Peer-to-Peer Networking and Applications (2020) 13:1967–1989
https://doi.org/10.1007/s12083-020-00919-0

An integrated P2P framework for E-learning

Nikita Bhagatkar[1] · **Kapil Dolas**[2] · **R. K. Ghosh**[3] · **Sajal K. Das**[4]

Received: 2 August 2019 / Accepted: 16 April 2020 / Published online: 29 June 2020

Abstract

The focus of this paper is to design and develop a Peer-to-Peer Presentation System (P2P-PS) that supports E-learning through live media streaming coupled with a P2P shared whiteboard. The participants use the "ask doubt" feature to raise and resolve doubts during a session of ongoing presentation. The proposed P2P-PS system preserves causality between ask doubt and its resolution while disseminating them to all the participants. A buffered approach is employed to enhance the performance of P2P shared whiteboard, which may be used either in tandem with live media streaming or in standalone mode. The proposed system further extends P2P interactions on stored contents (files) built on top of a P2P file sharing and searching module with additional features. The added features allow the creation of mash-up presentations with annotations, posts, comments on audio, video, and PDF files as well as a discussion forum. We have implemented the P2P file sharing and searching system on the de Bruijn graph-based overlay for low latency. Extensive experiments were carried out on Emulab to validate the P2P-PS system using 200 physical nodes.

Keywords P2P · Live streaming · Emulab · Whiteboard · Mesh architecture

1 Introduction

The effectiveness of an E-Learning system can be judged by the form and opportunities for interactions it offers to the peers for the assimilation of both live and stored contents. The goal of this paper is to build a platform that facilitates meaningful peer-to-peer (P2P) collaborations between the presenter and the audience. The proposed approach subsumes all forms of interactive discussions, including live classroom, conferencing, and peer-group discussions.

As illustrated in Fig. 1, the existing collaborative E-learning systems are based on three types of cloud-supported platforms, namely:

1. Collaborative creation of forms, reports or documents (e.g., Overleaf [7] and Google Drive [6]).
2. Collaborative development of large computer programs (e.g., Github [5]).
3. Peer-to-peer tutoring/reciprocal learning by teaching (e.g., Duolingo [53], Coursera [2], Kahoot [3], and Brainly [1], Zoom [13, 15]).

Social networking inspired the first two categories of E-learning systems. These systems combine online access through web-based interfaces that can mimic a live presentation or classroom lectures with a Facebook like secure interaction environment between the presenter and the audience. The third category of systems focus on the delivery of recorded video contents to learners with features like tracking progress of the learners. They include online tests, quizzes, assignments setting up a two-way interaction between the learners and the teachers and also connect with Learning Management System (LMS) like Canvas [18] or Moodle [25]. However, Brainly [1]

✉ Nikita Bhagatkar
bhagatkar.nikita@gmail.com

Kapil Dolas
kapileyes@gmail.com

R. K. Ghosh
rkg@iitbhilai.ac.in

Sajal K. Das
sdas@mst.edu

[1] Cisco, Bengaluru, Karnataka, India

[2] Google, Bengaluru, Karnataka, India

[3] Department of Electrical Engineering and Computer Science, IIT Bhilai, Sejbahar, Chhattisgarh, India

[4] Department of Computer Science, Missouri University of Science & Technology, Rolla, MO, USA

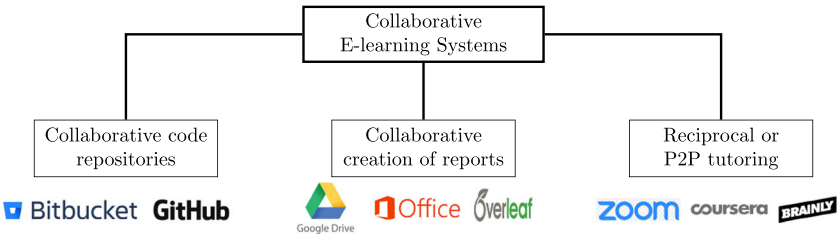


Fig. 1 Collaborative E-learning systems

follows a slightly different approach. It is predominantly a peer-to-peer learning tool that harnesses the crowdsourcing of like-minded students (peers) to combine problem-solving skills. It uses deep learning techniques based on historical data to predict particular requirements of a learner. Thus, a majority of social network based E-learning platforms provide only passive discussion forums with variations like incorporating peer mentoring and progress tracking.

Man-to-many interaction is integral to active learning as it happens in a live classroom or a physical meeting of a group of participants. A video conferencing system such as Skype [11] or Hangout [26] allows one-to-many screen sharing but not many-to-many sharing. However, the primary purpose of such a system is communication. So, the issues like scaling and many-to-many interaction are not addressed in a video conferencing system. For example, a group call in Skype is restricted to only 25.

Zoom [13] and WebEx [8], on the other hand, are developed as a cloud supported video meeting platforms with multiple screen sharing feature. WebEx is a product of Cisco Inc. Zoom can now be considered as the leader of modern enterprise video communication. These video conferencing systems scale up well in simultaneous many-to-many sharing. The participants can also use a whiteboard for co-annotations during video meetings. However, the video sharing feature deteriorates with the increase in the number of users. Both Zoom and WebEx being proprietary systems require subscriptions. Zoom's basic (free) version does not offer many of the sharing features and limited to 40 minutes meeting slots. WebEx does not have a free version. Recording of proceedings is allowed but with certain applicable limits on size. Furthermore, with increase in demand the basic users are forced to switch to audio conferencing mode. Neither Zoom nor WebEx allow media annotation. With touch screens document annotations are possible as inline images. As is the case with all proprietary systems, the architecture and internal details of implementations of both Zoom and WebEx are either too sketchy or not available. Therefore, we are unable to make a meaningful comparison.

The P2P-PS proposed in this paper is a light weight system that provides all the features Zoom or WebEx can offer. It does not require any server or cloud support for operation. Additionally, it allows one to create presentations and P2P discussions on the stored material. Co-annotations are possible during live video sessions. Annotations on store material are linked as meta data. One can annotate either media or documents. For Annotations on a media clip in a file, a user marks out both start and end time and puts text of the annotation. Such annotations are not possible either on Zoom or WebEx. Any user may comment on annotated material, setup a live video chat with other peers sharing a whiteboard alongside. The system can thus provide breakout sessions of groups for brain storming on a variety of stored content which aid reflective learning. It will, therefore, cater to the needs of a small group of learners and educators typical to a university campus. In current situation of pandemic outbreak of COVID-19, our P2P-PS may, in fact, be an ideal complementary system for offloading pressure on Zoom on WebEx. To organize the stored contents, we used an efficient implementation of Distributed Hash Table (DHT) based on de Bruijn graphs. A summary of features of the proposed P2P-PS system is as follows:

- It deploys a modified mesh-based architecture to leverage the spare capacities at the peers for flow control during the streaming of live media. The first peer who starts a streaming session is referred to as the 'speaker' while other peers joining later are 'listeners'.
- It allows a listener to initiate a query (e.g., seek clarifications) during live streaming using the "ask doubt" feature. The speaker alone may enable the dissemination of queries for maintaining the causality relation between a query and its corresponding explanation.
- All interacting peers (the speaker and listeners) may use P2P shared whiteboard using shapes, colors, and free-pen to illustrate points or seek clarifications. The whiteboard may be used in conjunction with live streaming.
- It incorporates an efficient DHT-based sharing and searching of media and document files, which is implemented using de Bruijn graph-based overlays.
- It allows tagging of stored material (both media and documents) for annotations, posts, comments, and announcements, which are stored separately to preserve the file mappings in DHT. Additionally, a gossip

protocol is used for fast synchronization of posts, comments, and announcements.
– It facilitates the creation of a mash-up presentation on the stored contents, which may optionally be accompanied by a P2P shared whiteboard.

We evaluated the proposed P2P-PS with the help of Emulab [49], a free testbed for emulation of network and distributed applications. We used 200 physical nodes on Emulab to carry out experiments. These physical nodes are container nodes on which we installed the P2P-PS software for emulation. We carried out separate experiments for streaming and stored contents. Experimental results not only establish the proof of concept but also indicate that interactive P2P-PS can become a useful E-Learning tool for remote classroom teaching as well as reflective learning through P2P interactions on the stored contents.

The rest of the paper is organized as follows. Section 2 gives an overview of the system architecture. Section 3 deals with live streaming using a modified mesh organization, while Section 4 focuses on the P2P shared whiteboard. Section 5 describes P2P-PS for stored contents. The design and implementation using de Bruijn graph-based DHT is discussed in Section 6. Section 8 presents annotations and discussion forum. It gives a comprehensive description on annotations of media and document files, posting of annotations, comments, and announcements. Emulation based experimental results are discussed in Section 9. Existing literature on P2P research that motivated our work is summarized in Section 10. Finally, Section 11 concludes the paper with directions for future research.

2 Overview of P2P presentation system

Figure 2 depicts a component level view of the P2P-PS system architecture. Live streaming uses a modified mesh architecture that exploits spare capacity at peers and adjusts both inflow and outflow rates for dynamic fanout at peers. The streaming may optionally be accompanied by an online P2P shared whiteboard, or a PDF file, or both. A shared whiteboard may also be used in standalone mode as a scratchpad to brainstorm ideas using shapes, colors, and free-pen through P2P interactions. A listener may interrupt the speaker during a presentation session and send a query (seek clarifications) as in a live physical lecture. The proposed system maintains the causality of a listener's query and the speaker's explanation by ensuring that the dissemination of queries occurs only after the speaker enables them. A stored streaming media may also create a replay of stored presentation which may be optionally accompanied by stored PDF (document) files, or the P2P shared whiteboard, or both. The following features facilitate the interactions on stored media contents: (i) Tagging parts of the media and PDF files, (ii) Annotations of tagged portions, (iii) Posting of annotated parts, (iv) Comments on annotated parts, (v) Announcements of the tagged parts, and (vi) Discussion forum.

We use a de Bruijn graph-based overlay for implementing distributed hash table (DHT) organization for stored

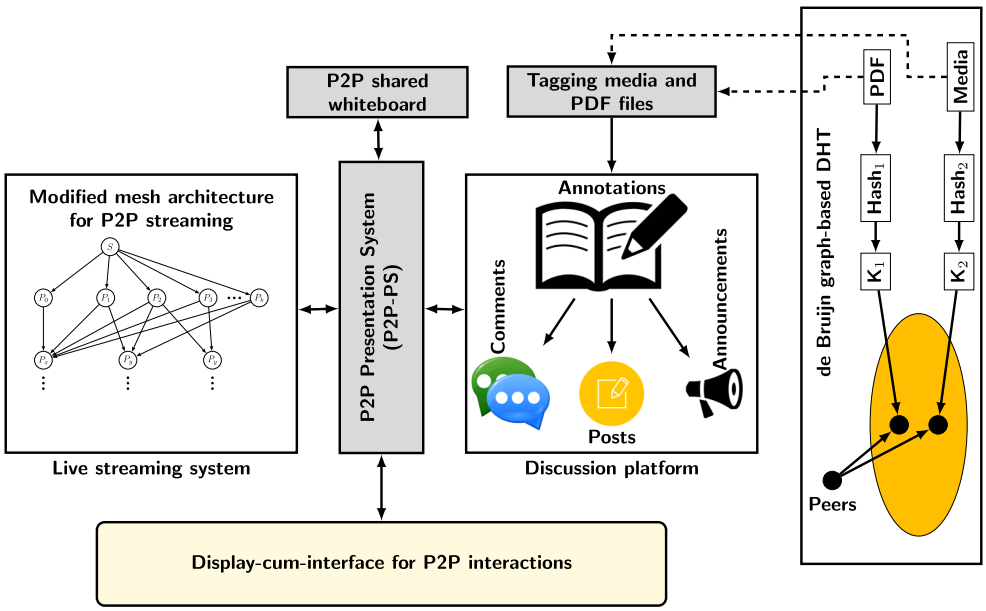


Fig. 2 Overall system architecture of P2P-PS

materials. There are two approaches for live media streaming on a P2P system [32]:

- Tree-based: It maintains a logical tree overlay where the root generates the stream, and other peers receive it from their respective parents. The leaves are free-riders.
- Mesh-based: A mesh architecture allows each peer to connect to other peers. It also lets a peer combine sub-streams received from more than one peer.

