Noname manuscript No.
(will be inserted by the editor)

CliqueStream: Creating an Efficient and Resilient Transport Overlay for Peer-to-Peer Live Streaming using a Clustered DHT

Shah Asaduzzaman · Ying Qiao · Gregor v. Bochmann

the date of receipt and acceptance should be inserted later

Abstract Several overlay-based live multimedia streaming platforms have been proposed in the recent peer-to-peer streaming literature. In most of the cases, the overlay neighbors are chosen randomly for robustness of the overlay. However, this causes nodes that are distant in terms of proximity in the underlying physical network to become neighbors, and thus data travels unnecessary distances before reaching the destination. For efficiency of bulk data transmission like multimedia streaming, the overlay neighborhood should resemble the proximity in the underlying network. In this paper, we exploit the proximity and redundancy properties of a recently proposed clique-based clustered overlay network, named eQuus, to build efficient as well as robust transport overlays for multimedia streaming. To combine the efficiency of content pushing over tree structured overlays and the robustness of data-driven mesh overlays, higher capacity stable nodes are organized in tree structure to carry the long haul traffic and less stable nodes with intermittent presence are organized in localized meshes. The overlay construction and fault-recovery procedures are explained in details. Simulation study demonstrates the good locality propoerties of the platform. The outage time and control overhead induced by the failure recovery mechanism are minimal as demonstrated by the analysis.

S. Asaduzzaman
School of Information Technology Engineering
University of Ottawa
800 King Edward Avenue, Room 5-105
Ottawa, ON, Canada K1N 6N5
Tel: +1-613-562-5800 ext-2190
Fax: +1-613-562-5664
E-mail: asad@site.uottawa.ca

Y. Qiao
School of Information Technology Engineering
University of Ottawa
Ottawa, ON, Canada K1N 6N5
E-mail: yqiao074@site.uottawa.ca

G. v. Bochmann
School of Information Technology Engineering
University of Ottawa
Ottawa, ON, Canada K1N 6N5
E-mail: bochmann@site.uottawa.ca

1 Introduction

With the widespread adoption of broadband residential Internet access, live multimedia streaming over the IP network may be envisioned as a dominating application on the next generation Internet. Global presence of the IP network makes it possible to deliver large number of commercial as well as amateur TV channels to a large population of viewers. Based on the peer-to-peer (P2P) communication paradigm, live multimedia streaming applications have been successfully deployed in the Internet with up to millions of users at any given time. With commercial implementations like CoolStreaming [21], PPLive [8], TVAnts [15] and UUSee [20], among others, large volumes of multimedia content from hundreds of live TV channels are now being streamed to users across the world.

Although naive unicast over IP works for delivering multimedia stream to a restricted small group of clients, the overwhelming bandwidth requirement makes it impossible when the number of user grows to thousands or millions. Several different delivery architectures are used in practice for streaming of live video content, which include IP multicast [5], infrastructure-based application layer overlays [7] and P2P overlays. P2P overlays are gaining popularity due to their ease of large-scale deployment without requiring any significant infrastructure.

Live multimedia streaming over P2P networks has several challenges to be addressed. Unlike file sharing, the live media need to be delivered almost synchronously to large

number of users, with minimum delay in playback compared to the playback at the source. Due to the large volume of data in the media stream, it is of paramount interest to avoid redundant transmission of the stream. Constructing efficient paths for streaming is especially hard because the nodes participating in the overlay have very minimal information regarding the topology of the underlying physical data transmission network. Moreover, the intermittent joining and leaving behavior, or *churn*, of the nodes makes it harder to maintain the overlay delivery paths once constructed. Heterogeneity of node bandwidths adds further complexity to the problems.

Existing P2P live streaming platforms can be broadly classified into two categories – *tree based* and *mesh based*. In the tree based platforms, nodes are organized in a tree topology with the streaming source at the root. The media content is pro-actively pushed through the tree. Although efficient in terms of avoiding redundant transmissions, the nodes that happen to be interior nodes in the tree bear an unfair burden of forwarding the content downstream compared to the nodes that become leaves of the tree. Some multi-tree approaches like SplitStream [2] and ChunkySpread [17] have been proposed that avoid this imbalance taking advantage of multiple description coding of the media. Nevertheless, a major argument against the tree-based overlays is that it is expensive to maintain the trees in presence of frequent node join and leave or *churn*.

A dramatically different approach is to allow each node to choose a small random set of overlay neighbors and thus create a mesh topology. The stream is divided into small fragments and each node comes to know what fragments are possessed by its neighbors through periodic exchange of their buffer-maps [21]. Required fragments to fill the current playback buffer are then downloaded or *pulled* from the neighbors as needed. Because of the unstructured and random nature of the topology, the mesh-based platforms are more robust to *churn*. However, there are several inherent disadvantages in the pull process such as longer delay and higher control overhead.

In most of the P2P streaming platforms, the overlay neighbors are chosen randomly [20, 21], which is important for maintaining global connectivity of the overlay network. However, this causes nodes that are distant in terms of proximity in the underlying physical network to become neighbors. There are two problems that arise from such random selection of neighbors. First, data travels unnecessary distances before reaching the destination. Second, because the data travel path is uncorrelated with the locality of the destination nodes, two nodes of very close proximity may receive data through completely disjoint paths from the source. This causes significant redundancy in data transmission and costs a huge amount of network bandwidth for the whole platform.

In this paper, we present the design of a P2P media streaming platform named CliqueStream that exploits the properties of a clustered P2P overlay to achieve the locality properties and robustness simultaneously. The clustered peer-to-peer overlay named eQuus [12] organizes the nodes into clusters of proximal nodes. It assigns identifiers to clusters and replicates the routing information among all nodes in a given cluster. The assignment of identifier also imposes a structured mapping of the identifier space to the proximity space.

