graph properties of the topology. Figure 7 shows the clustering, path length and eccentricities distributions. The definition of clustering is the same as before. The path length is the smallest path between any two nodes, in number of vertices. The eccentricity of a vertex (node) v is the maximum distance from v to all other vertices. A small average path length

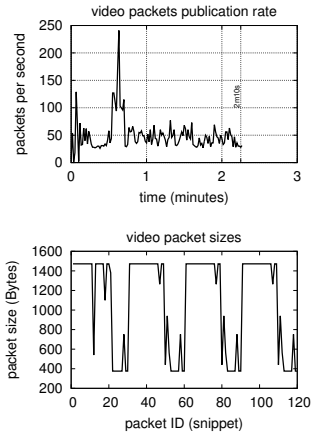

Fig. 8. Characteristics of the variable bitrate video stream.

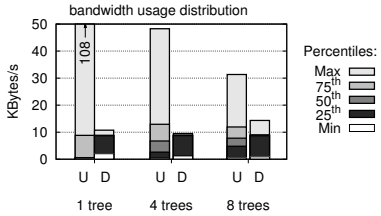

Fig. 9. Influence of the number of trees on load repartition.

might result in a fat dissemination tree at the DL (with a small maximal depth). However, it does not necessarily corresponds to low end-to-end latencies as the length metric is the number of vertices only. A fat tree also results in a imbalance in the use of nodes upload capacities. For instance, Vivaldi/random has small and even path lengths and eccentricity, but at the expense of using random, potentially high-delay, links. These metrics are higher for the Yao-Yao graph because only close links are used. Still, as previously noted, guaranteeing a connected network with Vivaldi/random is very difficult and results in slower convergence times. This implies that a link to a failed node will be replaced slower (in particular for the random portion of the view), putting the system at risk of partition under moderate churn. These results confirm Yao-Yao graphs as the choice for providing a connected, bidirectional and locality-aware abstraction to the DL.

C. Dissemination Layer and Client Application

We evaluate the DL and the client application together. The metrics of interest are: bandwidth distribution, dissemination delays for individual packets, and the fill ratio at the play buffers, which indicates the ability to play the video without degradation. We let the PSS and TC layers stabilize by running the system for 3 minutes before sending video packets from a source running VLC as a streaming server towards clients running VLC as a display client. The first video packet creates the initial dissemination trees (Section III-C). We start our evaluation in a static setting, and then evaluate LAYSTREAM under churn.

The video streamed by the VLC player is a MPEG-encoded

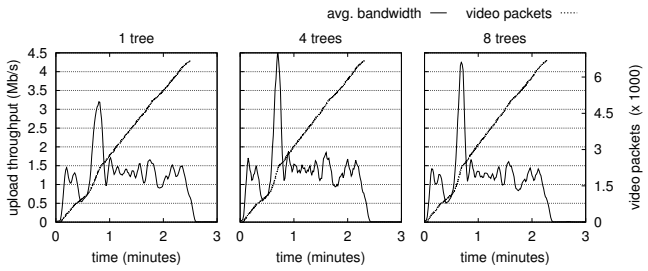

Fig. 10. Evolution of the upload throughput over time (moving average over five reported measurements)

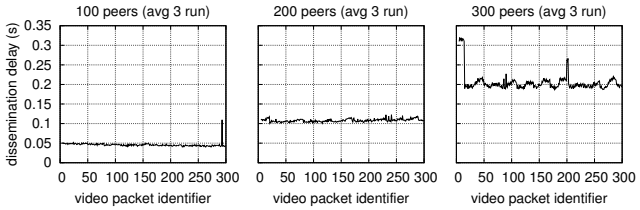

Fig. 11. Average dissemination delay for increasing numbers of nodes.

video with variable bit rate (VBR) with a duration of 130 seconds and size of 7 MB. We chose to use a relatively short video in order to exhibit the behavior of LAYSTREAM upon bootstrap and ending of the stream. VBR allows to use more bits to encode more complex scenes and less bits to encode simpler scenes yielding better overall quality. However, VBR is more challenging to the dissemination layer because it is prone to packet bursts and variable packet sizes [9]. Figure 8 presents the video characteristics. Around 40 seconds after the start of the video, a succession of fast paced scenes with no visual similarities result in a burst of messages, that the dissemination trees must handle.