A tree-based media streaming approach is inadequate for our purpose. First, when an internal node leaves the entire subtree below it is orphaned and get partitioned from the network. Second, a tree cannot handle the dynamicity of peers joining and leaving the network. Therefore, we used a variation of mesh-based media streaming.

In the mesh-based P2P system, the nodes randomly connect, thus forming a mesh. These connections could be used to deliver data either unidirectionally or bidirectionally. For example, CoolStreaming [56] maintains bidirectional connections, whereas PRIME [34] maintains unidirectional connections.

3 Modified mesh approach

The mesh-based approach uses a swarm-type content delivery strategy [50] as in BitTorrent [35]. After receiving data from a server, a peer may act as a server for other peers. A peer collects data from other peers in parallel and combines it in a single file, thereby efficiently utilizing the bandwidth of its neighboring peers. It also reduces the load on the primary server because many peers now share the content distribution load.

The modified mesh-based approach exploits the availability of spare capacity at a peer. The upload bandwidth and streaming rate together determine the fanout value. The feeder node to a fanout cannot support a faster outflow rate than its inflow rate. It essentially forms a logical a gradient relationship between the in-degree and the out-degree of a node for receiving the data packets continuously. In the following description, the terms “parent” and “children” will respectively refer to a node’s neighbor in an inflow and an outflow path.

During a media streaming session, all the nodes except for the ones connected directly to the source must maintain the relationship, ‘in-degree $\leq$ out-degree.’ Since it does not preclude the relationship ’in-degree $>$ out-degree,’ a node may have a higher number of inflows than outflows implying a node has more parents than the number of children. However, we discard such a case because if a node receives more packets than it could send out, then eventually it will lead to a buffer overflow problem, leading to loss of multiple packets and re-transmissions.

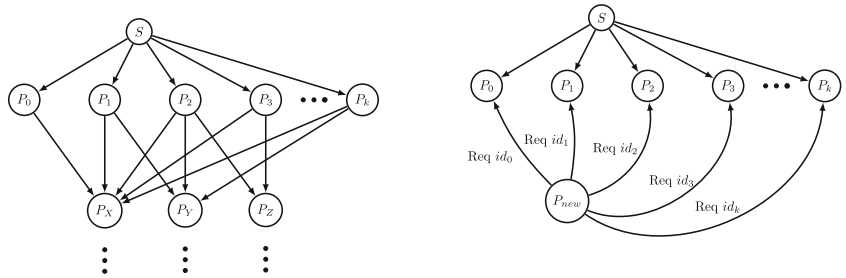
The media contents are divided into packet-sized chunks for streaming. The packets are propagated to the peers who assemble the received packets into a media file. In a live streaming the speaker is, typically, the source node which starts streaming of the media contents to its neighboring peers. The nodes other than the source are referred to as listeners. Being a multi-parent, multi-child architecture, a peer could ask for packets from its parent peers and deliver the the received packets to its children.

After authentication, the root (or the source) may start streaming data, but it does not send to any of its children (or listeners) unless the latter explicitly asked for the same. Figure 3a shows a partial view of a random mesh structure created during execution of the streaming process.

When a new peer P_{new} joins the system, it gets a list of nodes from a known bootstrap server as indicated in Fig. 3b. After attaching itself to the first parent, a node asks its parents for the latest (current) streaming packet id. P_{new} generates the initial requests for packets according to Algorithm 1. It essentially creates a gradient overlay network on top of a mesh network.

Algorithm 1 Generation of initial requests from a new peer on joining.

```
generateInitialPacketRequest()
    // BS is the bootstrap server
    get parentSet = {P_0, P_1, P_3, ..., P_{log n}} from BS
    for p ∈ parentSet do
        send "adopt-me" to p;
    end
    activeParents = NULL;
    set timeout to wait for "ack-to-adopt" from p ∈
    parentSet;
    while not timeout do
        if received "ack-to-adpot" then
            add {senderId} from "ack-to-adopt" to
            activeParents;
        end
    end
    // activeParents consist of all p ∈ parentSet from
    whom
    // "ack-to-adopt" were received
    set curr-id = latest packet-id from BS;
    for p ∈ activeParents do
        prepare self.req-id for p as in Fig. 3b;
        send self.req-id to p for initial packets;
    end
end
```



(a) Partial view of mesh. (b) Initial requests.

Fig. 3 A random mesh structure and initial requests from a new peer

Suppose P_{new} received its latest packet information from the parent P_0. P_{new} will then send requests for id_0 from P_0, id_1 from P_1, id_2 from P_2, and in general, id_k from P_k where $id_i = id_{i-1} + 1$ for $1 \leq i < k$. For example, let P_{new} get the initial reply from P_3, the next from P_1, and the next from P_k, then P_{new} requests for id_{k+1} from P_3, id_{k+2} from P_1, id_{k+3} from P_k, and so on. Figure 3b explains the strategy for subsequent pull requests for packets while the procedure is described by Algorithm 2.

Algorithm 2 Generation of subsequent requests by a peer.

```
generateNextPacketRequest()
    receive packet
    if seen[packet-id] == True then
        discard packet;
        continue;
    end
    else
        display(packet-data);
        seen[packet-id] = True
        extract parent-id, packet-id;
        for c ∈ childrenSet do
            if c.req-id == packet-id then
                Send packet to c;
            end
        end
    end
    while p ∈ activeParents do
        prepare self.req-id for p as in Fig. 4b;
        send self.req-id to p for the next packet;
    end
end
```

So, there is no need for a separate scheduler process. Each peer issues a request for the latest packet in a way akin to a reservation system.

3.1 Peer joining

Peer joining process has been explained in Algorithm 1 except for the authentication process. Initially, the root is responsible for generating the streaming content. It starts an active session on the mesh network. An ordinary peer or a listener joins by using the session key. The bootstrap server works also as the authentication server. A new peer, after authenticating itself submits the session key to join the streaming session. The bootstrap server adds the requesting peer to the active peer list after successful authentication (Fig. 4).

The bootstrap server maintains a list of n active peers having spare capacity to facilitate the peer joining process. As indicated by Algorithm 1, only a list of $\log_2 n$ out of the n active peers is provided to a new peer P_{new} to extend the gradient overlay graph. The reason for it is two-fold. Firstly, it ensures that the new peer can connect to the source by a low hop distance which improves the start up time for the new peer. Secondly, if any peer leaves the network alternative peers will be available to serve the new peer.

The peer P_{new} first determines which of the peers in the received parent-list could respond to the pull requests. For that, P_{new} sends "adopt-me" request to all the peers in the parent-list. A peer could accept or discard the request depending on its fanout value (out-degree) determined by the upload bandwidth (streaming rate). Initially, in-degree = out-degree for all nodes except for the one directly connected to the source node. Hence, if the number of parents (in-degree) of any node becomes less than its out-degree, then it requests the bootstrap server to send (in-degree − current_parent_count) number of active peers. The bootstrap server randomly selects a set of required number of active peers from the entire active peer list and returns the same to the requesting node.

3.2 Peer leaving

When a node quits gracefully, it proactively informs about its exit to the other children and the bootstrap server. Involuntary exit is a bit difficult to handle. There are two possible cases:

1. If the exiting node is not a source of inflow to any other peer in the network.

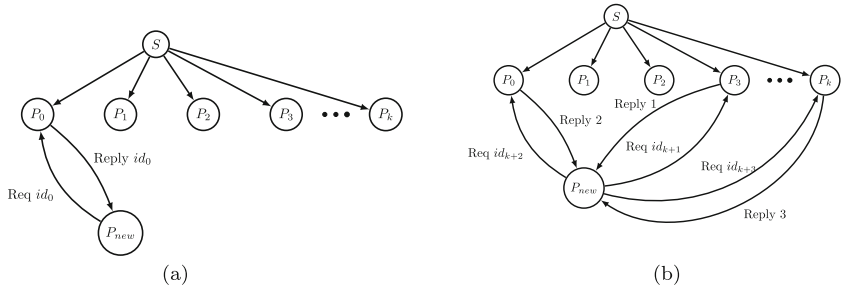


Fig. 4 Strategy for mesh-pull during streaming

2. If the the exiting peer is the only source of inflow to certain peers then the streaming at orphaned nodes stops.

In case 1, there can be no disruption. Though disruption may occur in case 2, an orphaned peer has to remember the packet-id which it requested in recent past. The peer just issue a new request for lost packet-id after realizing that the parent has left. Assume that it takes t units of time for transferring a packet between a sender and a receiver. If there is no response for "$2t$" units of time, then the peer compares the "Req-id" with the id of the last packet it has received. If the requested id is less than the last packet id it has received, then the requester assumes that the parent node is either dead or does not have the packet. So, it creates another "Req-id" for the same packet to a different peer. It means that non-availability of one packet may lead to a delay of at most $2t + t$=$3t$ time units.

If a peer P experiences a delay of timeout $> 2t$ units for a response from any of the parents $p(P)$, then P reports $p(P)$ to the bootstrap server BS. BS then initiates "are you alive?" message to $p(P)$. If there is no response for a time exceeding $2t$, then $p(P)$ is assumed to be dead. The bootstrap server removes the non-responding node from the list of active peers list and informs the child node P about the loss of its parent so that it could update its parent list. However, no parent list exists. A parent stores the list of its children. Hence, if a node exits involuntarily, then none of its children gets any information about the parent's exit. Every child node has to figure out on its own about the failure of its parent.

Suppose a node is left with only one parent who also fails abruptly. The orphaned peer must then request the bootstrap server to assign a new set of parents which may incur a delay. We used a scheme of proactive allocation of parents to reduce the delay for allocating parents to the orphaned nodes. Once a child notices that it has less than $k/2$ parents, it requests the bootstrap server to allocate at least one extra parent. Seeking one additional parent is to restrict the effect a "flash exit", which occurs when a talk (presentation) is about to end. Many peers start leaving the system at once. The number of parents for each existing peer falls below $k/2$ at a faster rate. Therefore, at that point, it might not be possible to satisfy the requirement of k parents. In the worst case, only a source node and a single peer may be available, and all other nodes may have departed. In this case, the source could be the only parent of the remaining peer.

3.3 Handling "ask doubt" feature

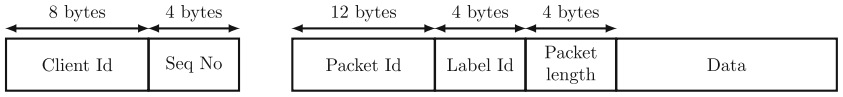
Every peer is aware of the source node's (the speaker's device) address. Whenever a listener wants to initiate a query, he/she clicks on the "ask doubt" button provided in the user interface of the application. It establishes a direct connection with the speaker's device. The speaker gets a notification of the query and may send an acknowledgment. The peer device is allowed to send the query after receiving the acknowledgment. The doubt or the query should be in the form of an audio message like it happens during a physical presentation. The message is unicast on the link between the source and the peer, which initiated the query. The child peers of the requester would then pull the data while the source pushes the data to its other children. Thus, the query is propagated in a push-based manner as it happens in the tree-based approach. It guarantees that the speaker can resolve no query before it is asked, thereby preserving the causality relationship.