We also exploit the existence of more stable and higher bandwidth nodes in the network to allow construction of efficient delivery structures without causing too much overhead from churn. Existence of stable nodes, or *super nodes*, are observed both in file sharing networks and media streaming networks [18]. Our proposed platform elects one or more stable nodes of highest available bandwidth in each cluster and assigns a special relaying role to them. To maintain transmission efficiency, a content delivery tree is constructed out of the stable nodes using the structure in the underlying routing substrate and content is pushed through them. Less stable nodes within a given cluster then participate in the content dissemination and pull the content creating a mesh around the stable nodes.

In most implementations of P2P streaming platforms, a separate streaming overlay is created for distribution of media from each source, usually called a *channel*. We argue that the user's participation behavior for individual channel is significantly different from the participation behavior with respect to the whole streaming platform. A user usually switches channels frequently while keeping the TV turned on for a long time. Therefore it is intuitively beneficial to have a two-layer architecture, where a single routing overlay is maintained for the whole platform and streaming paths are rapidly constructed for individual channels based on the structure of the substrate. Performance comparison between per-channel overlay and single overlay supporting multiple channels also supports the latter organization [4].

The rest of the paper starts with a review of the relevant features of the clustered P2P overlay named eQuus and discussion on the modifications we made into it. The design of the platform with details of its functional components is presented in Section 3. In Section 4 we discuss the locality and fault-tolerance properties of the platform.

2 eQuus: a Clustered DHT

2.1 Overview of eQuus

eQuus [12] is a structured peer-to-peer overlay which forms a distributed hash table (DHT) consisting of clusters or *cliques* of nodes instead of individual nodes. A unique identifier (id) is assigned to each clique instead of each individual

node. Nodes in the same clique are closer to each other than nodes in different cliques, based on some proximity metric such as latency. These nodes maintain an all-to-all overlay neighborhood among them, and hence they are collectively denoted as a *clique*. The number of nodes in a clique and the spread of these nodes in the proximity space are small enough to allow such all-to-all neighborhood. Although the DHT overlay is formed among cliques, individual nodes keep data structures (or routing tables) to represent the overlay, by maintaining pointers to individual nodes in other relevant cliques. IP addresses of the nodes are stored along with clique ids as pointers.

Unlike many DHT overlays, the nodes or the cliques do not assume random ids. Rather, the segmentation of the id-space closely resembles the segmentation of the proximity space into cliques. If all possible ids define the id space, each clique occupies a certain numerically contiguous segment of the id-space. Due to the id assignment process explained later in this section, cliques with numerically adjacent ids occupy adjacent segments of the proximity space. All the existing cliques in the network can thus form a successor-predecessor relationship based on the numerical sequence of the ids such that the successor and predecessor cliques are adjacent to each other.

As a new node joins eQuus network it becomes a member of the closest clique in the proximity space. To do this, the new node has to find a node in the desired clique. If it is assumed that an arbitrary node already connected to the overlay is known, it is possible to discover a node in the closest clique in logarithmic number of message exchange rounds as described in [12]. When the new node sends a join message to the contact node, it returns the list of all nodes in its routing table. The new node then determines the closest among the newly discovered nodes by pinging. Another join message is then sent to the closest node and the whole process is repeated until no node closer than the previously found nodes is discovered. An alternative method of finding the closest node connected to the overlay is to perform an expanding ring search in the IP network and contacting the discovered nodes for overlay connectivity. However, in this approach, the search time depends on the density of the overlay conneted node in the proximity of the new node and is not related to the number of nodes in the overlay.

Arrival of new nodes may cause a clique to contain more nodes than a system-defined threshold. The clique then offshoots a new clique by splitting itself into two halves. One of the halves retains the previous id. The other half gets a new id that differs from its parent's id by only one bit, effectively splitting the id space occupied by the parent clique into two halves. As the network grows, numerically consecutive segments of the id space is thus assigned to adjacent cliques. In fact, if the cliques are ordered by their numerical id they occupy the consecutive positions in a space-filling curve that fills the whole proximity-space. This is illustrated in Figure 1(a). Thus two cliques with numerically close ids are always close to each other in the proximity space, although the reverse may not always be true. Moreover, the longer the matching prefix of two different ids, the closer they are positioned. In other words, all the nodes in the whole id space may be hierarchically divided into local groups based on the length of the matching prefix in their ids. For example the cliques sharing id prefix 1011 may resemble a local group which is further divided into two sub-groups with prefixes 10110 and 10111.