We first evaluate the bandwidth requirements, and in particular the impact of using multiple independent dissemination trees constructed as the source dispatches in a round-robin fashion the packets to 1, 4 or 8 of its DL neighbors. Figure 9 presents the distribution of the upload (U) and download (D) bandwidth at all the nodes, over the complete video duration while Figure 10 presents the upload throughput (as a 5-second moving average over all per-second reports from the 200 nodes). Clearly, while the bandwidth requirement is the same in all cases, the use of a single tree results in a highly skewed upload utilization. This clearly improves when increasing the number of trees. In fact, for 8 trees, half of the nodes upload roughly the same amount they download and the remaining ones only upload slightly more, with a maximal upload requirement of 31 kB/s. Those are typically nodes closer to the source (in the TC layer) and as such most of their links will be active. Still, the improvement from one tree to eight is remarkable when we also take into account that there are no complex load balancing mechanism in place, or explicit construction of disjoint trees [7].

Results for the dissemination of individual packets are presented in Figure 11. We present the average dissemination delay for all nodes, using a single tree for 100, 200 and 300 nodes. Each scenario is executed three times, and the plots show the average across the different executions. Delays for the three scenarios are consistent and well below reasonable

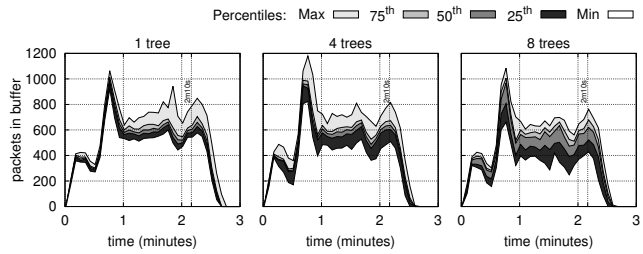

Fig. 12. Distribution of messages in buffers in a static setting.

safety margins to display the video at end nodes, between 0.05 seconds and 0.25 seconds. Note that if the evaluation was conducted in a WAN environment the absolute latency values would substantially higher due to the physical nature of the network. Still, we believe the original shape and evolution would remain similar to what we observed.

Next we study how buffer occupancy evolves over time. During the experiment, each node buffers the packets it receives and when required, sends them to VLC. For each packet, we measure the time when it is sent to the VLC client. We first need to determine the minimal number of packets in the buffer for starting the video display, that guarantees a playback without interruption. Through experimental observation we determined the required buffer size to be of 800 packets in a static environment and 1,600 under churn. The size of the buffer is more influenced by the variation in the publication rate and churn than the actual video length (6,682 packets). We thus expect a video with similar characteristics but longer length to require the same buffer size. The VLC client starts consuming packets from the buffer when it it reaches 80% occupancy. The buffer occupancy is then a good indicator of LAYSTREAM's ability to deliver packets to the video player *on time*. An empty buffer indicates packets are arriving too slowly and thus a more likely interruption in the playback. Figure 12 shows the buffer occupancy for 1, 4 and 8 trees. In all scenarios the buffer never gets empty: LAYSTREAM is able to disseminate the packets to all nodes on time. The larger variance for the 8 trees is due to the increased parallelism that results in more fluctuations on the rate at which packets are received. Still, the buffer occupancy remains sufficient to prevent problems in the video playback. Note that the time required to completely consume the packets in the buffer at the end reflects the lower packet publication rate (50 packets/second) in the last segment of the video (Fig 8, left).

Next we evaluate LAYSTREAM under churn. The average size of the system is 200 nodes, but a moderate (1%) or heavy (2%) fraction of the nodes leave or join the system, on average, every 2 minutes. While for the evaluation of the PSS we were interested in measuring its behavior under a catastrophic failure, here the goal is to study the impact of moderate but continuous churn on the system. Figure 13 (top) presents the evolution of the system size, and the stacked bars represent joins and leaves to/from the system for each minute. To better assess the impact of churn, we stream a video of 12 minutes, which is a concatenation of 4 instances of the video used for the previous experiments. Figure 13 (bottom) presents the buffer occupancy evolution and, as observed, churn has only a moderate impact on buffer occupancy which confirms LAYSTREAM's fast recovery capabilities. Note that the average number of packets in the

