CHAMS: Churn-Aware Overlay Construction for Media Streaming

ABSTRACT

Overlay networks support a wide range of peer-to-peer media streaming applications on the Internet. The user experience of such applications is affected by the *churn resilience* of the system. When peers disconnect from the system, streamed data may be delayed or lost due to missing links in the overlay topology. In this paper, we explore a proactive strategy that constructs a *churn-aware* overlay network that reduces the potential of future disruptions caused by churn events.

We describe CHAMS, a middleware solution for overlay network construction that mitigates the impact of churn. CHAMS uses a hybrid approach that implicitly defines an overlay topology using a gossip-style mechanism that takes the reliability of peers into account. Unlike existing solutions for reliable overlay construction, CHAMS supports a variety of topologies used in media streaming systems, such as trees, multi-trees and forests, and can be applied to existing systems. We evaluate CHAMS with different topologies and show that it reduces the impact of churn, while imposing low computational and message overheads.

Keywords: application-level multicast, peer-to-peer systems, overlay networks, media streaming, churn resilience.

1. INTRODUCTION

Video streaming is one of the most popular Internet applications. It is expected to constitute 90% of Internet traffic by 2013 according to Cisco. The majority of commercial video streaming services such as *YouTube* and *Google Video* are based on Content Delivery Networks (CDNs). Content is first pushed to a set of strategically placed content delivery servers and consumers then stream content from nearby servers. A CDN-based solution avoids the bottleneck of a central server, can achieve lower streaming latency, reduces network traffic and can serve more users. Its main challenge is the amount of resources required to serve a global usage base, making deployments expensive. To deliver streaming video with good quality, the bandwidth provisioned at content delivery servers must be proportional to the number of consumers. To cope with this requirement, commercial CDN operators such us *Akamai* and *Real Networks* operate a costly dedicated large-scale infrastructure.

Peer-to-peer (P2P) networks emerged as an alternative for providing media streaming services. In contrast to CDN-based streaming, users (also named peers) not only download content but also upload it to other users. The advantage of leveraging a user's upload bandwidth makes this architecture more cost-effective compared to CDNs. Application-layer multicast (ALM) systems, such as Bullet [16], Splitstream [7], Overcast [13] and CAN Multicast [22], can provide a P2P streaming service. They must ensure properties necessary for high-quality streaming, namely maximising bandwidth and minimising latency. Typically, ALM systems construct a logical overlay network through which streaming packets are routed. Many overlay topologies have been proposed: trees [13, 2, 9], meshes [16, 4] and forests [7]. Among existing topologies, tree-like overlays (i.e., single tree, multi-trees and forests of trees) are widely used for their efficiency.

The construction of overlay networks follows some rules in order to achieve the desired properties in terms of high bandwidth and low latency. These goals must be combined with the desire to maximise reliability, which is critical for high-quality streaming. A P2P system operates in a dynamic environment, in which peers can fail or leave the system abruptly. The event of a peer leaving or failing is referred to as *churn* [9]. In a tree-like overlay, a churn at an internal peer causes a service disruption on its descendants in the tree. Frequent service disruptions reduce user experience and are thus unacceptable. To handle this, one could employ a *reactive* solution [7, 16, 13, 19]: repair the tree overlay by finding a new parent for orphaned peers. This approach is efficient but it reduces streaming throughput due to the time needed to heal the tree. An interesting question is *"whether one can minimise the impact of churn using a **proactive** solution"*, which we address in this paper.

Our goal is to improve the reliability of ALM overlays in spite of the unreliable nature of peers. As a solution, we describe CHAMS, a middleware for overlay construction and maintenance that creates reliable tree-like overlays that support high-quality streaming. CHAMS can increase the reliability of existing ALM systems, such as Bullet [16] and SplitStream [7]. For scalability reasons, CHAMS uses a *hybrid* approach to define and maintain a reliable overlay implicitly using gossip-style communication. Each peer selects a set of peers to gossip with that form the desired overlay topology

and are chosen based on the perceived reliability of peers. To take the reliability of peers into account, we quantify it as a peer's age [23]. While hybrid topology construction was adopted by researchers in the past [6, 17, 25], previous efforts were not churn-aware when building overlays, thus reducing the practical applicability of this approach.

In summary, the contributions of this work are:

- *A proactive approach for churn-aware overlays.* We define reliable tree-like overlays that can be used as the basis of a P2P streaming service. Our solution maximises overlay reliability by taking individual churn risk at each peer into account.
- *The integration of churn-awareness with hybrid topology construction.* We describe an extension of a hybrid topology construction algorithm that makes decisions about how to connect peers based on estimated peer reliability, creating and maintaining reliable tree-like overlays.
- *An experimental evaluation based on real-world P2P traces.* We show through simulation results based on traces from real P2P streaming systems that CHAMS is feasible and efficient. CHAMS reduces the number of churn disruptions while incurring a low latency penalty.

The rest of this paper is organised as follows. Section 2 discusses related work. Section 3 states our assumptions. In Section 4, we describe the architecture of CHAMS and Section 5 gives details of the employed algorithms for overlay construction. We evaluate CHAMS in Section 6 and conclude in Section 7.

2. BACKGROUND

Next we describe related work by classifying existing ALM solution based on their approach. We also discuss how churn resilience is achieved in previous proposals for P2P overlay construction and maintenance.

Application-layer multicast. To support P2P streaming, a plethora of application layer multicast (ALM) solutions [1] have been proposed with varying properties in terms of throughput, scalability, reliability and delay. A high-level classification of systems defines two main types: gossip-based [5, 15, 11, 18] and overlay-based [16, 7, 9, 13] approaches.