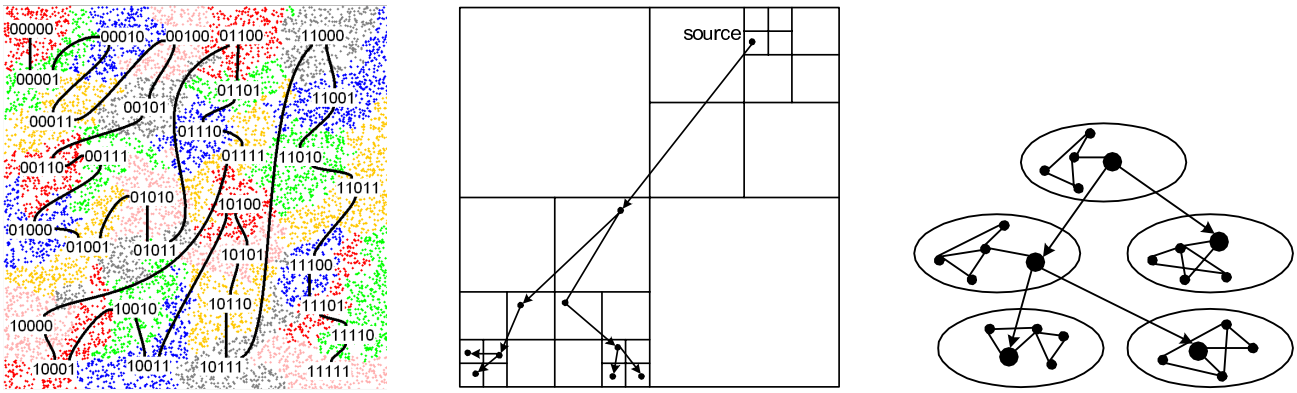
A message is routed towards a clique containing a certain id using the standard prefix matching algorithm. All nodes in the same clique share the same routing table in terms of clique neighborhood. The routing table contains clique ids with different length of prefix match with the current cliques id. For each clique id, IP addresses of k random nodes of that particular clique are stored. The routing table at the nodes in a given clique may vary in terms of these k pointers.

The prefix-matched routing implies that if a message is routed from clique A to clique B and also to clique C, the message will be first carried to a region that shares the common prefix of B and C along a common path. The path will then diverge towards each of B and C. The id assignment process ensures that the closer B and C are in terms of proximity, the longer is their common prefix. This implies that messages from a single source to multiple destinations in close proximity will travel along a long common path before diverging (Figure 1(b)). We exploit this property to create network efficient dissemination trees for live video streaming from a single source.

2.2 Introducing Stable Nodes

We modify the original design of eQuus by introducing stable nodes. Heterogeneous stability and capacity characteristics of the nodes are common in peer-to-peer networks. Thus the existence of stable nodes, or *super nodes*, is well-established in both file sharing and streaming peer-to-peer networks [18]. In the case that all the nodes receiving the streams are similarly low capacity and unstable, some high capacity servers may be deliberately introduced in locations across the network which may act as stable nodes.

We assume that each clique maintains t stable nodes all the time, where t is a system parameter. Stable nodes are elected from the existing eligible nodes in the clique. A node becomes eligible to be a stable node after being alive for a certain amount of time T, based on the observation that the more uptime a node has spent, the longer is its expected time before departure [18]. The clique always elects t nodes having highest outgoing bandwidths among the eligible nodes. For bootstrapping, when there is a single node in the whole



(a) The mapping of id-space to proximity space in eQuus (reproduced from [12])

(b) Streaming tree over eQuus cliques

(c) Streaming topology in CliqueStream

Fig. 1 Proximity and streaming topology

network, it immediately becomes a stable node. The election is initiated whenever the recruitment of a new stable node becomes necessary.

To reduce the overhead of maintaining a replica of the clique routing table at each node of a clique, we replicate the table among the stable nodes only. However, the all-to-all neighborhood of the nodes within a clique still remains. In addition, each node in a clique maintains the knowledge of which nodes in the clique are acting as stable nodes. This knowledge is updated whenever a new stable node is elected or an existing stable node ceases to act as a stable node. When a clique is split, a stable node remains a stable node in the cliques that are born after the split of a clique. New stable nodes are elected at the event of split to maintain sufficient number of stable nodes in each clique.

2.3 Modification in the Routing Mechanism

Inclusion of the stable nodes causes some modifications to the original routing method of eQuus. These modifications also assist in the construction of the transport overlay such that the stream of any particular channel is carried between two different cliques through only one link. Because inter-clique neighborhood is maintained only by the stable nodes, messages are routed through the stable nodes when traveling between cliques. Each stable node in a clique maintains addresses of k nodes for each clique it has as its routing table entry. Each node periodically updates this list of k nodes and always tries to have at least one of them to be a stable node. While routing a message to a particular clique, based on the routing table match, the stable node is preferentially selected instead of randomly choosing one of the k nodes. However, the routing works even if none of the k nodes is a stable node. If a non-stable node receives a message, it forwards the message to some stable node in the same clique.

In the live streaming platform, a CliqueStream layer is implemented on top of the modified eQuus routing substrate. The routing substrate provides the service of routing any message to one of the stable nodes of the clique specified by the destination clique id. The CliqueStream layer constructs the transport overlay for streaming, using the path taken by a message when routed by the substrate. To enable such construction, the routing substrate notifies the CliqueStream layer by invoking a callback method named *forward(msg)* before forwarding any CliqueStream message to the next hop. The *forward* method may modify the message including its destination. The routing layer then processes the modified message and forwards it accordingly. To stop forwarding of the message to further hops, the destination clique id may be modified to a *null* value. When a message arrives at its destination node, the routing layer delivers the message to the CliqueStream layer by invoking the *deliver* method.

3 System Overview

In this section we present the details of the CliqueStream video streaming platform. The purpose of the platform is to facilitate live streaming of multimedia content generated from arbitrary source nodes to a large set of destinations nodes. We use the term *channel* to denote a live stream of content from a single source, in a similar way used by television networks. In CliqueStream, a large number of streaming channels can be delivered using a single routing overlay, instead of creating and maintaining a separate overlay for each channel. A transport overlay is created on the routing substrate described in Section 2, for dissemination of each channel. The shared routing substrate allows better balancing of the forwarding load among the participating nodes.

3.1 Overlay Topology and Streaming Procedure

The topology of the transport overlay in CliqueStream is a combination of tree and mesh structure. We exploit the proximity features of the routing substrate described in Section 2 to form an efficient topology. We consider the total amount of *distance* travelled by a packet of media content in the proximity space to be delivered to every receiver, as a measure of efficiency. This reflects the load on the underlying physical network exerted by the stream transport overlay. A more compact transport overlay is thus treated as more a efficient one.