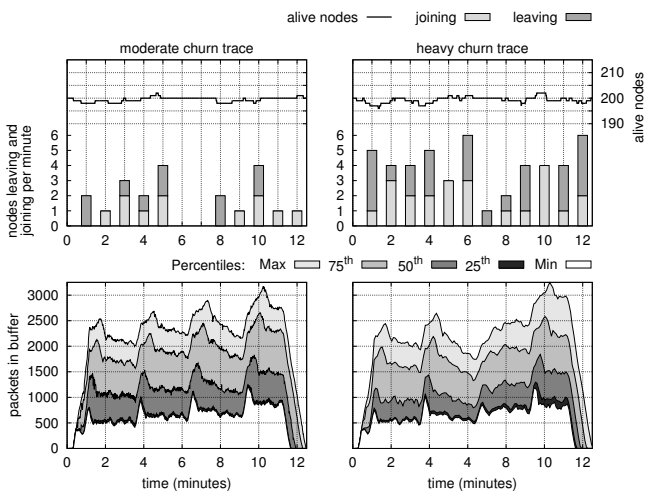

Fig. 13. Distribution of messages in buffers under churn.

buffers under churn is larger because, as stated previously, the buffer size is larger (800 vs 1,600 packets) to properly accommodate the variations imposed by churn. We conclude by analyzing the buffer contiguity in both a static and dynamic setting. Buffer contiguity indicates how many packets are consecutive, starting from the head of the buffer, and can thus be promptly delivered to the VLC player. This is complementary with buffer occupancy which only indicates the number of packets and does not consider holes in the packet sequence that might affect the video quality. Note that buffer contiguity is slightly different from the continuity index [38] used in other papers. The continuity index is the ratio of the number of packets that arrive before the playback deadline to the total number of packets. It is used to summarize the packets received in time. Buffer contiguity is more expressive in the sense that it allows to observe, at each instant, the amount of packets that can be played right away. Figure 14 shows the results for 1 and 4 trees for static and moderate churn conditions. LAYSTREAM guarantees buffer contiguity even under churn. This is particularly remarkable when using several trees as, regardless of parallelism, the buffer stores a great portion of contiguous video packets.

V. RELATED WORK

There is a large body of research on live video streaming extensively covered in surveys such as [1], [22]. Here, we focus exclusively on gossip approaches to live streaming evaluated in realistic environments. Most approaches to live streaming, and video dissemination in general, follow a *pull* approach where nodes explicitly request data from others. The goal is to conserve bandwidth by avoiding the duplicate packet receptions flooding yields. By placing the burden of retrieving packets on receivers, pull-based approaches such as GoalBit [4], CoolStreaming [38] or dHCPS [14] effectively address this problem at the expense of increased latency and communication overhead. GoalBit [4] is inspired by BitTorrent and relies on a tracker and requires super nodes to do most of the dissemination whereas in LAYSTREAM all nodes participate equally without any centralized control. This challenge is also addressed in [13], which aims at leveraging the upload bandwidth of the nodes while avoiding clogging uplinks. The protocol segments the

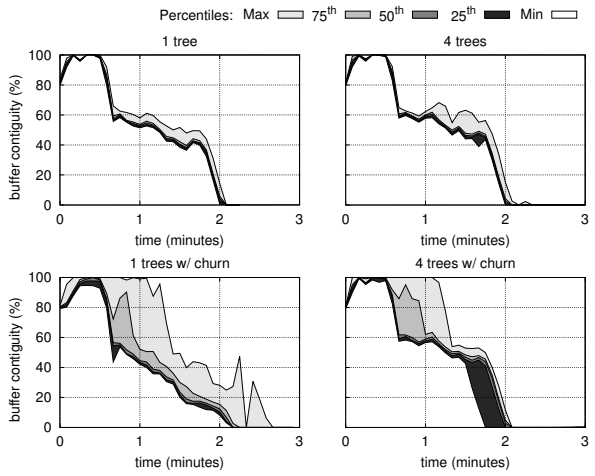

Fig. 14. Contiguous video segments in buffer.

video in large constant chunks, which is not particularly adapted for live streaming systems that must deal with small packets of non-predictable size packets due to the use of VBR [9].

The alternative to pulling data over unstructured overlays is to push it through trees or forests of trees. Since trees are fragile structures, they must be backed up by an efficient and fast repair mechanism. Approaches such as mTreebone [35] build a tree containing only the nodes known to be stable and use an unstructured overlay to cover all nodes. In this way, stable nodes can quickly receive the stream by pushing data over the tree while the remaining nodes pull data from the overlay.