Gossip-based. A gossip protocol can be *pull-based* [11] or *push-based* [15, 5]. In pull-based gossip, peers exchange information about which messages they have received. When a peer p learns through gossip exchange that it misses a message m, then p sends an explicit request for the message m to another peer q, and q replies by forwarding m to p.

In *push-based* gossip, when a peer wants to disseminate a message, it selects a set of peers at random based on a *fanout* parameter and sends the message to them [15]. When receiving a message for the first time, each peer repeats that process, while excluding the sender from a set of randomly selected neighbours. As a consequence, the path followed by the diffused message is not deterministic.

Overlay-based. In overlay-based systems, peers are first organised into an overlay network with a given topology, such as a tree, and then messages are routed along this topology. The overlay network is constructed with given constraints on network properties such as latency and bandwidth that are determined by an optimisation goal. Existing proposals differ in their optimisation goals [1], including minimising latency [17, 19, 9], maximising throughput [13, 16, 7] and achieving scalability [22, 4], churn resilience or reliability [8, 2, 12].

These two approaches have a tradeoff in terms of *scalability* and *resilience* vs. *adaptiveness* and *efficiency*. Gossip-based protocols scale well because they balance load among all peers in the system. However, gossip-based protocols are less efficient than overlay-based ones because they suffer from increased network traffic due to the redundancy in the push-based gossip or the exchange of received message identifiers between peers in pull-based gossip. This is the price to pay for avoiding the cost of topology construction and maintenance in overlay-based approaches. On the other hand, overlay-based approaches can adapt well to heterogeneity. Since gossip-based approaches are random in nature, it is difficult for them to take different node resources and network properties into account.

Hybrid. For the best of both worlds, hybrid approaches [6, 17, 25, 3, 10, 14] combine gossip- and overlay-based strategies. A hybrid protocol diffuses messages, adapting to node and network constraints, similar to an overlay-based protocol, while not imposing an overhead through overlay construction. To do so, a hybrid protocol uses a gossip-style mechanism, with deterministic behaviour in the gossip decision. Contrary to traditional random gossip, the subset of selected peers to gossip with is chosen deterministically by taking peer properties and constraints into account, such as the desired overlay topology. The union of links between selected gossip peers *implicitly* defines the required overlay.

Examples of hybrid protocols are Plumtree [17], which constructs a minimum latency tree, RASM [3], which defines a maximum reliability tree and Thicket [10], which defines a forest of minimum latency trees. In contrast to these efforts, CHAMS focusses on the reliability of the overlay and is not restricted to a single topology but supports different tree-like overlays. T-Man [14] also supports the construction of different overlay topologies using a hybrid approach, such as trees and ring. However, it constructs overlays with considering underlying peer properties such as reliability. Another relevant gossip-style overlay construction algorithm is the Scamp protocol [21], which defines a proximity optimised overlay in order to reduce network load. In contrast to CHAMS, the overlay defined by Scamp is unstructured, which is unsuitable for ALM systems that rely on a given topology. In addition, Scamp handles churn reactively, whereas CHAMS addresses churn proactively in order to minimise disruption during churn events.

Churn resilience. To provide churn-resilient P2P streaming, most existing solutions adopt a *reactive* approach [16, 7, 13, 19], healing the overlay when churn occurs. Based on the non-determinism in a P2P environment with probabilistic peer departure, it is necessary to react to disconnecting peers. While being efficient, a reactive approach leads to a reduction in throughput due to the time required to repair the overlay.

Proposals exist to address churn in a *proactive* manner [3, 2, 26, 24, 23, 28], attempting to reduce the impact of churn as much as possible. Overlay topologies are constructed by taking the likelihood of peer departure into account. Ex-

isting works [2, 12] consider a P2P overlay network as a probabilistic model, in which nodes can fail and links may lose messages with a given probability, approximated using Bayesian networks.

Peer reliability can be expressed in different ways. Sripanidkulchai et al. [23] show that peer lifetimes exhibit a heavy-tailed behaviour in real P2P applications, with a small number of peers having very long lifetimes. Based on the observation that old peers are more likely to stay longer in the system, it is possible to define a heuristic that regards old peers as more reliable. For P2P streaming, a peer that relies on old peers to receive stream packets reduces the risk of churn, hence improving the quality of the streaming.

In IRP [26], the authors use an estimate of peer reliability based on the age of peers. Long-lived peers are considered more reliable and are moved upwards in the tree overlay. In ROST [24], peer reliability is calculated as the product of the age and the outgoing bandwidth. This mitigates the depth of the constructed tree and reduces the overhead imposed to maintain the tree. Our approach for churn-awareness is also proactive, predicting churn risk based on observed peer age. In contrast to IRP and ROST, which build a single tree overlay, CHAMS can construct a variety of overlay topologies, including single trees, multi-trees and forests of disjoint trees.

3. P2P MODEL

We consider a P2P overlay network composed of peers that communicate by message passing. More formally, we model the overlay topology as a connected graph with a set of n peers connected with a set of bidirectional links.

Local peer view. To obtain a scalable solution, we assume that a peer p_i knows only its direct neighbours, denoted as N_i. At each peer p_i, the set of direct neighbours N_i represents the peers with which p_i has a peering relationship. The establishment and management of peering relationships is part of a *peering membership protocol*, which is the protocol executed by peers to join the P2P network and maintain a number of peering neighbours.

Reliability. Based on the analysis in [27, 23], we adopt the heuristic that if a peer has been participating in the system for a long time, it is more likely to remain. We argue that this heuristic is reasonable given the heavy-tailed distribution of peer lifetimes.