Because the nodes in a single clique are close to each other, arbitrarily interconnecting them in a mesh does not incur any significant inefficiency in the network. If at least one node in a clique receives the stream, other nodes in the same clique can form data-exchange partnerships as in CoolStreaming [21] and receive the channel. Therefore, we need some mechanism to deliver the stream to at least one node in each clique that has some nodes trying to receive the stream. For each channel, a dissemination tree is formed including a single stable node from each participating clique. The source of the stream is at the root of the tree. The stream is pushed from the source to all the participating stable nodes. The tree-mesh topology for dissemination of a streaming channel is illustrated in Figure 1(c). The following sub-section explains the protocol for forming the transport overlay.

3.2 Data Structures for the Overlay

All the nodes that receive or forward a channel constitute the transport overlay for that channel. There may be some stable nodes that do not intend to receive the channel but participate in the group as relay nodes. We use the term *member* node to collectively denote the *recipient* and the *relay* nodes in the transport overlay of a channel. Each channel is identified by a globally unique name. We assume the existence of a directory service that returns the IP address of the source node for each channel name.

The tree structure of the inter-clique transport overlay is maintained by the stable nodes in the participating cliques. Each stable node in a clique maintains a table *channelList* that stores information about all the channels being received or relayed by at least one node in the clique. The entry for each channel maps a channel name to a *channelInfo* data structure. There may be a single or several stable nodes in each clique depending on the replication strategy. In case there are multiple stable nodes, a consistent replica of the *channelList* is maintained in each of them. In our design, we decided to use at least two stable nodes per clique, to facilitate failure recovery as discussed later. For each channel, if one of the stable nodes acts as a relay node, the other is maintained as a backup-relay. The replication of the data-structures among the stable nodes adds to the stability of the tree structure. Having multiple stable nodes also facilitates sharing of the relaying load of different channels among the stable nodes. The number of stable nodes in a clique may increase based on the relaying load.

The *channelInfo* contains the meta-data needed to maintain the structure of the transport overlay for the channel. This includes pointers to the neighboring nodes in the transport overlay, both intra-clique and inter-clique. Intra-clique pointers are *relayNode* and *backupRelayNode*, and only IP addresses are stored for them. For inter-clique pointers, both clique-id and IP-address are stored. Inter-clique pointers include *parent* and *backupParent* – the relay node and the backup relay node in the upstream clique, and *childList* – list of relay nodes in the immediate downstream cliques in the transport overlay. To avoid inconsistency, updates to the *channelInfo* is always initiated by the relay node and then propagated to the other stable nodes.

Besides the replicated information of the tree structure maintained by all the stable nodes, the stable node that acts as the relay node in the clique for a given channel, maintains some additional information for the intra-clique part of the transport overlay. This includes a *streamBuffer* that holds a certain number of current segments of the stream that are relayed, and a corresponding bitmap *bufferMap* to identify the segments. The relay node also maintains a *recipientList* that lists IP addresses of all the nodes in the same clique that are receiving or relaying the channel, including the relay node itself. Each node in the clique, regardless of being stable or not, maintains a *streamBuffer*, a *bufferMap* and *partnerList* for every channel it currently receives. *partnerList* is a list of the nodes in the same clique with whom this node is exchanging the stream segments. Altogether, these data structures maintain the mesh-structured transport overlay inside a clique.

3.3 Node Join

When a node connected to the routing substrate wants to join a group to receive a channel, it sends a *join* request to one of the stable nodes in its own clique. Receiving a join request, the stable node first looks up the *channelList* if the requested channel is already there and which stable node is relaying it. If found, the join message is forwarded to that stable node. The relaying stable node maintains a *recipientList* that lists the nodes in the same clique that are receiving the channel. When the relaying stable node receives the request, it adds the requesting node to the list and returns a random subset of the *recipientList* to the requesting node. Receiving the reply, the requesting node can now request those nodes for their current *bufferMap* download stream segments. In turn,

those nodes also know the presence of the new node in *recipientList* and may include it in their *partnerList*.

If the stable node, on receiving the *join* request, does not find the requested channel in its *channelList*, it creates an entry for that channel making itself as the relay node and some other stable node as backup relay node. It then looks-up the address of the source node for the channel from the directory service and sends a *joinRemote* request to the source node to include the stable node as a member of the group. On receiving the *joinRemote*, the source sends an *addNode* message using the routing substrate, towards the clique from which it received the *joinRemote* request. The *addNode* message travels through nodes in several other cliques before reaching the joining clique. The routing substrate routes these inter-clique messages through the stable nodes (Section 2.3). While traveling through the cliques, the *addNode* message creates or extends the tree structured transport overlay and establishes a streaming path from the root to the joining stable node using one stable node in each intermediate clique.

When the *addNode* reaches a stable node of an intermediate clique, the data structures are updated to reflect the changes in the tree; the transport layer is notified by the *forward* method, it looks up its *channelList* table to find the relaying stable node for the particular channel. In case no entry for the channel is found in the *channelList*, the stable node initiates the relay election protocol to elect one of the stable nodes as relay node for the channel. The simplest version of this protocol is to select itself. Alternatively, the protocol may select the stable node with highest available uplink bandwidth, to ensure balancing of the relaying load among the stable nodes. At the same time, a *backupRelayNode* is also selected to complement the relay node. The *addNode* message is then forwarded to the existing or the newly elected relay node for that channel.