Brisa [25] and Thicket [12] propose mechanisms to construct and maintain trees under dynamic environments. These principles are used by the LAYSTREAM DL for building forests of trees that encompass all nodes. Alternatively, approaches such as TURINstream [23] organize nodes into clusters that form the vertices of the tree which when coupled with *Multi Description Coding* results in robust dissemination in a dynamic environment. These three systems assume an arbitrary random network, without guarantees in terms of network awareness as LAYSTREAM does. We have shown in this paper that the layering of gossip protocols with specific purpose and well-defined guarantees allows us to also meet this requirement.

Finally, DP/LU [26] pursues a similar goal to ours as it assesses the feasibility of live streaming based solely on unstructured overlays. Despite robust, unstructured overlays are less suitable than trees to adapt to a network topology which is crucial for efficient low-latency streaming (even though DP/LU considers bandwidth constraints).

VI. CONCLUSION

In this paper we set out to build and deploy a demanding application — live video streaming — out of a set of gossip protocols existing in the literature. Instead of coming up with new solutions to the existing problems of membership management, topology construction and dissemination, we picked up existing protocols known to solve these problems independently. As expected, several non-trivial problems appeared during the implementation. These were due to the gap between theory and practice, that was already reported in other work [8], [24]. Note that this is not a fault of the selected protocols *per se*, but instead a limitation of the theoretical models and simulation tools used that necessarily involve simplifications of the reality. In fact, from our experience, we expect similar issues to emerge had the choice of protocols for each particular layer been a different one. Nonetheless, our results show that it is possible to build real systems out of existing gossip protocols attesting their maturity. We also hope the open availability of our implementation helps researchers reproduce our results and promote similar efforts with other gossip protocols.

ACKNOWLEDGMENTS

This work is part-funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP - 01-0124-FEDER-022701 and by Project Smart- grids - NORTE-07-0124-FEDER-000056, co-financed by the North Portugal Regional Operational Programme (ON.2 O Novo Norte), under the National Strategic Reference Framework (NSRF), through the ERDF.

REFERENCES

[1] O. Abboud, K. Pussep, A. Kovacevic, K. Mohr, S. Kaune, and R. Steinmetz. Enabling resilient P2P video streaming: survey and analysis. *Multimedia Systems*, 17(3):177–197, 2011.

[2] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni. Tera: topic-based event routing for peer-to-peer architectures. In *DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 2–13, New York, NY, USA, June 2007. ACM Press.

[3] L. Barba, P. Bose, M. Damian, R. Fagerberg, W. L. Keng, J. O'Rourke, A. van Renssen, P. Taslakian, S. Verdonschot, and G. Xia. New and improved spanning ratios for Yao graphs. In *SoCG'14: 30th Symposium On Computational Geometry*. ACM, 2014.

[4] A. Barrios, M. Barrios, D. De Vera, P. Rodríguez-Bocca, and C. Rostagnol. Goalbit: a free and open source peer-to-peer streaming network. In *Proceedings of the 19th ACM international conference on Multimedia*, MM '11, pages 727–730, New York, NY, USA, 2011. ACM.

[5] O. Beaumont, A.-M. Kermarrec, L. Marchal, and É. Rivière. VoroNet: A scalable object network based on voronoi tessellations. In *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS 2007)*, Long Beach, CA, USA, Mar. 2007.

[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, 1999.

[7] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: high-bandwidth multicast in cooperative environments. *ACM Symposium on Operating Systems Principles*, 37(5), 2003.

[8] T. D. Chandra, R. Griesemer, and J. Redstone. Paxos made live. In *Proceedings of the twenty-sixth annual*

ACM symposium on Principles of distributed computing - PODC '07, pages 398–407, New York, New York, USA, Aug. 2007. ACM Press.

[9] H. Chang, S. Jamin, and W. Wang. Live Streaming With Receiver-Based Peer-Division Multiplexing. *IEEE/ACM Transactions on Networking*, 19(1):55–68, Feb. 2011.

[10] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris. Practical, distributed network coordinates. In *Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II)*, Cambridge, Massachusetts, Nov. 2003. ACM SIGCOMM.

[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 *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM Press.

[12] M. Ferreira, J. Leitao, and L. Rodrigues. Thicket: A protocol for building and maintaining multiple trees in a p2p overlay. In *Reliable Distributed Systems, 2010 29th IEEE Symposium on*, pages 293–302. IEEE, 2010.