Defining the reliability of each peer p_i as function of p_i's age requires continuous updates of this metric because the change in age. To avoid this, we define the reliability metric of a peer p_i as p_i's *join time*, denoted as $joinTime_i$. Contrary to the age, the reliability of a peer is inversely proportional to its join time—the lower the join time, the more reliable the corresponding peer is. Thus, the lower $joinTime_i$, the more reliable p_i is deemed to be. To be able, to compare different peers reliabilities metrics, we assume that join time values rely on a global clock among all peers.

From the definition of peer reliability metric, we can obtain a reliability metric of a path based on the reliability metrics of the peers along that path. More formally, we define the reliability metric of a path PR a the sum of individual *joinTime* of peers along that path:

$$PR = \sum_{p_i \in path} joinTime_i. \quad (1)$$

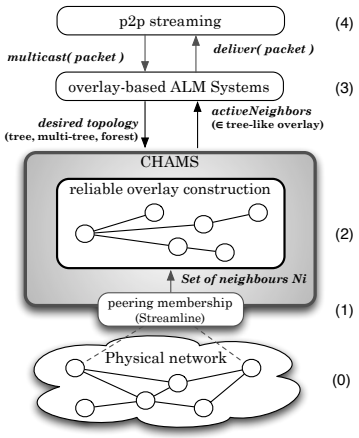


Figure 1: Overview of the Chams architecture

Similarly to the *joinTime* interpretation, the lower the value of PR of a path, the more reliable it is.

4. CHAMS MIDDLEWARE

CHAMS provides a middleware that handles overlay construction and maintenance for existing ALM systems. It constructs a churn-aware overlay that enhances the reliability of P2P media streaming. CHAMS can be used with existing ALM systems because it supports a range of tree-based overlay topologies such as trees, multi-trees and forests. By factoring out overlay construction, CHAMS simplifies the implementation of ALM systems and improves their reliability by proactively addressing churn.

Figure 1 illustrates how CHAMS supports ALM systems for P2P streaming. It provides a churn-aware overlay to an ALM system (layer 3), which in turn is used to build a P2P streaming application (layer 4). The streaming layer (layer 4) is responsible for partitioning the outgoing stream into stream packets containing video chunks of a certain length. These packets are relayed by the ALM system (layer 3).

During topology construction, CHAMS takes the reliability of the underlying network peers (layer 0) into account and obtains the required overlay topology in a distributed fashion using a hybrid approach (layer 2). This process leverages a peer membership protocol (layer 1), which associates each peer with a set of neighbours and notifies a peer of their arrival or departure. Next we describe the functionality in each layer in more detail and explain their interactions.

(1) Peering Membership. In a P2P system, a joining peer executes a peering membership algorithm in order to establish a peering relationship with a set of the existing peers. CHAMS uses the peering membership protocol defined in Streamline [20], which builds peering relationships in an adaptive manner.

To establish peering relationships, Streamline defines a set of candidates from the existing peers and orders them according to weight. The weight a peer determines the probability of its selection for the peering relationship. It is proportional to the peer bandwidth and the number of already existing neighbours. Therefore the connectivity of each peer depends on the available bandwidth of the peer. Based on that, we assume that each peer has enough bandwidth to gossip with all its direct neighbours.

As shown in Figure 1 (layer 1), Streamline informs each peer about the set of its direct neighbours, denoted as N_i.

To keep a peer's view of N_i up-to-date, Streamline notifies each peer when a change occurs in its neighbourhood.

(2) Reliable Overlay Construction. CHAMS follows a hybrid approach to create an overlay network implicitly by operating a gossip protocol. To build and maintain an overlay, CHAMS performs a periodic gossip of *control messages*, denoted as *cmsg*. The gossip protocol is deterministic—each peer maintains a subset of its neighbours, referred to as *activeNeighbours*, with which it exchanges control messages *cmsg*. The choice of neighbours to be added to the *activeNeighbours* set ensures that the union of links among those peers form the required overlay topology, such as a single tree, multi-tree or forest of trees.

Gossip-style mechanism. The periodic gossip of *cmsg* messages has two purposes: it constructs the desired overlay topology and it heals the overlay in case of disconnection. The gossip of a control message *cmsg* starts by having the stream source send *cmsg* to all its neighbours in its *active Neighbours* set. Initially, each *activeNeighbours* set of a peer p_i is initialised with the set of its direct neighbours N_i. Thus, the gossip of the first control message $cmsg_1$ is performed using flooding, where each peer sends $cmsg_1$ to all neighbours in its *activeNeighbours* set.

Reliable overlay construction. Some peers receive duplicates of control message $cmsg_1$. Based on reliability information, some of these duplicate paths are pruned in the overlay so that only paths with low PR are kept. In CHAMS, a peer p_i prunes paths with undesirable properties by removing direct neighbours that provided messages on those paths from its *activeNeighbors* set. Peer p_i also sends a *prune* message to these neighbours so that they will not forward subsequent control messages to p_i.

Overlay connectivity & healing. To ensure the connectivity of the overlay, a periodic exchange of information about the received *cmsg* messages is performed between each peer and its other neighbours (i.e., the neighbours that are not in the *activeNeighbours* set). We refer to this subset of direct neighbours as the *backupNeighbours* set. Based on this exchange, a peer can detect if it has missed *cmsg* messages. This may indicate that it has been disconnected from the overlay. Typically, a peer is disconnected when churn occurs at one of its ancestor peers in the upstream path.

To heal the overlay, the peer asks its neighbours within its *backupNeighbors* set that hold the missed *cmsg* messages for recovery. In turn, when a peer receives such a request, it first sends back the missed *cmsg* and moves the requesting peer to its *activeNeighbors* set. By doing so, the disconnected peer is implicitly reconnected to the overlay because the overlay is defined as the union of links among *activeNeighbors* peers.