4 P2P shared whiteboard

The shared whiteboard follows a push-based approach for data propagation as opposed to the pull-based approach followed by live streaming. There is no parent-child relationship between the nodes. All adjacent nodes are neighbors. The packet structure of the live board appears in Fig. 5. A shared whiteboard's packet consists of a packet-id, a label-id, the packet length, and the data. The packet-id is composed of the client-id and the sequence number.

The packet-id field uniquely identifies a packet. The client-id is the first 8B of the SHA-1 hash of the IP address of a user. The client-id ensures that a packet delivery does not occur to the same peer from whom it is received. The shared whiteboard supports multiple pages. Each page gets a unique label-id. The operations performed on every page are stored separately. The label-id makes the

Fig. 5 Whiteboard packet



(a) Packet ID. (b) Packet structure.

canvas repainting task easier on the receiver side. It also allows maintaining a consistent view of both the sender and the receiver. Due to a push-based approach, there may be data redundancy at the receiver side. So, the old packets are marked and then discarded. Propagating text data is not an intensive operation. Therefore, even with the push-based approach, it does not contribute much to the overhead.

4.1 Repainting shared whiteboard

The user interface of the whiteboard application provides many operations like basic shapes, colors, free-pen. Among these, free-pen is the most time-consuming operation. A single free-pen operation generates many events. Sending one event over the network consumes much bandwidth and degrades the system performance under the presence of moderate to heavy network traffic. We used a buffered approach to improve the performance of free-pen. The content of the entire buffer is sent if either the buffer is full or a mouse release event occurs. At the receiver side, the operations replayed on receipt. The events get repainted in a sequential manner only. The size of the buffer depends on the packet size or maximum transmission unit (MTU).

4.2 Consistency of board views at peers

Maintaining a consistent view of all peers is most important for the shared whiteboard application. The system maintains a separate list for every label-id. Whenever a new packet arrives, after extracting its label-id, the corresponding packet is placed in that label-id list at the appropriate position according to its sequence number. The algorithms for packet generation, sender and receiver processes are provided in Algorithms 3, 4, and 5 respectively. A push-based approach causes packet losses and network delays. It may lead to an inconsistent view of the board among the peers if the replay at a receiver happens without considering the delayed packets. For handling delayed packets, at first, each packet is placed at its correct relative position, among the other packets. Then the repainting is done using the label-id and the sequence number in the packet. It is not a compute-intensive operation. The repainting is also very quick, as most computers can perform more than 10^8 operations per sec. Furthermore, the repainting performed only for one single page of the board at a time.

Algorithm 3 Packet generation algorithm.

```
packetGeneration()
    while BufferNotFull OR
    MouseReleaseEventNotHappened do
        Create data packet; SendQueue.put(packet);
    end
end
```

Algorithm 4 Sender's algorithm.

```
sendPacket()
    packet = SendQueue.get();
    for nodes in neighbor do
        Send to all except from whom it is received;
    end
end
```

Algorithm 5 Receiver's algorithm.

```
recvPacket()
    receive data packet;
    extract packet-id;
    if seen[packet-id]==True then
        continue;
    end
    else
        seen[packet-id]=True;
        SendQueue.put(packet);
        extract client-id, sequence no., label-id, packe
        length and data;
        if Label[label-id].list.last-sequence no. ≤
        sequence_no then
            Label[label-id].list.append(sequence_no.)
            last-sequence_no = sequence_no;
            draw(data);
        end
        else
            insert (sequence no., data) at appropriate
            position in Label[label-id].list;
            repaint page with label-id from that
            sequence_number;
        end
    end
end
```

A peer joining late may get an inconsistent view of the shared whiteboard. We used a combination of push-pull to

solve this problem. The current label-id does not match with the peer's label-id, then a comparison is performed with the sequence numbers. If the current label-id or the sequence number is a positive number, then it is safe to assume that data propagation has already started. Hence, a peer joining late sends requests (pulls) for the previous data from one of the neighbors. Each operation belonging to a page is tagged with the corresponding page number. A page may as well be left blank intentionally. In other words, the number of operations performed on a page may be zero or a positive number. In a pull request, a peer first requests for the data of the page in the current view, which can be extracted with the help of label-id. Therefore, a pull request must include label-id. The responder replies with the number of operations performed on the page. The receiver then verifies the reply. If the number of operations performed on that page is zero, no further request is needed. Otherwise, a request for remaining operations would be made. These operations are requested one at a time in a pull-based manner until the latest sequence number. After the late joiner receives all the packets, only then his/her session gets activated.

An explicit deferred joining process is necessary to avoid the inconsistent intervention of a late joiner. To understand why it happens, consider a user who joins after half an hour of the start of a session. Suppose the user immediately starts some operations on the first page of the canvas when the user's device is still in the process of receiving data from other peers. Such an intervention by the user leads to an overwriting of the previous data. Therefore, we defer the activation time of the late joining peer until the data synchronization is complete. Another solution could be to enable the group-undo feature in the system, i.e., every undo command gets propagated to the entire group, and required changes will be done on every peer's canvas. Even if the user joins the system before the sync is complete and starts scribbling, we could use a group-undo command to undo those operations. The inclusion of the group-undo feature is not available in the current work.

A screenshot of the video streaming along with illustration on shared white-board is given below in Fig. 6. The presenter's video appears to right bottom corner when she was explaining the Pythagoras theorem.

The screenshot in Fig. 7 depicts the user's control panel for video streaming combined with shared whiteboard.

Before starting a new session the user generates and verifies a new session key to indicate a new presentation session. The presenter can either uses a stored contents (video or audio) or can start a new streaming session. By sharing the screen a user (the presenter) can initiate a multi-way presentation with a bunch of listeners with video and shared whiteboard.

5 P2P-PS for stored contents

P2P interaction on stored materials is another important aspect of our system. It relies on an efficient, robust file sharing and searching component that incorporates many additional features. The most important among these features is tagging selected parts of audio, video and PDF files. A participating peer can raise queries by creating annotations and postings of the tagged portions of media and document files. A peer may also give comments on posts for resolution of queries. A stored media may be streamed tightly coupled with a shared whiteboard for live discussion much like a live meeting, or a brainstorming session. Therefore, the P2P-PS on stored contents is a concomitant system of overall P2P-PS platform. It helps in reflective learning and even creating a mash-up presentation using stored repositories. The file sharing system is implemented using de Bruijn graph overlays.

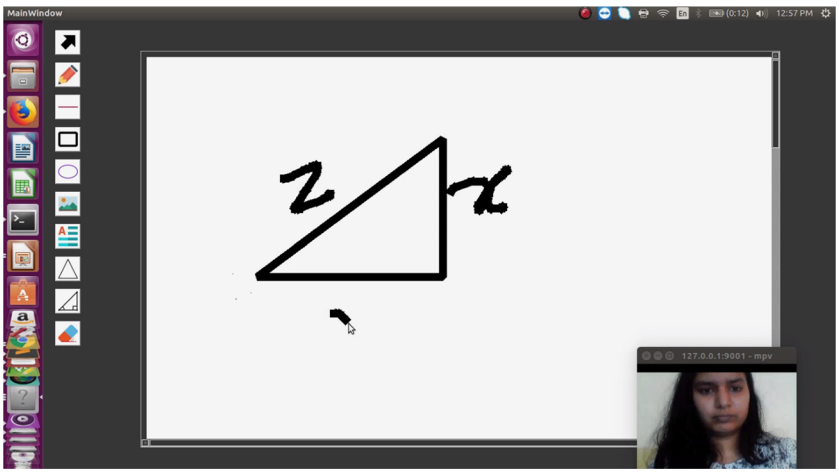


Fig. 6 Screenshot of the presenter explaining with aid of shared whiteboard

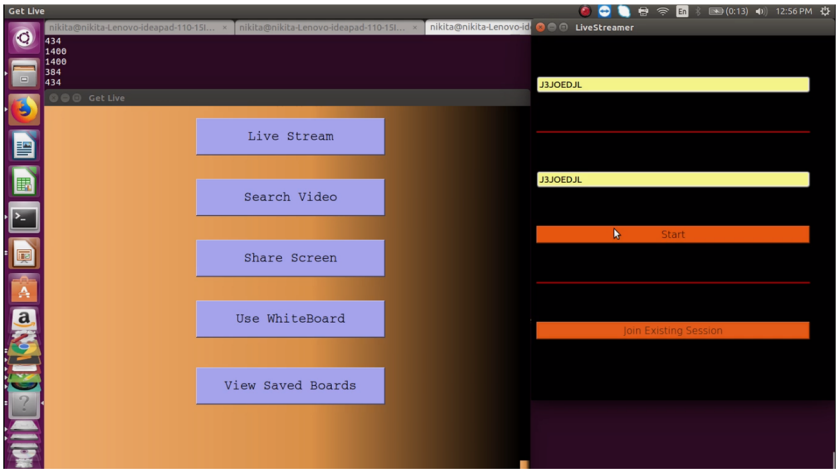


Fig. 7 A screenshot of control panel with session key verification

5.1 Choice of de Bruijn graphs for DHT implentation

The stored contents are organized into a DHT using de Bruijn graph-based overlays. A de Bruijn graph is a labeled directed multigraph with fixed out-degree K. Every node of the graph has an ID or a label of fixed length. Let the length of each ID be D and alphabet Σ size be K. De Bruijn graph $B(K, D)$ of $N = K^D$ nodes can be constructed as follows. Each node is connected by an outgoing edge to K other nodes. There is an outgoing edge from a node A to a node B if ID_B can be created by applying a left shift to the label ID_A and appending one symbol from alphabet set Σ to the rightmost position, i.e., $ID_B = ID_A[1:] + a$ where $a \in \Sigma$. If labels are considered as base K numbers, then outgoing edges will be to all the nodes with labels equal to $(ID_A * K + k)(\mod N)$ where $0 \leq k \leq K-1$. Figure 8 illustrates an example of a de Bruijn graph $B(2, 3)$.

Routing in a de Bruijn graph is specified as a string. Let S and D denote the source and destination nodes with labels ID_S and ID_D respectively. Then the string for the routing path from S to D is obtained as follows: (1) Find the maximum overlap between the suffix of ID_S and prefix of ID_D; (2) Remove the overlap part from prefix of ID_D; and (3) Append the remaining suffix part of ID_D to ID_S.

For example, assume that a look-up for destination node 1011 is initiated by a source node with 1110. The maximum overlap of the suffix of 1110 and prefix of 1011 is 10 as shown in Fig 9. Hence, the string for routing path will be 111011, i.e., 1110 appended by the non-overlapped part 11 of id of destination node.

The next hop v is derived from a current hop u as follows:

1. v is a neighbor of u in the input de Bruijn graph.
2. The length of longest suffix of v that is same as a prefix of ID_D which has a length 1 more than the length of the longest suffix of u that is equal to a prefix of ID_D.

This routing scheme is known as substring routing.

The structure and routing method indicate that the diameter of de Bruijn graph is $D = \log_K N$. It has low clustering and exhibit $(K-1)$-node-connectivity. K-node-connectivity is not possible due to self-loops on K nodes with IDs of the form "$\alpha\alpha\ldots\alpha$", for $\alpha = 0$ to $K-1$. The nodes can be linked together to form a ring which makes the graph K-regular and also achieves K-node-connectivity. K-node-connectivity makes the graph more resilient to fault-tolerance. Even the failure of any $(K-1)$ nodes cannot disconnect the graph and diameter remains at most $D+1$. Loguinov et al [33] showed that the expected congestion in de Bruijn graph is much less than the other counterparts under a similar rate of load, due to larger bisection width of the graph.