[13] D. Frey, R. Guerraoui, A.-M. Kermarrec, and M. Monod. Boosting gossip for live streaming. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1–10, 2010.

[14] Y. Guo, C. Liang, and Y. Liu. dHCPS: decentralized hierarchically clustered p2p video streaming. In *Proceedings of the 2008 international conference on Content-based image and video retrieval - CIVR '08*, New York, New York, USA, July 2008. ACM Press.

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

[16] M. Jelasity, A. Montresor, and O. Babaoglu. T-man: Gossip-based fast overlay topology construction. *Computer Networks*, 53(13):2321–2339, aug 2009.

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

[18] I. Kanj and G. Xia. On certain geometric properties of the yao–yao graphs. *Journal of Combinatorial Optimization*, pages 1–10, 2012.

[19] A.-M. Kermarrec, A. Pace, V. Quema, and V. Schiavoni. Nat-resilient gossip peer sampling. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, ICDCS '09, pages 360–367, Washington, DC, USA, 2009. IEEE Computer Society.

[20] J. Leitão, J. Pereira, and L. Rodrigues. Hyparview: a membership protocol for reliable gossip-based broadcast. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 419–428, Edinburgh, UK, June 2007.

[21] L. Leonini, E. Rivière, and P. Felber. SPLAY: Distributed systems evaluation made simple. In *NSDI'09: Proceedings of the 6th Symposium on Networked Systems Design and Implementation*, pages 185–198. USENIX, April 2009.

[22] Y. Liu, Y. Guo, and C. Liang. A survey on peer-to-peer video streaming systems. *Peer-to-Peer Networking and Applications*, 1:18–28, 2008.

[23] A. Magnetto, R. Gaeta, M. Grangetto, and M. Sereno. TURINstream: A Totally pUsh, Robust, and effIcieNt P2P Video Streaming Architecture. *IEEE Transactions on Multimedia*, 12(8):901–914, Dec. 2010.

[24] F. Maia, M. Matos, J. Pereira, and R. Oliveira. Worldwide consensus. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 257–269. Springer-Verlag, June 2011.

[25] M. Matos, V. Schiavoni, P. Felber, R. Oliveira, and E. Rivire. Lightweight, efficient, robust epidemic dissemination. *Journal of Parallel and Distributed Computing*, 73(7):987 – 999, 2013.

[26] F. Picconi and L. Massoulié. Is there a future for mesh-based live video streaming? In *Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on*, pages 289–298. IEEE, 2008.

[27] R. V. Renesse, K. 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, May 2003.

[28] R. V. Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. *Middleware Conference*, pages 55–70, 2009.

[29] E. Rivière, R. Baldoni, H. Li, and J. Pereira. Compositional gossip. *ACM SIGOPS Operating Systems Review*, 41(5):43, Oct. 2007.

[30] A. Stavrou, D. Rubenstein, and S. Sahu. A lightweight, robust P2P system to handle flash crowds. In *IEEE Journal on Selected Areas in Communications*, 2004.

[31] S. Voulgaris, D. Gavidia, and M. V. Steen. CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.

[32] S. Voulgaris, M. Jelasity, and M. van Steen. A robust and scalable peer-to-peer gossiping protocol. In G. Moro, C. Sartori, and M. Singh, editors, *Agents and Peer-to-Peer Computing*, volume 2872 of *Lecture Notes in Computer Science*, pages 47–58. Springer Berlin Heidelberg, 2005.

[33] S. Voulgaris, E. Rivière, A.-M. Kermarrec, and M. van Steen. Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks. In *Proceedings of IPTPS'06: 5th International Workshop on Peer-to-Peer Systems*, Santa Barbara, USA, feb 2006.

[34] S. Voulgaris and M. van Steen. Vicinity: A pinch of randomness brings out the structure. In *Middleware 2013*, pages 21–40. Springer, 2013.

[35] F. Wang, Y. Xiong, and J. Liu. mTreebone: A Collaborative Tree-Mesh Overlay Network for Multicast Video Streaming. *IEEE Transactions on Parallel and Distributed Systems*, 21(3):379–392, Mar. 2010.

[36] Y. Wang and X.-Y. Li. Distributed spanner with bounded degree for wireless ad hoc networks. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IPDPS'02, Washington, DC, USA, 2002. IEEE Computer Society.

[37] A. C.-C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

[38] X. Zhang, J. Liu, B. Li, and Y.-S. Yum. CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 2102 – 2111 vol. 3, march 2005.