(3) Application-Level Multicast. When a peer in the ALM system wants to send a stream packet, it asks CHAMS for the subset of neighbours to which the packet should be propagated. The union of links between that peer and this subset of its neighbours is guaranteed to be part of the overlay requested by the ALM system. The constructed overlay remains scalable because the full topology is not known to any single peer but it is defined implicitly by the union of the *activeNeighbours* sets. Therefore, CHAMS never has to expose the complete overlay topology to the ALM system.

5. THE CHAMS ALGORITHM

In this section, we describe the algorithm at the heart of CHAMS middleware (Algorithms 1 and 2). Various tree-like structures can be constructed and maintained using CHAMS, by passing it the desired overlay topology as parameter, as suggested in Figure 1. Hereafter, we give a brief description of each tree-like topology that CHAMS can build.

Tree. This structure is widely used [13, 2, 16] because its acyclicity simplifies routing and avoids redundancy which saves node resources and limits produced network traffic.

Multi-trees. This structure includes several trees covering the same set of peers but rooted at different source peers. This topology is particularly used in P2P multi-party conferencing applications.

Forest of disjoint trees. This structure was first proposed by [7]. It represents a special type of multi-tree structure with one source but where each node is internal in only one tree and is a leaf in the other trees. Based on this condition, the trees of the forest structure are *disjoint*. In [7], the stream is divided into multiple sub-streams named *stripes*. Each stripe is routed through a dedicated disjoint tree of the forest. The number of stripes is noted K. In our algorithms, we also use this notation to refer to the forest's trees and their number.

5.1 Data Structures & Initialisation

Algorithm 1 depicts the data structure maintained at each node performing CHAMS as well as its initialisation. As explained in the previous section, the implicit definition of the reliable overlay relies on the diffusion of control messages *cmsg*. Each control message *cmsg* being diffused holds a set of information necessary for the overlay definition. This information includes the source node that initiates the diffusion of the current *cmsg*, the neighbour that forwards this *cmsg*, *sender*, and the reliability metric PR of the path serving this message. This metric, computed according to Equation 1, represents the key information that influences the implicit definition of the reliable overlay.

To define a tree-like overlay, we associate each tree of the overlay with a *flow* of control messages *cmsg* (line 6). Each node maintains a set of flows corresponding to the trees on which that node is included. In the single tree overlay, only one flow is defined at each node. In the multi-tree overlay, the number of flows corresponds to the number of trees; in the forest overlay, the number of flows corresponds to the number of stripes of the forest. Each *flow* has a unique ID, which is either the id of the source node—in the case of tree and multi-tree topologies—or the id of the stripe for the forest topology.

At each node, a *flow* data structure includes the set *active Neighbors*, with which the current node gossips stream packets and our control messages *cmsg*, and the set *backup Neighbors* used to ensure connectivity of the tree. In addition, each *flow* data structure indicates the reliability metric PR perceived so far.

A flow is created first at the source of the corresponding tree (line 11). For the case of *forest*, a flow is created for each stripe (line 14). In our algorithm, we assume that the number of stripes in the forest is K is given. Note that, when a flow is initialised, all neighbours are in the *backupNeighbours* set. Before disseminating control messages *cmsg*, a subset of these neighbours moves to the *activeNeighbours* by calling the $initActiveNeighbor()$ procedure. The subset of neigh-

bours to include to the *activeNeighbours* set depends on the required topology.

5.2 Parameterisation by Tree-like Topologies

The initialisation of the *activeNeighbours* set depends on the target topology overlay, i.e., tree, multi-tree or forest. In the case of single tree of multi-trees, this set is initialised with the set of direct neighbours N_i (line 33). This reflects the fact that our periodic gossip starts initially as a flooding where the first control message is sent to all neighbours, as explained in Section 4. For the forest topology, however, the disjointedness condition should be verified. That is, each node can be an internal node in only one flow. At a node p_i, this means that p_i can have only one flow where the size of its *activeNeighbours* set exceeds 1. In that flow, the *activeNeighbours* set is also initialised with the set of direct neighbours N_i (line 30). For any other flow $f\prime$, this set contains only one element—the neighbour that provides p_i with $f\prime$ control messages (line 29).

Algorithm 1 : data structures & initialisation at p_i

```
1: uses: Streamline
2: input: topology
3: input: K

4: data structure: cmsg                    {periodically diffused message}
5:   fields: mID, flowID, source, sender, PR
6: data structure: Flow                    {flow of cmsg messages}
7:   fields: ID, source, activeNeighbors, backupNeighbors,
     sender, PR, receivedMsgs
8: joinTime_i ⟵ ...                        {join time of p_i}

9: initialization:
10:  flows ⟵ ∅                             {set of flows}
11:  at source:
12:    if topology = Forest then
13:      for all stripeID ∈ K do
14:        f ⟵ new Flow(stripeID,p_i,∅,N_i,p_i, joinTime_i,∅)
15:        flows ⟵ flows ∪ f
16:        call initActiveNeighbor(f)
17:    else
18:      f ⟵ new Flow(p_i.ID, p_i, ∅, N_i, p_i, joinTime_i, ∅)
19:      flows ⟵ flows ∪ f
20:      call initActiveNeighbor(f)

21: procedure initActiveNeighbors(f)
22:  if topology=forest then
23:    if f.source = p_i then
24:                                {At the source of the forest.}
25:      let n: ∄ f′ ∈ flows | n ∈ f′.activeNeighbors
26:      f.activeNeighbors ⟵ n
27:      f.backupNeighbors ⟵ N_i \ n
28:    else
29:      if (∄ f′ ∈ flows : | f′.activeNeighbors | > 1) then
30:        f.activeNeighbors ⟵ N_i
31:        f.backupNeighbors ⟵ ∅
32:  else
33:    f.activeNeighbors ⟵ N_i
34:    f.backupNeighbors ⟵ ∅
```