When a stable node, being the relay node for the channel in its clique, forwards the *addNode* to another clique towards the destination, it stores its clique id and IP address and IP address of the backup relay node in the message. The relay node in the receiver clique updates the *parent* and *bakupParent* entries for that channel based on this information. The *addNode* message is then forwarded further towards the destination clique. Besides, the relay node also sends an *addNodeAck* message to the parent node. Receiving the *addNodeAck*, the upstream relay node adds the sender of the message to its *childList* table and initiates pushing of the stream to the new child along with others.

When the *addNode* message is finally delivered to the stable node that originated the *joinRemote* request, it updates the *channelInfo* data structure for the channel in a similar manner as above. A response is then sent back to the node that initially sent the *join* request, containing the current *recipientList* and *bufferMap*. The joining node then starts downloading the stream segments from the relay node or other nodes possibly included in the *recipientList*. The message exchange protocol for node join is illustrated in Figure 2(a).

3.4 Graceful Departure of Nodes

A node may leave the transport overlay of a channel or leave the whole system. The underlying routing substrate needs to be updated when a node leaves the system. In case the number of nodes in a clique becomes lower than a threshold, the clique merges with its successor clique. Details of this are discussed in [12]. When a non-stable node leaves the transport overlay of a channel, it sends *leave* message to all its mesh neighbors, including the relaying stable node. The relay node updates the *recipientList* and other neighbors update their neighborhood table. If the number of mesh neighbors becomes lower than a system defined threshold, a node can refresh the neighbor list by asking the relay node for a random list of recipients in the clique.

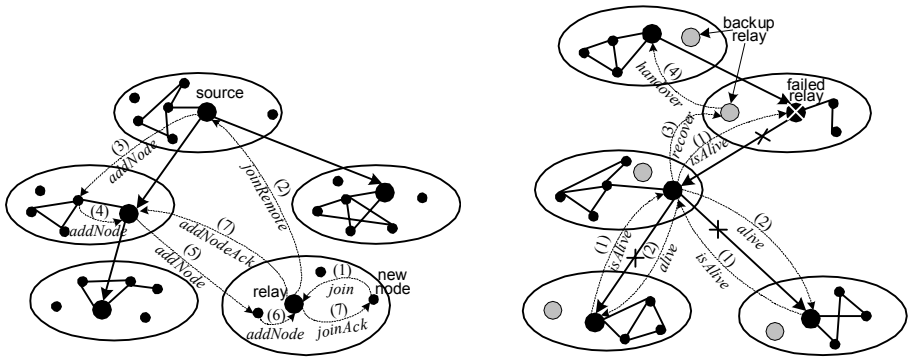
A stable node does not depart from relaying a channel if it is alive and connected to the routing substrate, unless both of its *childList* and *recipientList* are empty. If it wants to leave the CliqueStream platform including the routing substrate, then it initiates a relay election protocol among the other stable nodes in the clique and the stable node with highest available bandwidth is selected. Then the leaving node initiates the *handOver* protocol to transfer the relaying role for the channel it was relaying. The parent node is notified of the new relay node and the *channelInfo* for the particular channel is updated in all the stable nodes in the clique to reflect the assumption of new relay node. The departing stable node also initiates a stable node election protocol concurrently. The node departs after initiating the *handOver*.

3.5 Failure of Nodes and Reconstruction of Delivery Trees

Apart from graceful departure, nodes may suddenly depart or crash. Here we describe how node failures are detected and how the streaming tree is reconstructed.

The failure of a non-stable node is detected by its mesh neighbors and their neighbor list is replenished by finding new neighbors, in the same way as in graceful departure.

When a stable node fails, all the downstream stable nodes, in the streaming tree for each of the channels the stable node was relaying, stop receiving the stream. After passing a small threshold of stoppage time, all of them will react to recover from the failure of the upstream relay node. However, the protocol we devised quickly resolves which relay node actually failed and then transfers its responsibility to the back-up relay node in the same clique. Failure of the relay node is also detected by the backup relay node as



(a) Node join (b) Recovery from stable node failure

Fig. 2 Message exchange during node join and failure recovery

the stable nodes in a clique periodically exchange heartbeat messages.

Any stable node, detecting the stoppage of receiving the pushed stream to itself, checks whether its parent is still alive by sending an *isAlive* message to the parent and waits for an *alive* message as reply. In case the reply timeouts, it sends a *recoverTree* message to the node designated as *backupParent* to take over. On receiving the *recoverTree* message or detecting the failure of the relay node through heartbeat timeout, the *backupRelayNode* initiates a recovery of the link. It retains a replica of the *channelInfo* data structure, and it knows the parent node of the failed relay node. A *handOver* message is sent to that parent to consider the backup node as a child instead of the failed node. Thus the failure is recovered completely locally. A new backup relay node is also designated at this time. The failure recovery procedure is illustrated in Figure 2(b).

In the very unlikely event when both the relay node and backup relay node fail concurrently, the tree will not be recovered and the node that sent the *recoverTree* message to the backup parent will not receive any stream data. Passing a certain amount of time without receiving any stream data after receiving the *alive* message from the parent or after sending the *recoverTree* message, the downstream nodes will realize that both relay and the backup relay failed in some upstream node. All of these downstream relay nodes will join the streaming group independently using the join procedure.

3.6 Split and Merge of Cliques

As described in Section 2, when arrival and departure of nodes in the routing overlay make a clique too large or too small, the clique splits into two or merges with its successor clique. In addition to the routing table updates done by the routing substrate, the tree structure of the transport overlay may also need to be updated during split and merge.