The de Bruijn graph possesses better asymptotic degree-diameter properties compared to some of the widely used DHTs, such as Chord [48], Trie [19], CAN [45], Pastry [46]

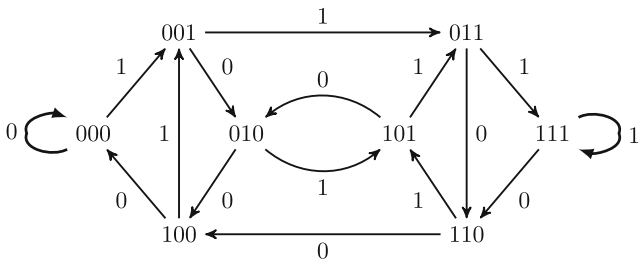


Fig. 8 An example of de Bruijn graph $B(2, 3)$

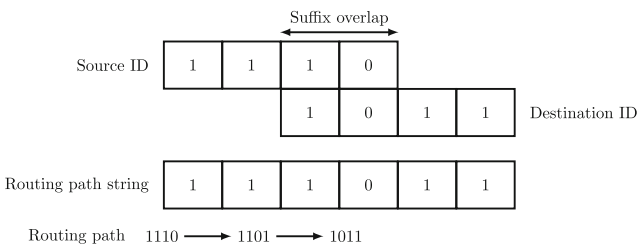


Fig. 9 An example of routing in de Bruijn graph

Table 1 Asymptotic degree-diameter properties

Graph	Degree	Diameter D
de Bruijn	K	$\log_K N$
Trie	$K+1$	$2\log_K N$
Chord	$\log_2 N$	$\log_2 N$
CAN	$2d$	$d/2N^{1/d}$
Pastry	$(b-1)\log_b N$	$\log_b N$
Butterfly	K	$2\log_K N(1-o(1))$

and Butterfly [37]. Tables 1 and 2 provide a summary of comparisons in terms of degree and diameters from the analysis made by Loguinov et al. [33]. The blank cells in Table 2 indicate that corresponding node degrees are not supported.

Table 3 compares the average distance between the nodes in de Bruijn graph to the optimal Moore graph [33] with the same degree K. In a de Bruijn graph it remains very close to optimal values even for the smaller values of K.

6 Overlay network based on de Bruijn graphs

Loguinov et al. [33] proposed some guidelines for incremental construction of a de Bruijn graph. However, the paper falls short of an actual implementation as it does not address the problem of maintaining de Bruijn structure in presence of churning, where churn refers to dynamicity involving nodes leaving and joining.

For the purpose of implementation, we choose the parameters $K = 8$, and $D = 8$ for a de Bruijn graph. It allows around 16 million nodes inside network labeled "00000000-77777777" in octal strings. However, the diameter of a de Bruijn graph remains O(1) (8 to be precise). The DHT overlay in our system would support efficient look-ups, and an epidemic dissemination-based protocols can be designed to infect all the nodes in just a few rounds.

Table 2 Graph diameter for $N = 10^6$ nodes

k	de Bruijn	Trie	Chord	CAN	Pastry	Butterfly
2	20	–	–	huge	–	31
3	13	40	–	–	–	20
4	10	26	–	1,000	–	16
10	6	13	–	40	–	10
20	5	10	20	20	20	8
50	4	8	–	–	7	7
100	3	6	–	–	5	5

Table 3 Average distance between pair of nodes for $N = 10^6$

K	Moore graph	de Bruijn
2	17.9	18.3
3	11.7	11.9
4	9.4	9.5
10	5.8	5.9
20	4.5	4.6
50	3.5	3.5
100	2.98	2.98

6.1 Node information structure

We refer to the nodes in the underlying de Bruijn graph as "virtual" nodes. For maintaining the underlying graph, a physical node in our system is responsible for a range of *virtual* nodes with consecutive IDs. A range of a physical node in our system is referred to as its zone. Initially, when a single physical node joins the DHT, it becomes responsible for all virtual nodes from 00000000 to 77777777. The virtual ID-space may be visualized as a ring, where each physical node is responsible for only an arc segment of the ring. Figure 10 illustrates an example of three physical nodes responsible for three different zones.

The structure is similar to Chords [48]. However, unlike the Chord overlay a physical node responsible for an ID space arc may be located somewhere randomly within it. A node A has an outgoing edge to another node B, if there is at least one virtual node in A's zone having an outgoing edge to a virtual node in B's zone. Each node keeps a list of its outgoing and incoming edges. Each node in these lists knows its address and zone ID. The details of the structure maintained at each node is given in Table 4.

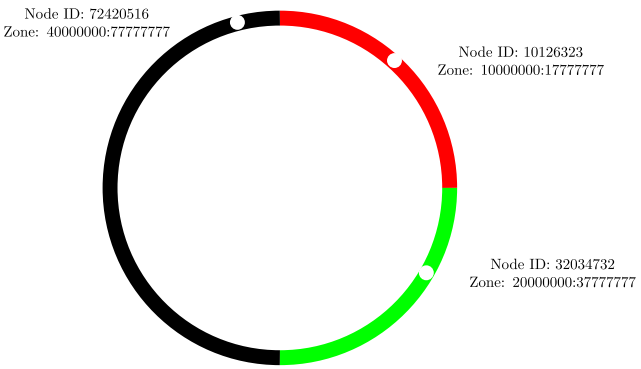


Fig. 10 Three physical nodes in a de Bruijn based ring with Zone ownership

Table 4 Structure maintained at each node

Item	Information
ID	ID of node is a label lying in the Zone
Zone	A range of virtual IDs for which node is responsible
External Address	Address on which others should contact
Outgoing Edges	List of nodes connected by outgoing edge, along with info about their Zone and Address
Incoming Edges	List of nodes connected by an incoming edge, along with info about their Zone and Address

6.2 Joining of nodes

If A tries to join, it selects a random ID and forwards a join request to identify the owner of the zone in which the chosen random ID falls. For convenience of description, we use the following convention: any intermediate node receiving the join request initiated by A is referred to as B; while the node whose zone contains the random ID chosen by A, is node C. The problem of joining is split into three parts according to actions of A, B and C is explained below.

Node A requests the rendezvous server (whose public IP address is known to all) to provide the external address (IP address) of A and those in a list of random peers. A picks one peer randomly from the list supplied by rendezvous server and sends a join request to that peer. The request contains A's external address, and a random virtual ID from the entire ID-space, i.e., 00000000 to 77777777.

The join request initiated by A tries to identify C, the owner of the random ID sent in the join request. When node A sends such a request for the first time, its ID is chosen using SHA-1 hash value of its external address. But, upon retries, a random ID from the entire region is picked.

On receiving the join request, B forwards it to the next node on the routing path. If a routing path is not provided in the request, B will create one using B's ID and destination ID given in the join request. The correct path will be sent along with the join request.

On receiving the join request from A (possibly through intermediate nodes), C sends the information regarding its zone ID, the incoming and the outgoing links. C does not accept further Join requests until joining of A is complete, or the timeout of 10 seconds occurs.

The reply from C contains a structure, namely the virtual node label of C, external address of C, and outgoing and incoming edges of C. Now C waits for A to complete the joining process and sends information about A's new zone. Then C updates its own zone by sending keys with values to be managed by A, notifying the neighbors about the change in zone, and dropping the edges destroyed due to the shrinking of its zone.

Next, A picks the half part of the zone not containing C's ID and chooses a random ID (label) from the picked zone as own ID. A sends information of its ID and its zone ID to C, and the attachment of node A in DHT overlay is complete. A needs to perform the following two actions to complete the joining process: (i) disseminate its ID and zone information to all the links shared by C; (ii) identify the links to be dropped, check if there is any new link to C, and make the corresponding changes to the list of incoming and outgoing edges.

Finally, A willingly accepts the load shared by C. Due to space limitation, the algorithm is included here. Interested readers can review the Send Join Request and Receive Join Request algorithms in Appendix A and Appendix B, respectively.

6.2.1 Join example

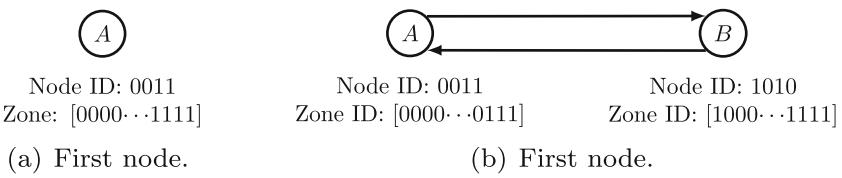
To keep the example small, we show the Node Join process in de Bruijn graph with $K = 2$ and $D = 4$. Figure 11a and b illustrate joining process of the first two nodes in the system.

The process of joining of the third node is shown in Fig. 12a. Notice that the joining of the third node requires the removal of the (dashed line) link $A \longrightarrow B$. For the joining of the fourth node two new links are to be inserted as shown by dotted lines in Fig. 12b.

6.3 Leaving of nodes

Consider the leaving of a node A from the system. A node C is identified who can merge the zone of A into its own Zone. There are two parts to the leave process, as the leaving node is aware of its both the predecessor and the successor in the DHT overlay.

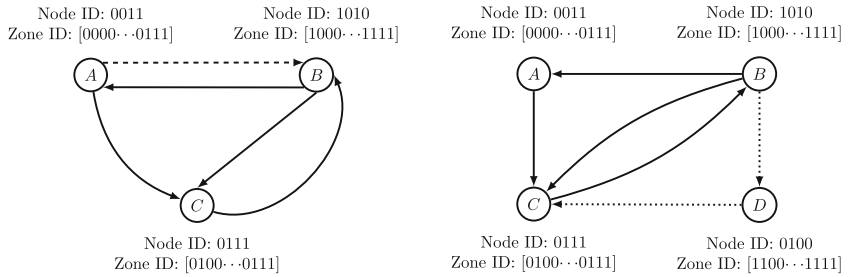
A identifies a successor zone and a predecessor zone. The first virtual ID belonging to the successor zone equal to



(a) First node. (b) First node.

Fig. 11 Joining of the first pair of nodes in the system

Fig. 12 Joining of the next pair of nodes



(a) $A \rightarrow B$ link gets removed. (b) $C \rightarrow D$ & $D \rightarrow B$ links get added.

A's Zone.endID + 1. Similarly, the last virtual ID belonging to the predecessor Zone is equal to A's Zone.startID -1. A picks one of the two IDs, finds the respective owners of the zones and sends a request to leave. The leave request contains the information regarding ID, Zone, incoming and outgoing links to the chosen neighbor, say C. If C agrees within timeout period of 5 seconds, A sends the load to C and the leave process is complete. Otherwise, A tries the same with the other neighboring node. If the timeout occurs again, the entire procedure is repeated assuming chosen nodes were busy in other leave procedure. Algorithm 8 in Appendix C specifies the precise steps executed by A.

Node C receiving the request checks if it is going to leave the system. If not, it accepts the request and agrees to take over the load by merging the two zones. The merging process includes merging of the incoming and outgoing links as well. C then notifies all the linked nodes regarding the update. Algorithm 9 in Appendix D gives a step-wise description of C's operations.

During join and leave, nodes whose zone changes notify the nodes linked to them by outgoing or incoming edges. When such notification is received, a node adds or drops links caused by the change. Every two minutes, each node sends keep-alive message to all linked nodes. These nodes update the timestamp with the zone information corresponding to that node. If no update is received in last five minutes for a linked node, it is considered dead, and owner of the successor zone takes the responsibility of orphaned zone.