5.3 Reliable Overlay Construction

Algorithm 2 depicts the implicit construction of a tree-like overlay. As already mentioned, this construction relies on the diffusion of control messages, which is initiated periodically at the source node (line 1). During each period, the source node creates a new control message, noted m, (line 4) and gossips it to its active neighbours (line 6). Each control message is tagged with a unique identifier and stored in the *receivedMsgs* set of the corresponding *flow* (lines 17 and 28).

When a node p_i receives a control message m from a neighbour *sender* (line 12), it first checks whether it already has the corresponding flow f. This means that p_i checks if it is already included in the tree of f. If not, then the new flow is added to the set of flows known at p_i (line 15). After that, p_i initialises its set of active neighbours in f (line 16).

If the flow f already exists, p_i checks whether the current flow is still served by the same sender neighbour (line 22). The sender of the flow may change due a change in the underlying P2P network, e.g., a churn event occurs. In this case, p_i simply updates the information related to f with the new sender and reinitialises its set of its active neighbours (lines 23 to 26).

Next p_i checks whether the current control message m has already been received by checking in the *receivedMsgs* set (line 27). If m is received for the first time, then it is added to the *receivedMsgs* set and gossiped to p_i's neighbours in the *activeNeighbors* set (line 29). During this gossip, when a node sends a message m to a neighbour, it also sends additional information representing the reliability of the path traversed by m (line 9). The reliability metric PR is computed iteratively at each node p_i that forwarded m by incrementing m's PR with p_i's $joinTime_i$.

Otherwise, if m is a duplicate sent by a new neighbour, it corresponds to an alternative branch for including p_i in the current tree. As already mentioned, it is the reliability information brought by the duplicate messages that allows to select the most reliable branch as part of the tree. To decide whether to switch to this new branch as part of the current tree or to keep its current branch p_i compares the current reliability metric $f.PR$ (line 31) of the flow f and the reliability metric $m.PR$ of the new path serving f's messages via another neighbour *sender*.

When PR via *sender* is lower, a switch to this more reliable branch is performed (lines 32–36). To switch under the *sender*, p_i keeps *sender* as its flow provider by adding it to its *activeNeighbors* set and prunes the previous path of f's tree. This pruning consists of removing previous sender $f.Sender$ from the *activeNeighbors* set and sending a *prune* message to it.

Upon receiving a *prune* message (line 42) from a neighbour *sender*, p_i also removes *sender* from its *activeNeighbors* set, hence p_i will not propagate the following control messages to *sender*. This mechanism implicitly defines the tree part of the tree-like overlay that includes the most reliable paths.

5.4 Overlay Healing

The selection of the most reliable paths to be part of the overlay does not completely avoid overlay partitions but only minimises the number of times that they occur. Thus, a *reactive* strategy to reconnect the reliable overlay when a churn occurs is still required. Here, we briefly describe a simple healing mechanism inspired by [17, 10].

First, the detection of a partition in tree of flow f relies on the periodic exchange of a summary of the received *cmsg* messages between each node and its neighbours in the *backupNeighbors* set of f. When a node p_i receives a summary, it verifies if all indicated messages exist in its f's *receivedMsgs* set. For each, missed message *missed*, p_i waits for a timeout to receive it through its current flow sender. If the message *missed* has not been received, p_i assumes that it either has never been included in the tree of f (e.g., a newly joining peer) or has been disconnected from that tree.

To incorporate the tree, p_i sends a *Graft* message to all *backupNeighbours* that announced the message *missed* to p_i

Algorithm 2 : periodic gossiping at p_i

```
1: To update overlay, do periodically at source:
2:   for all f ∈ flows: f.source = p_i do
3:     mID ⟵ hash(f.ID, seqNbr)
4:     m ⟵ new cmsg(mID, f.ID, p_i, p_i, 0)
5:     f.receivedMsgs ⟵ f.receivedMsgs ∪ {m}
6:     call gossip(m, p_i, p_i, f.ID)

7: procedure gossip(m, sender, source, flowID)
8:   for all p_j ∈ flows.get(flowID).activeNeighbors: p_j ≠ sender
     do
9:     m.PR ← m.PR + joinTime_i   {the reliability metric
       PR to p_j via p_i}
10:    m.sender ⟵ p_i
11:    Send(GOSSIP, m, source, flowID) to p_j

12: upon Receive(GOSSIP, m, source, flowID) from sender do
13:   if ∄ f ∈ flows : f.ID = flowID then
14:     f ⟵ new Flow(flowID, source, {sender}, N_i \ {sender},
       sender, m.PR)
15:     flows ⟵ flows ∪ f
16:     call initActiveNeighbor(f)
17:     f.receivedMsgs ⟵ f.receivedMsgs ∪ m
18:     call gossip(m, f.sender, f.source, f.ID)
19:   else
20:     let f ∈ flows : f.ID = flowID
21:     f.PR ⟵ m.PR
22:     if f.sender ≠ sender then
23:       f.activeNeighbors ⟵ {sender}
24:       f.backupNeighbors ⟵ N_i \ {sender}
25:       f.sender ⟵ sender
26:       call initActiveNeighbors(f)
27:     if ∄ e ∈ f.receivedMsgs : e.mID = m.mID then
28:       f.receivedMsgs ⟵ f.receivedMsgs ∪ m
29:       call gossip(m, f.sender, f.source, f.ID)
30:     else
31:       if m.PR <f.PR then
32:         f.activeNeighbors ⟵ f.activeNeighbors ∪ {sender}
33:         f.backupNeighbors ⟵ f.backupNeighbors
           \ {sender}
34:         f.activeNeighbors ⟵ f.activeNeighbors \ {f.sender}
35:         f.backupNeighbors ⟵ f.backupNeighbors
           ∪ {f.sender}
36:         Send(PRUNE, f.ID) to f.sender
37:         f.sender ⟵ sender
38:         f.PR ⟵ m.PR
39:       else
40:         Send(PRUNE, f.ID) to sender
41:         activeNeighbors ⟵ activeNeighbors \ {sender}

42: upon Receive(PRUNE, flowID) from sender do
43:   let f ∈ flows : f.ID = flowID
44:   f.activeNeighbors ⟵ f.activeNeighbors \ {sender}
45:   f.backupNeighbors ⟵ f.backupNeighbors ∪ {sender}
```