When a clique merges with its successor, we denote the former as merging clique and the latter as merged clique. The new clique after merger retains the id of the merging clique and the id of the merged clique vanishes. The stable nodes of the previously individual cliques maintain their stable status for a while. They update their *channelInfo* data by merging the data from the merging and the merged cliques including all channels relayed by one or the other or both of these cliques. In case two relay nodes are found for the same channel, the one that earlier belonged to the merging clique prevails and the children from the relay in the merged clique are transferred to that node.

When overpopulated, a clique splits into two, and one of them retains the previous clique's id. Let us denote this clique as primary, and the other clique as offspring. The stable nodes of the previous clique remain as stable nodes in the new cliques and they belong to either the primary or the offspring clique according to the proximity rules of splitting. The channels relayed by the stable nodes belonging to the offspring clique may need to be handed over to the stable nodes in the primary clique to make the streaming tree consistent with the routing tables. This is needed only if the channel has a non-empty *childList*. In case there are some recipient nodes in the offspring clique for that channel, the stable node re-joins the channel using the new clique id before performing the hand-over. In case a channel relayed by a stable node in the primary clique has some recipient now belonging to the offspring clique, they are requested to re-join the channel. This will result in a stable node in the offspring clique to become a relay node for that channel. Note that new stable nodes are recruited in both the primary and the offspring clique as necessary to accommodate the channels. At the beginning, the stable nodes in the offspring clique are underloaded. However, they soon get new relay loads when new join messages are routed through them.

4 Analysis of System Features

In this section we discuss the notable features of the CliqueStream platform. The main argument of the paper is that clique-based overlays allow creation of a streaming topology with good locality properties compared to other approaches. The CliqueStream approach also allows fast and localized recovery mechanism in presence of node departures. The following sub-sections discuss each of these features in details.

4.1 Locality

The locality property in the overlay network is achieved when overlay neighbors are in close proximity in the physical network. There are twofold benefits of forming a locality-aware overlay – first, the stretch of the streaming path from the source to the recipient nodes is minimized, and second, a significant portion of the streaming paths from the source to each individual recipient are shared. To demonstrate these two aspects of locality, we performed some simulation experiments.

For the simulation model, we assumed that the nodes can be laid out in a 2-dimensional Euclidean space based on some proximity metric, such as network latency. We also assume that the nodes are uniformly distributed in the 2-dimensional space. While the uniform distribution does not accurately reflect the node distribution in large networks like the Internet, several works have shown that nodes in the Internet can be mapped on an Euclidean space with good accuracy [6].

First, we tried to demonstrate that, if a message is routed from a source node to two different destination nodes, the fraction of the path that is common to both routing paths is correlated to the distance between the two destinations. This implies, when two nodes are close enough in the Euclidean space, a large portion of the paths from the source to the two nodes are shared. We measure the commonality of the two paths using a convergence metric used in [1]. If d_c is the length of the common path and d_1 and d_2 are the lengths of the paths from the diverging point to the two nodes, then the convergence metric $C = (\frac{d_c}{d_c+d_1} + \frac{d_c}{d_c+d_2})/2$. C has a value 0 when the two paths are completely disjoint and 1 when they are completely shared.

We created an eQuus overlay of 100000 nodes, where the minimum and maximum clique size parameters were set to be 32 and 128, respectively. The length of the id was 64 bits. The parameter b that defines how many bits of the id are matched in each routing hop was set to 2. Note that the maximum fan-out of a streaming tree in CliqueStream is 2^b. We chose 100 random nodes as source, and for each source we chose 100 random pairs of destination nodes. The convergence metric is then computed for each pair of paths. In the simulation runs, we placed the nodes on an arbitrarily chosen 3500×3500 2-d plane. The length (or cost) of an overlay link between two nodes, which is used for computing the convergence factor and network load (defined later), is computed as the Euclidean distance between the two nodes on the given plane. Figure 3 plots the average convergence metric against the distance between two destinations. This clearly shows the correlation of convergence to the distance between the pair of destinations.

In the next set of experiments, we evaluated the properties of the streaming tree created over the routing substrate. We constructed a routing substrate with 50000 nodes and constructed a streaming tree using a random subset of nodes. For comparison, a random tree was created with the same set of nodes joining the tree in the same order. Each newly joining node randomly chooses one of the existing tree-nodes as its parent. To evaluate the stretch of the source-to-recipient streaming paths we used the ratio of the length of the routing path to the length of the shortest possible source-to-recipient path (which is the Euclidean distance). The average stretch for source-to-node paths is computed for various sizes of transport overlays. To evaluate the load on the network due to redundant data transmission paths, we used a network load metric that counts the total length of paths traveled by a message (and its replicas) to disseminate the message from the source node to all the recipients. For comparison across different sizes of transport overlays, the metric is normalized by dividing it by the number of nodes in the transport overlay.

We considered two other types of optimally constructed trees for comparison– one that has minimal average stretch for the source-to-destination paths, and the other that has minimal network load. The optimal stretch tree is constructed by connecting the newly joining node as close as possible to the root, subject to the maximum fan-out constraint (which is the same as in CliqueStream). The optimal load tree is constructed by connecting each newly joining node to the node that is closest to the new node.

Figure 4 compares the average stretch of the source-to-recipient paths for different tree construction protocols. The CliqueStream trees have significantly lower stretch than random trees and are pretty close to the optimal stretch trees. In fact, the stretch of a CliqueStream tree is defined by the stretch of the lookup paths in the routing substrate, which is bounded by the logarithm of the total number of nodes in the substrate.