If there is an outgoing edge from a virtual node in one zone to a virtual node in another zone, then there exists an outgoing edge between one node to the other. On many occasions, zones of the neighboring nodes change. A brute force way to find an edge in the underlying graph would slow down join and leave operations.

Node A receives zone update from another node B. A checks if the size of its zone is larger than or equal to N/K. If so, then a link exists (because every possible suffix of length $D-1$ is present in A's zone). Otherwise, it determines the range $[L_A, R_A]$ of the suffix of length $D-1$ lying in the A and range $[L_B, R_B]$ of the prefix of length $D-1$ lying in B's zone. If $L_A \in [L_B, R_B]$, or $R_A \in [L_B, R_B]$ then there exists an edge from node A to node B. Similarly, there exists an edge if L_B or R_B lies within $[L_A, R_A]$. Consider the following two examples to understand how range overlap is used to determine the connectivity.

- Consider two zones: node A's zone ['00000000', '17777777'] and node B's zone ['40000000', '77777777']. Since the size of zone of node A is larger than '10000000', it will have all possible suffixes of length $D-1$. Therefore, it will have link to any node in the system. Same is the case for B and hence, both have outgoing edges to each other.
- Consider two other zones: node A's zone ['00000000', '00777777'] and node B's zone ['01000000', '01777777']. The range of suffixes of length $D-1$, for node A is ['0000000', '0777777']. The range of prefixes of length $D-1$, for node B is ['0100000', '0177777']. The intersection between the two ranges is non-empty, and hence an edge exists from node A to node B. (In the underlying graph, '00100000' has an outgoing edge to '01000000'.)
- For the above example, range of suffixes of length $D-1$, for B is ['1000000', '1777777']. The range of prefixes of length $D-1$, for A is ['0000000', '0077777']. The intersection between two ranges is empty, and hence no edge can exist from B to A.

7 Sharing, searching, and global indexing

Our implementation supports two types of searches (i) by ID/key, and (ii) by keywords. A user can share one or more directories which contain the file(s). By default, at least one directory is shared which may be empty. The said directory is located in the "Downloads" folder of the user's system. Anything downloaded from the P2P system is stored in this directory and is automatically shared. We calculate 160-bit SHA-1 value for the contents of the file and use it as a key. Since, each node has 24-bit

label, the first 24 bits of a key are used for the routing purpose. For example, first 24 bits of the SHA-1 hash value "2fd4e1c67a2d28fced849ee1bb76e7391b93eb12" is "2fd4e1" which is equal to "13752341" in octal.

7.1 Pre-processing of files

At the start of the application and after every 30 minutes, the directories are scanned for any new or modified files. Both new and modified files are queued for processing. SHA1 of a file is computed, and it is stored in the local database. Any deleted or renamed file is also identified. For deleted files, the entries related to them are deleted from the local database. A set of important keywords are associated with file to improve the chances of finding a document or file in the DHT. We provide support for extracting text from PDF and video files that come along with the subtitles. If a PDF is created using scanned copies, the text is extracted using an OCR tool. Once the text has been extracted, TF-IDF scheme is used to identify top 100 keywords for a file using its content. In this scheme, the file is considered to be a set of documents of 1000 words each. In the TF-IDF scheme, a term is assigned a higher score if it has larger frequency than other terms, but appears in a less number of the documents. The scoring scheme is, therefore, based on Inverse Document Frequency. These keywords are stored in the local database along with file's modification times, which can be used to avoid processing of the file again. For all keywords, SHA-1 is also computed to obtain the keys. The users can also manually add up to 10 keywords.

7.2 Global indexing of files

A Key is a globally unique ID of an object (e.g., SHA-1 value), and the Value contains the name of the object, the size of the object, the address of the object and the timestamp of the last refresh. The keywords are stored as meta data and maintained in the local database at the external nodes of an overlay. The purpose is to enable keyword based search for related document and media files. We need two basic operations, namely, Put and Get.

For each file, MultiKeyPut is used to insert key-value pairs of all keywords in the system. The keywords also include words in the file name. The associated value stores information such as SHA-1 hash of file, size of the file, name of the file and address of the file. After every 30 minutes, MultiKeyPut is called for keywords of each file, thus refreshing the timestamp for keys in corresponding nodes. A node will check for the key's timestamp every 10 minutes, if a particular node has not updated a key in last 60 minutes, corresponding key-value pair is considered to be not available and dropped from the local store. The delay of 60 minutes is kept considering a lost update in the network.

7.3 File searching

Searching operation is performed one at a time. A user can cancel the search at anytime. Since, there can be delayed replies, a unique search ID is associated with the Get requests to distinguish the results. Any node returning result/s must provide search ID along with the result. The packets received for the current search ID are kept, and all other are discarded.

A query is broken down into keywords, and search is conducted with MultiKeyGet procedure. For any reply that comes while search is active, the ranking of the results may change. The file having more number of query keywords in it, gets a higher rank. The files with same number of query keywords is distinguished by the replication/popularity in the system. The more popular result gets a higher ranking than the others.

A user might receive the ID or the key of a file from another user via some communication channel. If user wishes to download file with given ID, he/she can enter it in Get File dialog which calls Get method. On receiving the results, a download request is initiated.

7.3.1 File download

When results are fetched using the above search methods, a user can choose to download a file. If chosen file is available with multiple peers, multi-threaded approach for downloading is used. Different chunks are requested from multiple peers, and they are written to the file as they are downloaded.

8 Annotations and discussion forum

Most PDF viewers have tools to highlight a portion of the text. But, any editing of the content changes hash value of a file. Modification of hash value of a file is undesirable, as the file in P2P learning environment is archival in nature. Therefore, the features should be provided without editing the file itself. Furthermore, the users may also want to see portions that are highlighted by others. Some other users may wish to tag some text or provide a hyperlink to some other external resource. All these operations are grouped as peer learning activities without any change in the contents. We, therefore, defined a general format for annotations. The general annotation format is a template for other formats. The following information is stored in an annotation:

- Annotation ID: A unique ID to identify an annotation.
- File ID: A unique ID for file. SHA-1 value is used.
- Time stamp: Time of creation of the annotation.
- Author: Identity of the creator.

– Text: Associated text, if any.
– Properties: Specific properties of the annotation.

The annotations are stored separately in the local database, and synchronized among peers. The "Properties" field stores the properties for specific types of annotations. The text field may be in the form of a hypertext, as the support for links and images is provided.

A user can create and use annotations either for personal use or can share them with others to spread useful information. A shared annotation is a post that can be discussed among the participating peers. So, we integrated a discussion forum along with dissemination of annotations.

8.1 PDF annotations

Highlighting in the PDF documents can be done by either selecting text or selecting a rectangle. We define properties of each type separately. Text selection provides the flexibility of selection of a specific word or a set of words, or a set of lines in a particular page. If selection spans multiple pages, the user should select words or lines on each page separately. The user may also provide a comment or add some text related to a highlighted area. The additional text/comments are stored with the annotation. Since each annotation of this type is designed for only one continuous region of a text, we define following properties for an annotation.

– PageNumber: Denotes the page number on which annotation is present.
– FirstWord: Denotes the index of the first selected word
– LastWord: Denotes the index of the last selected word

Rectangular region selections are generally helpful in inserting annotation related to diagrams, or non-textual information. A user can choose one rectangular region of a page and highlight the portion. Following properties define each rectangular region:

– PageNumber: Denotes a page number on which the annotation is present.
– TopLeft: Denotes co-ordinates of the top-left corner of the selected rectangle with respect to the the top-left corner of a page
– BottomRight: Denotes the co-ordinates of the bottom-right corner of a selected rectangle with respect to the top-left corner of a page.

8.2 Media annotations

Video annotations being similar to Audio annotations, we do not distinguish between the two. A user can select a duration of the video and tag it. The two end-points of a duration are rounded to the nearest integers and at least 5 seconds of the duration is selected. The properties for such a selection are: StartTime (the time at the start of duration) and EndTime (the time at the end of duration).

8.3 Posts, comments and announcements

A post inherits the format of the corresponding annotation, adds an additional field for Title. A post also has a constraint of minimum (resp. maximum) number of 100 (resp. 1600) characters in the Text field.

– ID of Post: A unique ID to identify a post
– File ID: A unique ID for file. SHA-1 value is used
– Time stamp: The time of creation of annotation
– Author: The identity of the creator
– Text: Associated text
– Title: A short description of the post
– Properties: Specific properties of PDF, Audio/Video annotations are stored.

When an annotation is converted into a post, Text field of annotation is copied into the Text field of the post. The properties field of post supports multiple annotations. Multiple annotations are stored in a list, serialized into a string and then stored in Properties field of the corresponding post. If multiple annotations are to be attached to a post, File ID is set to zero.

The comments on a post follow similar format as the post. But, instead of Title field, it has ReplyTo field.

– Comment ID : A unique ID to identify a comment
– File ID: A unique ID for file, SHA-1 value is used
– Time stamp: The time of creation of the annotation
– Author: The identity of the creator
– Text: Associated text
– ReplyTo: ID of post or comment to which this is a reply
– Properties: Specific properties of the PDF or Audio/Video annotations are stored.

Announcements are similar to posts in the discussion forum. They are useful for posting information about locations of newly added documents. The sharing of announcements is carried out in a similar fashion. They are forwarded to other peers as soon as they are received. On the front-end, announcements appear separately from other posts.

8.4 Synchronization of posts and comments

Posts and comments are synchronized using following two protocols: (i) an epidemic dissemination based protocol to spread recently published posts and comments; and (ii) a reconciliation (or anti-entropy session) performed at regular

intervals with the neighbors to handle missed updates, since a user will not be active at all times.

The consistency requirements of the discussion platform is weaker because once a post is made, it cannot be updated. For a particular post or a comment, only one person is responsible. Posts and comments need not be delivered immediately. The happened-before relation of the comments and the posts is maintained due to their hierarchical structure. The causal ordering of other messages can only be provided through time-stamps associated with them. We assume that clocks of the machines are synchronized with NTP servers.

8.4.1 Epidemic dissemination

Every node receives posts and comments through their neighbors. Our application periodically checks if any new post or comment arrived since the last check. If there are new posts or comments, they are sent to all the neighbors except the ones from which they were received. This ensures the delivery of messages to all. The period of such check is set to one minute. The diameter of our system is $D = 8$, therefore, any new message gets delivered to the entire system in less than ten minutes. The interval time can be set to zero, and in that case, messages will be delivered to others instantly.

8.4.2 Reconciliation

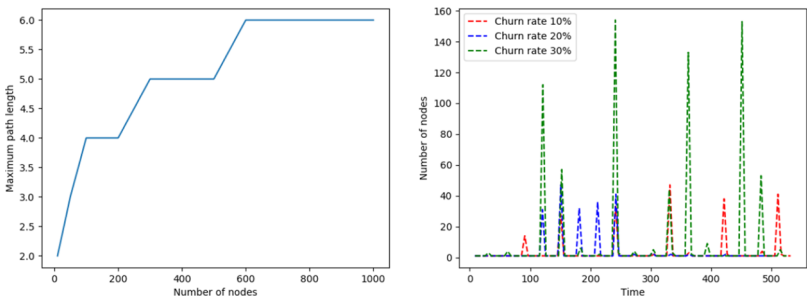
Reconciliation procedure executes at regular intervals of thirty minutes. It begins by randomly picking a neighbor at the start of an interval. A predefined time of seven days is chosen. When posts from last seven days are sorted by time, the first post's timestamp is picked, and shared with the neighbor. On receiving the reconciliation request. The neighbor chooses all the posts that started on or after the given timestamp, and sorts them according to timestamps. A list of IDs of these posts or comments are created. The list is divided into chunks of 256 entries. A hash value is calculated for each chunk, based on the post/comment IDs. These hash values are shared with the initiator. The initiator also calculates the hash values in the similar fashion, and the hash values are compared. If the hash value is different, then the initiator asks a neighbor to share the IDs in the particular chunk. It then gets the list of IDs in the chunk from the neighbor. Both of them make changes to the corresponding lists by adding missing IDs and recalculate hash values for that chunk. New list of hash values from the mismatched chunk is sent to the initiator. When the last chunk's hash matches reconciliation is over. Most of the time, only last few chunks would differ, and the number of message exchanges is low.