in their summaries. When receiving a *Graft* message from p_i, the p_i's neighbour moves p_i from its *backupNeighbours* set to its *activeNeighbours* set and sends the control message *missed* to p_i. Since the *missed* message can be announced by several neighbours, p_i could be reconnected by several *backupNeighbors*. In this case, our mechanism selects the branch with the best reliability. Note that to preserve the required topology, e.g., a forest, only peers that are willing to accept new children announce their summary of the received *cmsg* messages periodically. In other words, a peer p_i will not announce received messages of flow f if p_i is unable to serve neighbours with f's messages.

6. EVALUATION

We present experiments designed to evaluate the benefit of CHAMS. Our evaluation goals are to investigate the gain in overlay reliability when using CHAMS and the latency of the constructed overlays. Our results show that CHAMS can build reliable overlays that reduce significantly the impact of churn in terms of disruptions experienced by peers. In terms of latency impact, overlays built by CHAMS have depth close to overlays designed to minimise latency.

Due to space constraints, we focus on single tree and forest overlays. Given the simplicity of the multi-tree overlays, which are multiple instances of single trees, their behaviour is similar to single tree overlays.

6.1 Evaluation Set-up

We evaluate the performance of CHAMS using the Sinalgo simulator. Sinalgo acts in rounds, which we consider as our time unit. In each round, a node receives and sends messages from/to its direct neighbours.

Our reliability definition is related to peer lifetimes. In simulation, we obtain join and departure times of peers from real P2P application traces. Next we describe the traces and the peer connectivity that defines the number of direct neighbours of each peer.

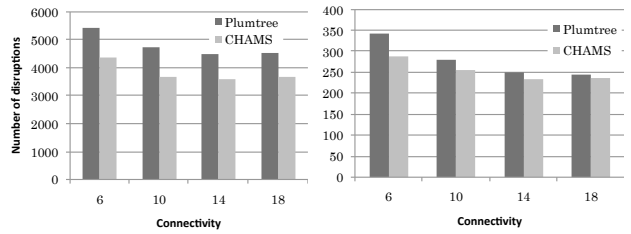
Traces. To consider various types of P2P networks, we select traces with different sizes in terms of numbers of peers. *PPLive:* This trace comes from the PPLive streaming application. They were collected and analysed after taking a snapshot of the PPLive network with 3449 peers[29]. The measurement includes two video channels. We use a trace of a single channel (referred to as *PPLive-ch1* in [29]). *Bittorrent:* This trace was collected from a Bittorrent file sharing application in 2005. It consists of the file distribution of the SlackWare Linux distribution and includes 14 Bittorrent swarms. We only use swarm number 7 (referred to as *T705'P2P-S7* in [29]) with 226 peers.

Connectivity. The peer connectivity (i.e., the number of direct neighbours at each peer) impacts the structure of the built overlay. When peer connectivity is low, the tree-like overlay tends to be "long and skinny". Conversely, when connectivity is high, the overlay tends to be compact. To show the effect of peer connectivity, we vary this parameter, denoted as c, which is the number of peers that a newly joined peer connects to.

Protocol comparison. To show the reliability gain of CHAMS, we compare it to other hybrid protocols that define the same overlay topology optimised for latency. For single tree construction, we compared to PLUMTREE [17], which defines a minimum latency spanning tree. For the forest overlay construction, we use THICKET [10], which implicitly constructs a forest of trees with minimum latency. We implement PLUMTREE and THICKET in the Sinalgo simulator[1] and execute them in the same scenarios as CHAMS.

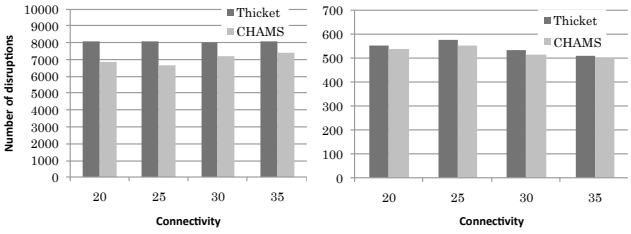
Note that both of these protocols have an advantage in terms of minimising the impact of churn. They construct a minimum latency tree, with each peer keeping the path through which it received messages first. This path tends to minimise the number of intermediate peers. Thus, the tree built by PLUMTREE and the forest built by THICKET tend to be shallow. In such an overlay, the number of disruptions due to churn is reduced because the included paths have small numbers of internal peers and the number of leaves peer is high. When churn occurs at leaf peer, it does not result in a disruption. Defining a minimum depth tree to enhance the reliability was studied in [23]. However, this

[1] `http://dcg.ethz.ch/projects/sinalgo`



(a) *PPLive* trace (b) *Bittorrent* trace

Figure 2: Number of disruptions due to churn in a single tree topology.