Figure 5 compares the network load per member metric for different tree construction protocols. It shows that the network load per node in CliqueStream is significantly lower than that in the random tree. The network load of CliqueStream is also lower than that of the optimal stretch tree and pretty close to that of the optimal load tree. Another observation is that the network load per node actually de-

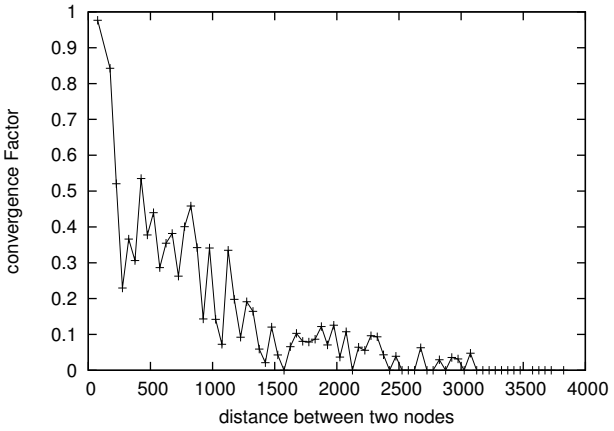


Fig. 3 Convergence of streaming route

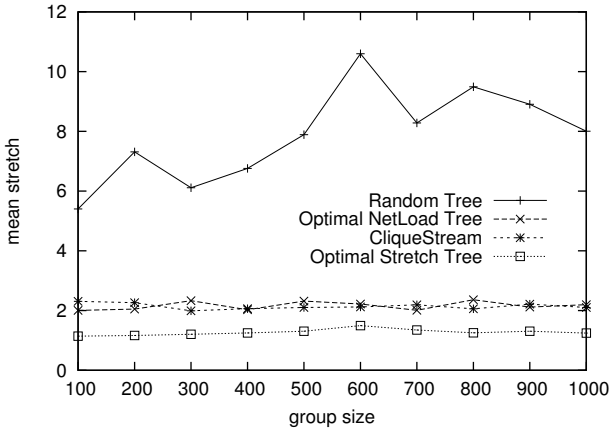


Fig. 4 Stretch of the streaming path in different tree construction protocols

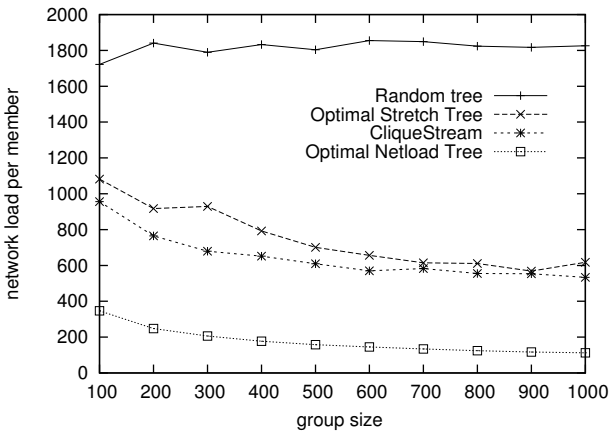


Fig. 5 Network load per member in different tree construction protocols

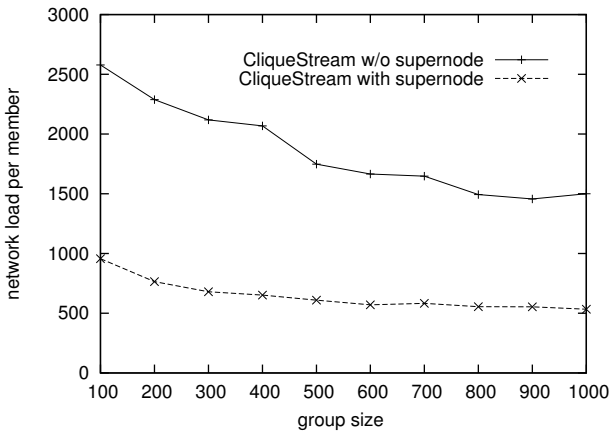
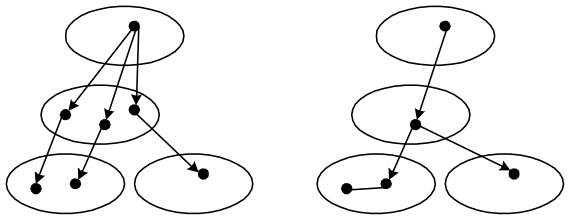


Fig. 6 Use of stable nodes reduces the network load

creases when more nodes are added in the tree. This implies better scalability of the CliqueStream platform.

The benefits of using stable nodes in CliqueStream is evaluated in Figure 6. The main argument behind using stable nodes is that it eliminates redundant streaming paths and thus reduces the network load. This effect is demonstrated in Figure 6, where the tree in CliqueStream with stable nodes causes less network load irrespective of the size of the group. If stable nodes are not considered and a stream is forwarded along the eQuus routing paths from source to individual nodes, there may be multiple overlay links carrying the traffic between the nodes between the same pair of cliques, as illustrated in Figure 7(a) and 7(b).



(a) No stable node (b) With stable node

Fig. 7 Stable nodes eliminate redundant paths

The use of stable nodes, however, causes some of these nodes to act as relay node even if the node itself is not receiving the particular channel. This may result in unnecessary relay load for the stable nodes. The worst case scenario occurs when each member of every clique is recipient of a different channel. In this case, if there is only one relay node per clique, each relay node has to relay all the channels to 2^b downstream relay nodes. On the other extreme, the maximum benefit of aggregation of streaming paths in the stable nodes can be achieved for very popular channels and when the popularity of different channels are concentrated in different network proximities. To avoid the worst case scenario, CliqueStream recruits more stable nodes in a clique when the relay load exceeds the capacity of the existing stable nodes. The number of stable nodes in a clique is bounded only by the total number of nodes in the clique. In any case, the relay load on a stable node for a single channel is bounded by a constant 2^b.