9 Simulation results

Emulab provides an environment for experiments over testbeds consisting of a large-scale distributed network. We deployed scripts for the experiments using the Emulab portal to acquire both physically distributed and purely simulated nodes [49]. It fitted the kind of experiments we wanted to run for establishing the efficacy of our approach to build a low latency file sharing, searching, and inline annotations framework ideally suitable for the E-Learning system using a P2P organization. As far as the whiteboard is concerned, sharing our primary motivation was to find out how a P2P learning system may work in a LAN environment and support up to a maximum of 250 nodes. We carried out the simulation experiments with up to 1000 nodes.

9.1 Live streaming and whiteboard

The first experiment is to find out the stability of our system. During the process of joining, as explained in Section 4, a peer gets a list of $\log_2 n$ active peers. Running simulations on up to 1000 nodes, we found that the choice of $\log_2 n$ leads to a stabilization of network as the size of the overlay increases. Once the bootstrap server returns the list, the peer send "adopt me" request to all the active peers in the list. A peer on receiving join request could either accept or discard the request based on its fanout value. Running



(a) Maximum path length. (b) Churning with 10%, 20%, and 30% churn rates.

Fig. 13 The results on Emulab for stabilization of the maximum path length and churning

Table 5 Minimum throughput value

Number of nodes	Throughput
20	177
50	152
90	173
120	156
150	139
170	173
200	174

simulations on up to 1000 nodes, we found that the choice of $\log_2 n$ leads to a stabilization of network as the size of the overlay increases. We found that the maximum path increases with an increase in the number of nodes, but later it gets stabilized. As shown in Fig. 13 from 700 to 1000 nodes, the maximum path length has stabilized to 6 units.

The mesh-overlay might get disconnected in the presence of churn. A node with 0 in-degree has no parent, and hence, it could not receive data until it finds at least one parent. So, the stabilization of the overlay is essential in the presence of churning. It means only one node could have in-degree 0. This node should be the source node only. We performed simulations on 1000 nodes for churn rate 10%, 20%, 30%. The results appear in Fig. 13b. The Y-axis denotes the number of nodes having in-degree equal to zero. The time is measured in seconds. We observed that even with 30% churn rate, the overlay stabilizes within 5 seconds.

Table 5 depicts the minimum throughput value analyzed on a different number of nodes. Here, throughput is defined as the number of packets received in one second.

We have obtained these results by performing experiments on Emulab. In Emulab experiments, each node receives 5000 packets from its parent nodes. For packet size of 1400 bytes and streaming rate of 2Mbps, the number of packets generated per second = 179. Hence, 5000 packets will be generated in $\frac{5000}{179} = 27.93$ seconds. Hence, we can see from Table 5 that our results are closely related to the theoretical results. The average time and maximum time to deliver packets are shown in Fig. 14a and b respectively. The maximum time to deliver packets is a reflection of the delay between the source node and the farthest node.

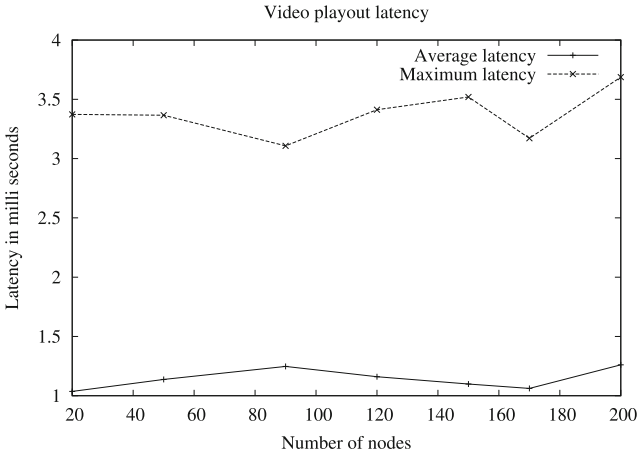


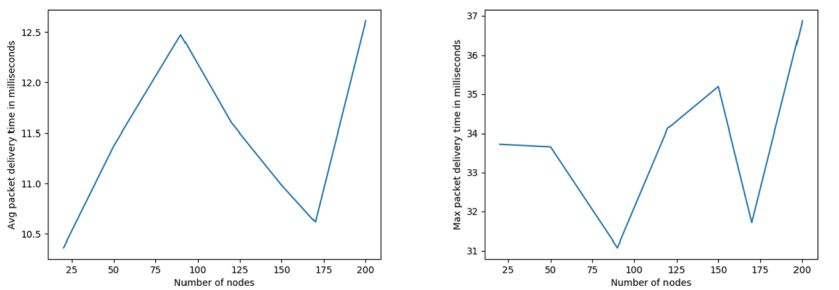
Fig. 15 Streaming latency experienced at the listener-end devices

In terms of practical utility of the system we need to ensure that absolute latency experience by a listener in live streaming should be long. We therefore, calculated observable latency at a listener device for varying number of nodes. The plot in Fig. 15 shows that the average latency varied between 10.5ms to 12.5ms, while the maximum latency varied between 31ms to 37ms. Since, we used modified mesh-based architecture for Bittorrent swarm type dissemination of video packets, it allowed us the keep latency under desired range.

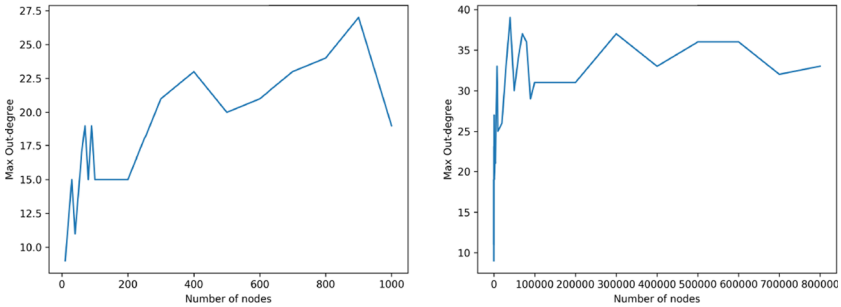
9.2 De Bruijn overlay

De Bruijn graph has diameter and out-degree equal to eight. At the application layer, the diameter does not exceed the value of eight. However, out-degree can vary according to the size of a zone. In theory, the maximum out-degree would be less than $K \times O(\log_K N)$ with high probability, where N is the number of nodes in the system. Considering

Fig. 14 Video packet delivery time for 200 nodes in milliseconds



(a) Average video packet delivery time (b) Maximum video packet delivery time



(a) Up to thousand. (b) Up to 800000.

Fig. 16 Maximum out-degree of nodes

$N = 800,000$ nodes, the maximum out-degree among all runs was 41. We experimented with different values of N and found the average out-degree to be 7.99. It is due to the absence of self-loops.

The graphs in Fig. 16a and b show the median values of the maximum out-degree of a node. The simulation results match the theoretical results. The graph shows that the median value for the maximum out-degree is 31 when $N = 100000$. Figure 17 shows the distribution of out-degree. It shows that very few nodes have degree $> 2 \times K$. Only 380 out of 100000 nodes have a degree of more than 16, which is only 0.38% of the total. None of the nodes have out-degree $> K \times \log_K 100000$. In an equivalent Chord implementation, the average out-degree is $O(\log N)$, which is > 20. Our experiments also determined that minimum in-degree=7 and the maximum in-degree=8.

Another experiment with the varying value of N from 100 to 100000 was performed where each node queried for 10 random keys. The results, plotted in Fig. 18, showed that average number of hops per query stayed well below the value of $\log_K(N)$. The distribution of access count of a node for $N = 100000$ and a million queries is shown in Fig. 19. For $N = 100000$, the average hop count is 5.51. It means, on an average, 5.51 nodes were accessed per query. The distribution shows that only 1741 nodes (or 1.74% nodes) were accessed more than 2 times the average, and only 89 nodes (or 0.089% nodes) were accessed more than 3 times the average access count. The maximum any node was accessed was 250 times. Any node in this system supports more than 1000000 routing queries per minute or more than 17000 routing queries per second. It would mean the system can support at least 68 times more query workload (or 680 queries per second per node) without any degradation in the performance.

We also carried out experiments on Emulab to find out the maximum latency and the success rate of lookups. In the experiment, the nodes were connected in a LAN environment. Every node randomly picked twenty-five words from the list of three thousand words available to everyone. After a delay of five minutes, the nodes sent out queries for the same set of words. The boot-up times of machines in Emulab differ up to ±5 minutes. Hence, the join procedure for the de Bruijn overlay network would have involved a transfer of the load for most of the nodes. The results were calculated for the varying number of nodes, as

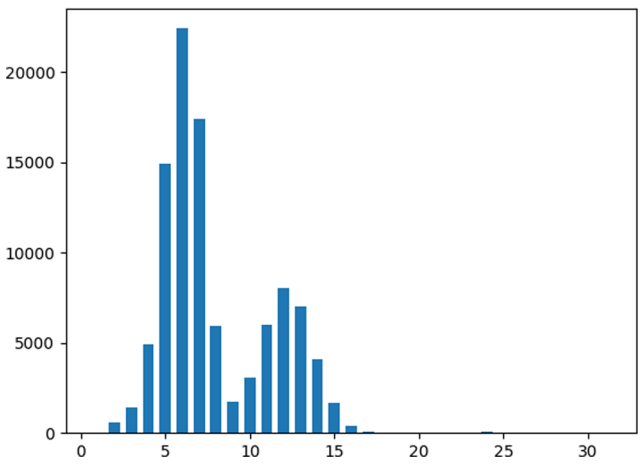


Fig. 17 Distribution graph of out-degree of nodes

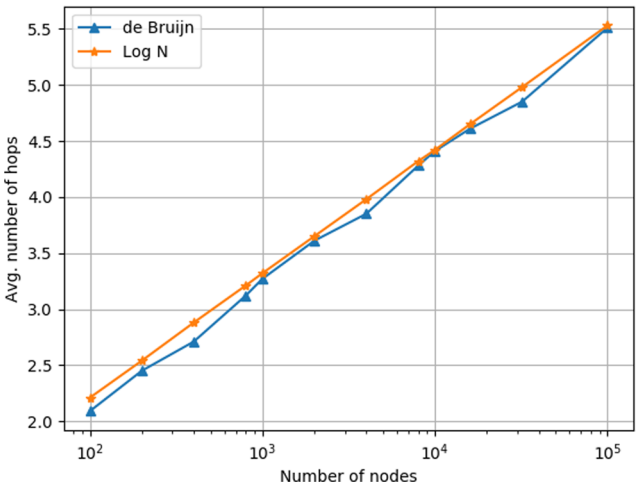


Fig. 18 Average number of hops for queries in de Bruijn overlay

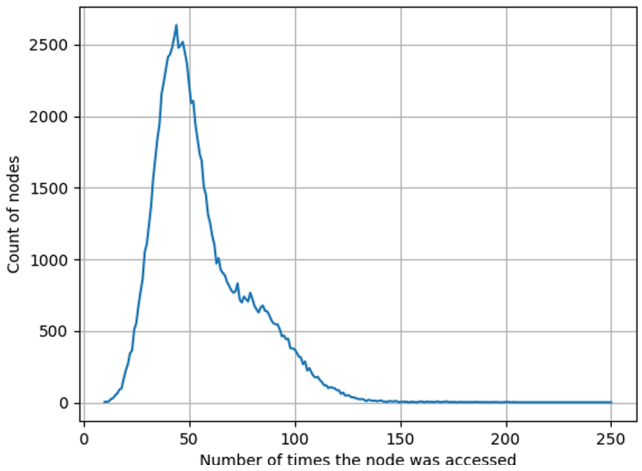


Fig. 19 Query load distribution in de Bruijn overlay with $N = 100000$ and 10 queries per node

shown in Table 6. It indicates that none of the lookups failed, even during the dynamic joins in the setup.