(a) *PPLive* trace (b) *Bittorrent* trace

Figure 3: Number of disruptions due to churn in a forest topology ($K = 2$)

work does not use a hybrid approach to build such an overlay.

6.2 Reliability Gain

To show the reliability gain, we measure the total number of disruptions seen by peers during the whole simulation. An disruption is caused by a churn event. When such an event occurs at a peer, i.e., when it leaves the system, all its descendant peers in the overlay observe a disruption. Note that, as argued in Section 1, a disruption does not mean an end of the streaming service. When churn occurs, our reactive healing mechanism reconnects the overlay and hence reestablishes service availability. The aim of CHAMS is to reduce the number of times that this reactive mechanism has to be invoked.

Single tree. Figure 2 shows the total number of disruptions experienced by peers in a single tree overlay. The number of disruptions when using the CHAMS tree is significantly lower compared to the PLUMTREE overlay. In both cases, the number of disruptions decreases as the connectivity c increases. As c increases, the reliability of both CHAMS tree and PLUMTREE tree goes up. This is because, as the number of links in the overlay increases, more links associated with peers are available. In PLUMTREE, this enables the construction of a more shallow tree with more leaf peers. In CHAMS, the tree is more reliable due to larger choice of alternative paths. As the connectivity increases, more links are created, offering a larger choice of paths.

Note, however, that the reliability gain of CHAMS against PLUMTREE decreases when the connectivity c increases. For instance, the number of disruptions seen in the *Bittorrent* trace when c is high (e.g., $c = 18$) is almost the same for both approaches. As we discuss in Section 6.4, this is due to the trade-off between the depth of a path and its reliability.

Forest of disjoint trees. In Figure 3, we show the total number of disruptions seen by peers in a forest overlay composed of two disjoint trees, i.e., $K = 2$. As can be seen, the number of disruptions for this topology is higher than for the single tree overlay in Figure 2. This is related to the fact that the forest overlay tends to minimise the number leaves in order to maximise the use of available resources. Therefore, the majority of peers are internal. Due to the disjointness property of the forest overlay, they are internal in only one tree of the forest.

Due to the proactive nature of our solution, the total number of disruptions in CHAMS is lower than in THICKET. However, the advantage of CHAMS in the forest topology is less pronounced than in the tree overlay. Again, this is due to the property of the forest topology, which forces the majority of peers to be internal in one of the forest trees. Therefore even unreliable peers are included as internal nodes in one tree. Although this optimises the use of available resources [7], it increases the impact of churn.

Table 1: Tree overlay coverage (%)

Trace	PPLive		Bittorrent	
c	Plumtree	CHAMS	Plumtree	CHAMS
6	88.77	84.89	99.5	99.25
10	87.79	85.21	99.55	99.34
14	88.52	84.42	99.56	99.40
18	86.37	84.43	99.55	99.42

Table 2: Forest overlay coverage (%)

Trace	PPLive			Bittorrent	
c	Thicket	CHAMS	c	Thicket	CHAMS
20	81.13	77.925	16	98.165	98.12
25	82.65	79.305	20	97.745	96.905
30	80.565	75.92	24	94.855	97.345
35	81.18	77.47	28	96.735	97.83

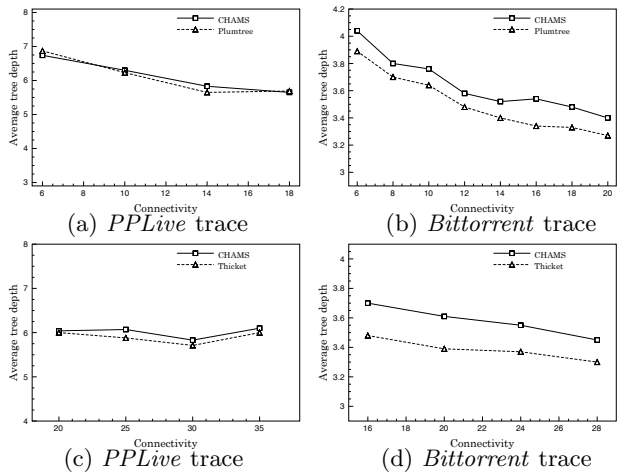
6.3 Overlay coverage

At any time t, the constructed overlay topology may not cover all peers. This is caused by churn that temporarily disconnects peers. To reconnect the overlay, our healing mechanism is performed as described in Section 5. To show its effectiveness, we discuss the overlay *coverage*. We measure it as the percentage of peers that are included in the overlay at a given time. Tables 1 and 2 show the average coverage of the tree and the forest overlays, respectively. Note that the presented average also includes the coverage at the start of the simulation when the first gossip round did not yet reach all peers.

As shown in these tables, the percentage of covered peers using both CHAMS and the comparison protocols is very high. This confirms the efficiency of the healing mechanism in CHAMS. The coverage of the CHAMS overlay is marginally lower than in the comparison protocols.

6.4 Path Latency

As argued in [17, 10], avoiding long paths in the overlay topology is desirable for minimising latency. For this, we investigate the depth of CHAMS overlays in comparison with PLUMTREE and THICKET. In CHAMS, the path reliability is calculated iteratively at each internal peer by simply adding join times. Thus, a long path has its reliability metric PR increased proportionally to the number of peers. Unless individual peer join times are significantly different, a long path tends to be less reliable. As a result, CHAMS prefers shorter paths.