In another experiment, the nodes are allowed to leave the system with 10% probability after every three minutes. Our approach achieved a success rate of 99.39% and the maximum latency of 52ms for 200 nodes. Assuming that the human tolerance limit is about 200ms, the response is pretty good.

10 Related works

The scope of building innovative P2P applications is, thus, enormous. However, this paper focuses on just two key application spaces, namely, communication and collaboration. Instant messaging [27], VoIP services [10], P2P-SIP [9], social networking [41] and media streaming [54] are among the few popular P2P applications in communication space.

Most of deployments of P2P live streaming systems [24, 31, 39, 56] use mesh-pull architectures. Some of the recent work in focused on analytical modeling and efficiency of live streaming on P2P networks. Three different variations of multi-request mechanism has been presented in [55]. It concludes that there is the performance gap between analytical modeling and simulation which can be narrowed by evolving some realistic strategy on pull requests. It may provide comparative performance of a push-based mechanism. In a recent work Terelius and Johansson [52] investigated P2P network for efficient live streaming. They have provided a theoretical analysis of topology convergence of P2P network in presence of churn. The network graph converges to be a complete gradient overlay network. Interestingly, the proposed modified mesh network employs a modified packet reservation strategy akin to the concept of gradient network.

Bittorrent [42] also uses mesh-pull where each peer advertises the chunks of the stream it has in its cache. The advertisement is in the form of buffer maps. The peers use buffer maps to enhance viewing quality, start-up latency, bandwidth utilization, among others. In the context of P2P learning, we use a modified mesh architecture, which is presented in Section 3.

In a collaboration space, content distribution is a crucial application. In this paper, we are interested in the most basic form of content distribution, i.e., file sharing and searching [36, 40, 42, 47]. Two fundamental problems in file sharing are (i) to locate peers and (ii) to find a route between a pair of peers using proximity neighbor selection. Distributed Hash Tables (DHTs) [48] were introduced to handle the problems mentioned above. An efficient proximity neighbor selection is found to be highly effective when used in Distributed Hash Tables (DHT) with prefix-based routing like Pastry [46] or Tapstry [57]. In this paper, we use de Bruijn graph-based overlays, which also employ a prefix-based routing scheme. De Bruijn graph-based DHTs possess better degree diameter properties compared to other DHTs such as Chord [48], Pastry [46], Trie [19], CAN [45] and Butterfly [37]. Furthermore, as indicated by emulation both average and maximum out-degrees are low even for moderately large de Bruijn graphs. Some guidelines are available in the literature for de Bruijn graph-based implementation of DHTs [17, 20, 33]. However, it does not address the maintenance of DHT in the presence of churn (membership change over time due to peers leaving and joining unpredictably).

The main focus of the paper is P2P technology in E-Learning. A majority of E-Learning platforms are aided by a centralized core or cloud server [16, 28] because querying learning objects is far more complicated than searching or fetching a document or a music file.

Table 6 Lookup latency and success rate

No. of nodes	Success rate	Max Lat.(ms)	Avg. Lat.(ms)
80	100%	50	16
120	100%	70	20
160	100%	58	21
200	100%	52	21

A critical aspect of the research on P2P E-Learning is on peer to peer data management and provide infrastructure for complex queries on educational objects [38, 51]. Edutella [38] project used W3C metadata standard RDF to provide an infrastructure on which complex queries are possible over P2P networks based on JXTA platform [22]. Piazza [51] addressed the problem of sharing semantically heterogeneous data in a distributed manner. P2P architectures in E-Learning space meant for facilitating the download of large files in the background through a lightweight P2P core [43], which integrates a Bittorrent-based small self-producing component.

Some existing applications of P2P technology in E-Learning focused on load balancing by creating an integrated model consisting of P2P and client-server model [30]. The idea is to separate the server functions from learning content. The functions of the server is distributed in a P2P manner. The peers manage the learning contents and provide them to the learners. So the computation and storage costs are distributed. Since learning objects are not stored at the user's machine, copyright issues do not arise. However, the system cannot be considered as a P2P model for E-Learning.

The IEEE/ACM 2013 computer science curricula listed out challenges in creating and maintaining heterogeneous test-beds which is needed for research, teaching, and learning concerning computer network technologies on a global scale. Global Environment for Network Infrastructure [12] and Future Internet Research and Experiments [21] are two important learning projects which focused on near-ubiquitous availability of heterogeneous test-beds for carrying out experiments on future internet technologies for learning. FORGE [29] toolkit leveraged both FIRE and GENI for developing learning material to provide an eco-system for teaching and self-learning using tools and experiments available under open policies. Teaching and experiments related to computer science courses in universities Seattle project [4] offered access to Planet Lab for deploying and test student assignments.

No E-Learning platform exists, at least in our knowledge, which provides an environment anywhere close to the P2P-based presentation system that mimics peer interactions on a live presentation and allows learners to tag and annotate learning material to raise queries, etc. The availability of P2P interactive whiteboard in conjunction with live streaming, gives users a feel of physical attendance in a remote presentation. The only proposal that comes somewhere close to our system is by Priyankara et al. [44]. It essentially proposes a P2P content sharing mechanism or a natural extension of file sharing and searching capabilities on learning material. It is evident from related literature that most E-Learning platforms using P2P technology are intended for monitoring or individual mentoring requirements of the learners in a framework of remote learning paradigm.

11 Conclusion

We have studied two different architectures to support live streaming. Based on the analysis, we modified mesh architecture to support dynamic fanout leveraging spare capacity whenever available. Hence, our system could also support heterogeneous nodes satisfying the minimal bandwidth requirement, as explained in Section 3. In our experience with actual sessions on the live board, we found a consistent display of results. The repainting was quite efficient at all peer nodes. The buffered approach has helped to optimize the P2P shared whiteboard's performance.

Our simulations on Emulab proved that even in the presence of churning, the overlay structure for live streaming gets stabilized moderately quickly. Since the maximum path length also gets stabilized, the latency is bearable in Live sessions of P2P interactions with live streaming. The Emulab experiments are important to establish that experimental throughput is close to theoretical throughput values. Both the current and earlier version of the system is available from Bitbucket links:

1. https://bitbucket.org/p2pElearning/icls/src/master/ [14].
2. https://bitbucket.org/p2pElearning/distro1/src/master/ [23].

.

As a part of future work, we plan to include more tools in our shared whiteboard and enable group-undo operation. Various video-coding techniques are available to neutralize the effect of packet loss. In the future, we would like to incorporate those techniques for live streaming. The back end support for file sharing, searching, and inline annotation also needs to be integrated fully with a learning management system.

Appendix A: Send join request algorithm

Algorithm 7 ReceivedJoinRequest.

Require: *Join request packet JOIN(ID, JoinExtAddr);*
if *(ID ∈ MyZone)* **then**
    send MyZone, MyID, MyOutEdge, MyInEdges to JoinExtAddr;
    queue new joins until this join completes or timeOut==10s;
    receive ID and Zone information of the new node;
    split MyZone into half and discard the new node's Zone;
    identify and insert in appropriate lists any new edges with the newNode's Zone and MyZone;
    notify all neighbors about update of Zone by sending MyZone and MyID;
    drop edges (with any of the neighbors) that no longer exists;
    share existing Key, Value pairs in lying new node's Zone with the new node;
end
else
    // Receiver is a forwarder;
    identify the next node on routing path to which request is to be forwarded;
    send JoinRequest(ID, JoinExtAddr) to next node;
end

Appendix B: Received join request algorithm

Algorithm 7 ReceivedJoinRequest.

Require: *Join request packet JOIN(ID, JoinExtAddr);*
if *(ID ∈ MyZon)* **then**
    send MyZone, MyID, MyOutEdge, MyInEdges to JoinExtAddr;
    queue new joins until this join completes or timeOut==10s;
    receive ID and Zone information of the new node;
    split MyZone into half and discard the new node's Zone;
    identify and insert in appropriate lists any new edges with the newNode's Zone and MyZone;
    notify all neighbors about update of Zone by sending MyZone and MyID;
    drop edges (with any of the neighbors) that no longer exists;
    share existing Key, Value pairs in lying new node's Zone with the new node;
end
else
    // Receiver is a forwarder;
    identify the next node on routing path to which request is to be forwarded;
    send JoinRequest(ID, JoinExtAddr) to next node;
end

Appendix C: Send leave request

Algorithm 8 SendLeaveRequest.

```
Require: The address of the rendezvous server, socket;
left ← FALSE;
while (!left) do
    if (MyZone covers entire ID-space) then
        inform the server that all other nodes have left;
        left ← TRUE;
    end
    else
        list ← [MyZone.startID-1, MyZone.endID + 1];
        // succID is the ID of the successor;
        succID ← randomID(list);
        if succID lies with one of the neighbors then
            next_node ← address of the neightbor containing the succID;
        end
        else
            create a routing path from MyID to succID;
            next_node ← address of the next node from the routing path;
        end
        send LeaveRequest(ID, Zone, OutEdges, InEdges, extAddress) to next_node;
        wait for reply up to 5 seconds;
        if (Reply == "Yes") then
            send confirmation and keys in MyZone to successor;
            inform the rendezvous server, that leave is complete;
            left ← TRUE;
        end
    end
end
```

Appendix D: Received leave request

Algorithm 9 ReceivedLeaveRequest.

```
Require: Leave request packet LEAVE(ID, Zone, OutEdges, InEdges, LeaveExtAddr);
// LeaveExtAddr is the external address of leaving node;
if (ID ∈ MyZone) then
    if (not leaving the system) then
        send "Yes" to LeaveExtAddress;
        wait for confirmation up to 5 seconds;
        if (confirmed) then
            merge Zone, OutEdges and InEdges;
            receive the load;
        end
    end
    else
        // Leaving self;
        send "No" to LeaveExtAddress;
    end
end
else
    identify the next node on path to which the request should be forwarded;
    send LEAVE(ID, Zone, OutEdges, InEdges, LeaveExtAddr) to the next node;
end
```

References

1. Brainly.in - for students. by students. https://brainly.in/. Accessed 24 June 2019
2. Coursera. https://www.coursera.org/. Accessed 24 June 2019
3. Kahoot. https://kahoot.com/. Accessed 24 June 2019
4. Seattle: open peer-to-peer computing. seattle.poly.edu/ (2016). Accessed 24 June 2019

5. Github guides. https://guides.github.com/activities/hello-world/ (2018). Accessed 13 August 2018
6. GSuite learning center. https://support.google.com/a/users/?hl=en#topic=9296556 (2018). Accessed 13 August 2018
7. Overleaf training resources. https://www.overleaf.com/tutorial (2018). Accessed 13 August 2018
8. How to make the most of a conference call. https://today.duke.edu/2019/10/how-make-most-conference-call (2019). Accessed 24 March 2020
9. Ahson SA, Ilyas M (2018) P2p SIP: network architecture and resource location strategy. In: SIP handbook. CRC Press, pp 221–244
10. Azfar A, Choo KKR, Liu L (2014) A study of ten popular android mobile voIP applications: are the communications encrypted? In: 47th Hawaii international conference on system sciences. IEEE, pp 4858–4867
11. Baset SA, Schulzrinne H (2004) An analysis of the skype peer-to-peer internet telephony protocol. arXiv:cs/0412017
12. Berman M, Chase JS, Landweber L, Nakao A, Ott M, Raychaudhuri D, Ricci R, Seskar I (2014) GENI: a federated test-bed for innovative network experiments. Comput Netw 61
13. Bernazzani S (2020) Everything you need to know about using zoom. https://www.owllabs.com/blog/zoom. Accessed 23 March 2020
14. Bhagatkar N, Dolas K, Ghosh RK (2018) Integrated collaborative learning software. https://bitbucket.org/p2pElearning/icls/src/master/. Accessed 20 March 2020
15. Brown S (2020) Skype vs. zoom: which video chat app is best for working from home? https://www.cnet.com/news/skype-vs-zoom-which-video-chat-app-is-best-for-working-from-home/. Accessed 23 March 2020
16. Cole J, Foster H (2007) Using moodle: teaching with the popular open source course management system. O'Reilly Media, Inc.
17. Datta A, Girdzijauskas S, Aberer K (2004) On de Bruijn routing in distributed hash tables: there and back again. In: Proceedings. Fourth international conference on peer-to-peer computing, 2004. Proceedings. IEEE, pp 159–166
18. Falcone K (2018) A case study of faculty experience and preference of using blackboard and canvas lms. Ph.D. thesis, University of Phoenix
19. Freedman MJ, Vingralek R (2002) Efficient peer-to-peer lookup based on a distributed trie. In: International workshop on peer-to-peer systems. Springer, pp 66–75
20. Gai A, Viennot L (2004) Broose: a practical distributed hashtable based on the de-Bruijn topology. In: Proceedings of fourth international conference on peer-to-peer computing. IEEE, pp 167–174
21. Gavras A, Karila A, Fdida S, May M, Potts M (2007) Future internet research and experimentation: the FIRE initiative. ACM SIGCOMM Comput Commun Rev 37(3):89–92
22. Gong L (2001) Project JXTA: a technology overview. Tech. rep., SUN Microsystems. Accessed 20 March 2020
23. Gupta V, Kumar S, Hussian M, Ghosh RK (2017) P2p E learning. https://bitbucket.org/p2pElearning/distro1/src/master/. Accessed 24 June 2019
24. Hei X, Liang C, Liang J, Liu Y, Ross KW (2006) Insights into PPLive: a measurement study of a large-scale P2P IPTV system. In: Proceeding of international world wide web conference of IPTV workshop
25. Horvat A, Dobrota M, Krsmanovic M, Cudanov M (2015) Student perception of Moodle learning management system: a satisfaction and significance analysis. Interactive Learning Environments 23(4):515–527
26. Icard SB (2014) Educational technology best practices. Int J Instruct Technol Distance Learn 11(3):37–41
27. Jennings RB, Nahum EM, Olshefski DP, Saha D, Shae ZY, Waters C (2006) A study of internet instant messaging and chat protocols. IEEE Netw 20(4):16–21
28. John R (2014) Canvas LMS course design. Packt Publishing Ltd
29. Jourjon G, Marquez-Barja JM, Rakotoarivelo T, Mikroyannidis A, Lampropoulos K, Denazis S, Tranoris C, Pareit D, Domingue J, Dasilva LA, Ott M (2017) FORGE toolkit: leveraging distributed systems in elearning platforms. IEEE Trans Emerging Topics Comput 5(1):7–19
30. Kawato T, Higashino M, Takahashi K, Kawamura T (2019) Proposal of e-learning system integrated P2P model with client-server model. In: 2019 international conference on electronics, information, and communication (ICEIC), pp 1–6
31. Kumar R, Liu Y, Ross K (2007) Stochastic fluid theory for P2P streaming systems. In: 26th IEEE international joint conference on computer communications (INFOCOM 2007). IEEE, pp 919–927
32. Liu Y, Guo Y, Liang C (2008) A survey on peer-to-peer video streaming systems. Peer-to-Peer Networking and Applications 1(1):18–28
33. Loguinov D, Kumar A, Rai V, Ganesh S (2003) Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In: Proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications. ACM, pp 395–406
34. Magharei N, Rejaie R (2009) Prime: peer-to-peer receiver-driven mesh-based streaming. IEEE/ACM Transactions on Networking (TON) 17(4):1052–1065
35. Magharei N, Rejaie R, Guo Y (2007) Mesh or multiple-tree: a comparative study of live p2p streaming approaches. In: INFOCOM 2007. 26th IEEE international conference on computer communications, pp 1424–1432
36. Mahanta K, Khataniar G (2013) Peer-to-peer (P2P) file sharing system: a tool for distance education. Int J Comput Sci Inform Technol Secur (IJCSITS) 3:155–158
37. Malkhi D, Naor M, Ratajczak D (2002) Viceroy: a scalable and dynamic emulation of the butterfly. In: Proceedings of the twenty-first annual symposium on principles of distributed computing. ACM, pp 183–192
38. Nejdl W, Wolf B, Qu C, Decker S, Sintek M, Naeve A, Palmér M, Risch T (2002) EDUTELLA: a P2P networking infrastructure based on RDF. In: Proceedings of the 11th international conference on world wide web, WWW '02, pp 604–615
39. Palacios S, Santos V, Barsallo E, Bhargava B (2019) MioStream: a peer-to-peer distributed live media streaming on the edge. Multimed Tools Appl 78:24,657–24,680
40. Parameswaran M, Susarla A, Whinston AB (2001) P2P networking: an information sharing alternative. Computer 34(7):31–38
41. Paul T, Famulari A, Strufe T (2014) A survey on decentralized online social networks. Computer Networks 75:437–452
42. Pouwelse J, Garbacki P, Epema D, Sips H (2005) The Bittorrent P2P file-sharing system: measurements and analysis. In: International workshop on peer-to-peer systems. Springer, pp 205–216
43. Prakash LS, Saini DK, Kutti NS (2009) Integrating EduLearn learning content management system (lcms) with cooperating learning object repositories (LORs) in a peer to peer (P2P) architectural framework. ACM SIGSOFT Software Engineering Notes 34(3):1–7
44. Priyankara HAC, Jayasuriya KAKDDB, Gunasekara UDND (2018) Vyapthi: a leveraged P2P content sharing platform for distributed e-learning systems. In: 2018 18th international conference on advances in ICT for emerging regions (ICTER), pp 126–132
45. Ratnasamy S, Francis P, Handley M, Karp R, Shenker S (2001) A scalable content-addressable network, vol 31. ACM

46. Rowstron A, Druschel P (2001) Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: IFIP/ACM international conference on distributed systems platforms and open distributed processing. Springer, pp 329–350
47. Saroiu S, Gummadi PK, Gribble SD (2002) Measurement study of peer-to-peer file sharing systems. In: Multimedia computing and networking, vol 4673. International Society for Optics and Photonics, pp 156–171
48. Stoica I, Morris R, Liben-Nowell D, Karger DR, Kaashoek MF, Dabek F, Balakrishnan H (2003) Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans Netw (TON) 11(1):17–32
49. Stoller MHRRL, Duerig J, Guruprasad S, Stack T, Webb K, Lepreau J (2008) Large-scale virtualization in the Emulab network testbed. In: USENIX annual technical conference, Boston, MA
50. Stutzbach D, Zappala D, Rejaie R (2005) The scalability of swarming peer-to-peer content delivery. In: International conference on research in networking. Springer, pp 15–26
51. Tatarinov I, Ives Z, Madhavan J, Halevy A, Suciu D, Dalvi N, Dong XL, Kadiyska Y, Miklau G, Mork P (2003) The piazza peer data management project. SIGMOD Rec 32(3):47–52
52. Terelius H, Johansson KH (2018) Peer-to-peer gradient topologies in networks with churn. IEEE Trans Control Netw Sys 5(4):2085–2095
53. Vesselinov R, Grego J (2012) Duolingo effectiveness. Tech. rep., City University of New York, USA. Accessed on 13 August 2018
54. Yamada Y, Fujita S (2018) Load balancing in P2P video streaming systems with service differentiation. In: 2018 sixth international symposium on computing and networking workshops (CANDARW), pp 539–543
55. Zhang J, Zhang X, Yang C (2018) Towards the multi-request mechanism in pull-based peer-to-peer live streaming systems. Comput Netw 138:77–89
56. Zhang X, Liu J, Li B, Yum Y (2005) Coolstreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming. In: Proceedings of 24th IEEE annual joint conference of the IEEE computer and communications societies INFOCOM 2005, vol 3. IEEE, pp 2102–2111
57. Zhao BY, Huang L, Stribling J, Rhea SC, Joseph AD, Kubiatowicz JD (2004) Tapestry: a resilient global-scale overlay for service deployment. IEEE J Select Areas Commun 22(1):41–53

Nikita Bhagatkar completed her M.Tech. degree in CSE from IIT Kanpur in 2018. She currently holds position of Software Developer in Cisco India.

Kapil Dolas completed his M.Tech. degree in CSE from IIT Kanpur in 2018. Currently he is working as a Software Engineer for Google India. His primary area of interest is Distributed Computing.

R. K. Ghosh is a professor in the Department of Computer Science & Engineering at the Indian Institute of Technology, Kanpur. Earlier for a brief duration (2001-2002), he also held a Professor's position in the Department of Computer Science & Engineering at Indian Institute of Technology, Guwahati. A few of the other positions he has held in the past include UN Teacher Fellow at IIST Macau, Visiting Scientist at INRIA Sophia Antipolis, France, Visiting Faculty in the Dept of computer Science at the University of Texas at Arlington, USA. His primary research interests are in mobile computing, distributed systems and wireless networks. He has authored 3 books one titled "Wireless Networking and Mobile Data Management" have been published recently. He serves as an associate editor of Indian Academy of Science journal on Engineering Sciences. Dr. Ghosh has been Principal Investigator of several projects related to parallel processing, distributed software engineering, mobile and cloud computing, many of which involved international collaborations.

Sajal K. Das is a professor of Computer Science and the Daniel St. Clair Endowed Chair at the Missouri University of Science and Technology, where he was the Chair of Computer Science Department during 2013-2017. He served the NSF as a Program Director in the Computer Networks and Systems division during 2008-2011. Prior to 2013, he was a University Distinguished Scholar Professor of Computer Science and Engineering and founding director of the Center for Research in Wireless Mobility and Networking at the University of Texas at Arlington. His research interests include security and trustworthiness, wireless sensor networks, mobile and pervasive computing, crowd sensing, cyber-physical systems and IoTs, smart environments (smart city, smart grid and smart health care), distributed and cloud computing, biological and social networks, and applied graph theory and game theory. He has published extensively in these areas with over 700 research articles in high quality journals and refereed conference proceedings. Dr. Das holds 5 US patents and coauthored 4 books. He serves as the founding Editor-in-Chief of Elseviers Pervasive and Mobile Computing Journal, and as Associate Editor of several journals including the IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Mobile Computing, and ACM Transactions on Sensor Networks. Dr. Das is an IEEE Fellow.