(a) *PPLive* trace (b) *Bittorrent* trace

(c) *PPLive* trace (d) *Bittorrent* trace

Figure 4: Average overlay depth in terms of number of peer hops relaying messages from the source to all peers.

Figures 4(a) and (b) show the average depth of the CHAMS and PLUMTREE tree overlays, respectively; Figures 4(c) and (d) show the average depth for a two-tree forest overlay. At each peer, the depth represents the number of peer hops relaying a message from the source to that peer. The average depths in the CHAMS overlays are close to the ones in the compared protocols, although, as expected, CHAMS has marginally longer paths. Overall, while improving reliability, CHAMS does not compromise latency.

7. CONCLUSION

In this paper, we presented CHAMS, a scalable system for constructing reliable tree-like overlays that support application-level multicast for P2P streaming. CHAMS minimises the impact of churn by taken the churn risk of individual peers into account while constructing an implicit topology using a hybrid approach. Our experimental evaluation shows that CHAMS can construct churn-aware overlays while incurring a low latency penalty when compared to other protocols. In future work, we plan to extend CHAMS to construct overlays optimising bandwidth and latency in addition to reliability. In addition, we want to explore a real-world deployment on the public Internet as part of a P2P streaming system.

8. REFERENCES

[1] M. Allani, B. Garbinato, and F. Pedone. Application layer multicast. In B. Garbinato, H. Miranda, and L. Rodrigues, editors, *Middleware for Network Eccentric and Mobile Applications*, chapter 9. 2009.

[2] M. Allani, B. Garbinato, F. Pedone, and M. Stamenkovic. A gambling approach to scalable resource-aware streaming. In *Proceedings of SRDS*, 2007.

[3] M. Allani, J. Leitao, B. Garbinato, and L. Rodrigues. Rasm: Reliable algorithm for scalable multicast. In *Proceedings of PDP*, 2010.

[4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *SIGCOMM*, 2002.

[5] 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.

[6] N. Carvalho, J. Pereira, R. Oliveira, and L. Rodrigues. Emergent structure in unstructured epidemic multicast. In *Proceedings of DSN*, Edinburgh, UK, June 2007.

[7] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *Proceedings of SOSP*, pages 298–313, 2003.

[8] Y. Chawathe, S. McCanne, and E. A. Brewer. RMX: Reliable multicast for heterogeneous networks. In *INFOCOM*, pages 795–804, 2000.

[9] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM Sigmetrics*, 2000.

[10] M. Ferreira, J. Leitão, and L. Rodrigues. Thicket: A protocol for building and maintaining multiple trees in a p2p overlay. In *Proceedings of SRDS*, 2010.

[11] D. Frey, R. Guerraoui, A.-M. Kermarrec, M. Monod, K. Boris, M. Martin, and V. Quéema. Heterogeneous gossip. In *Proceedings of Middleware*, 2009.

[12] B. Garbinato, F. Pedone, and R. Schmidt. An adaptive algorithm for efficient message diffusion in unreliable environments. In *Proceedings of IEEE DSN*, 2004.

[13] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. J. W. O'Toole. Overcast: reliable multicasting with on overlay network. In *OSDI*, 2000.

[14] M. Jelasity and O. Babaoglu. T-Man: Gossip-based overlay topology management. In *Proceedings of ESOA*, 2005.

[15] A. M. Kermarrec, L. Massoulié;, and A. J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Trans. Parallel Distrib. Syst.*, 2003.

[16] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proceedings of SOSP*, 2003.

[17] J. Leitao, J. Pereira, and L. Rodrigues. Epidemic broadcast trees. In *Proceedings of SRDS*, 2007.

[18] H. C. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, and M. Dahlin. Flightpath: Obedience vs choice in cooperative services. In *OSDI*, 2008.

[19] F. Liu, X. Lu, Y. Peng, and J. Huang. An efficient distributed algorithm for constructing delay and degree-bounded application-level multicast tree. In *Proceedings of ISPAN*, Washington, DC, USA, 2005.

[20] A. Malekpour, F. Pedone, M. Allani, and B. Garbinato. Streamline: An architecture for overlay multicast. In *Proceedings of NCA*, 2009.

[21] L. Massoulié, A.-M. Kermarrec, and A. J. Ganesh. Network awareness and failure resilience in self-organising overlay networks. In *SRDS*, 2003.

[22] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *NGC '01*, pages 14–29, 2001.

[23] K. Sripanidkulchai, A. Ganjam, B. M. Maggs, and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *SIGCOMM*, 2004.

[24] G. Tan, S. A. Jarvis, and D. P. Spooner. Improving the fault resilience of overlay multicast for media streaming. In *DSN*, pages 558–567, 2006.

[25] C. Tang and C. Ward. GoCast: Gossip-enhanced overlay multicast for fast and dependable group communication. In *Proceedings of DSN*, 2005.

[26] Y. Tian, H. Shen, and K.-W. Ng. Improving reliability for application-layer multicast overlays. *IEEE Trans. Parallel Distrib. Syst.*, pages 1103–1116, 2010.

[27] E. Veloso, V. A. F. Almeida, W. M. Jr., A. Bestavros, and S. Jin. A hierarchical characterization of a live streaming media workload. *IEEE/ACM Trans. Netw.*, 2006.

[28] M. Yang and Z. Fei. A proactive approach to reconstructing overlay multicast trees. In *INFOCOM*, 2004.

[29] B. Zhang, A. Iosup, J. Pouwelse, and D. Epema. The peer-to-peer trace archive: design and comparative trace analysis. In *Proceedings of the CoNEXT Student Workshop*, 2010.