4.2 Startup Delay

The startup delay is an important metric of user perceived performance in a video streaming system. It is defined as the time elapsed from the time when the user expresses her intention to receive a channel of video to the time the stream starts on the display. Factors that contribute to this startup delay are the time required to connect the node to the system (connection time), and the time required to activate the reception of the channel (channel switch time).The latter includes time for signaling, for the transmission of the video frames and the playout buffering time for accumulating sufficient stream packets in the local buffer before starting the playback. The user experiences the cumulative delay for both of the joining times when she turns on her node, (e.g. the set-top box). However, when the user switches to a new channel while having her node turned on, only the channel switching time is experienced.

The connection time is the time required for joining the routing substrate. It is equivalent to the node join time in eQuus, which is the time required for $2\log_{2^b}(N/c)$ round trip messages [12]. Here N is the total number of nodes in the routing overlay and c is the number of nodes per clique.

The channel switch time is the time required to join the transport overlay of a channel. A node either needs to create a mesh connection with the nodes in the same clique that are already in that transport overlay, or ask the stable node in the clique to join the streaming tree of the overlay. In the first case, three round-trip message exchanges are required – one round to get a list of in-clique nodes that are receiving the channel, and one round to establish the partnership with them. The stream packets are pulled from the partners in the third round. In the second case, $3 + 2\log_{2^b}(N/c)$ one-way messages are exchanged before the first packet of the stream arrives at the joining node. This shows that the channel switching time of the user is at most one third of the time needed for the initial startup.

The buffering time depends on the system defined playback buffer size, which in turn is based on the jitter experienced in the stream. Our overlay design does not affect the stream jitter, hence we leave the buffer size to be determined empirically.

4.3 Playback Latency

Playback latency, another important metric for user perceived performance in video streaming systems, is defined as the time lag between the occurrence of a live event at the source and the playback of the event at the node where the stream is being watched. Transmission time – the time needed for a stream packet to travel from the source to the playback node, and buffering time are the contributors to the playback latency.

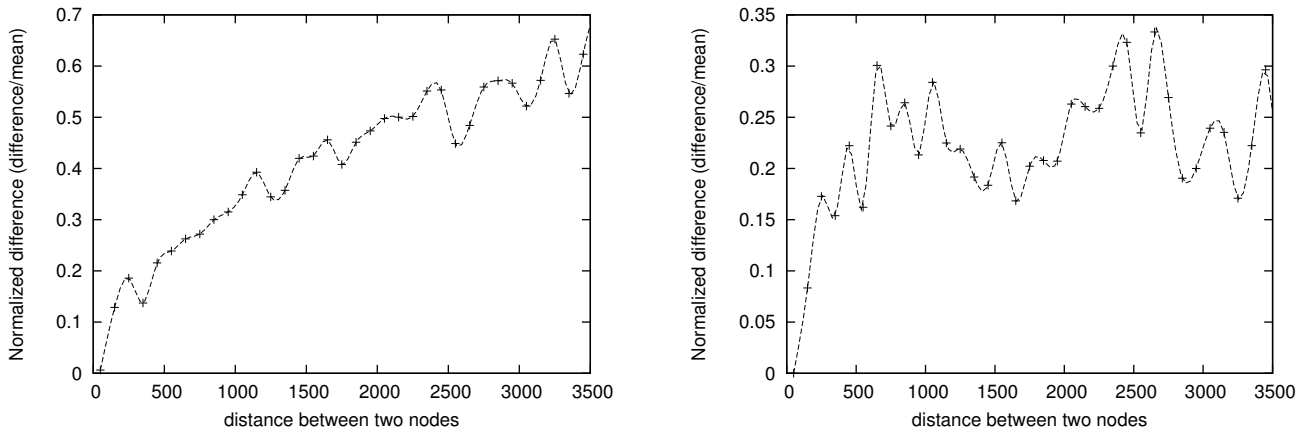
The transmission time has two parts – the time for a packet being pushed through the tree and the time for pulling a packet in the mesh inside a clique. In each part, the transmission time equals the number of overlay hops times the forwarding time at each hop, plus the sum of the latencies of the hops in the underlying network. Number of overlay hops in the tree is $\log_{2^b}(N/c)$ in the worst case. The total underlay latency equals the stretch times the distance between the source and the playback node in the latency space. As described in eQuus [12], the stretch is bounded by a constant and independent of N. Inside a clique, the number of hops within the mesh is logarithmic to the clique size, because each packet is distributed along an induced tree. Altogether, the total transmission time is $O(\log N)$.

An interesting feature of the CliqueStream platform is that the playback latency is highly correlated to the locality, i.e. the latency at two nearby nodes are similar. This follows from the fact that nodes in the same clique receive the stream almost along the same path, and the streaming path for nodes in nearby cliques are also largely shared. This effect is further demonstrated by a simulation study. Figure 8(a) shows the normalized difference in the total latency of the overlay hops in the inter-clique part of the transport overlay, for pairs of receiver nodes at different distances in the latency space. Figure 8(b) shows the corresponding difference in number of hops from the source. The difference in total latency clearly shows the correlation with the distance between the receiver-pair. The difference in number of hops does not grow much for distant pairs of recievers, possibly because the maximum number of overlay hops for the given network size was reached.

4.4 Failure Recovery and Availability

Improved fault tolerance of CliqueStream results from two facts. On the one hand, relatively more stable and higher capacity nodes are placed as internal nodes of the tree and less stable nodes are the leaves of the tree. Thus the effect of failure of non-stable nodes are localized inside the cliques. The use of receiver-driven pulling on a mesh-like topology inside the clique makes it further adaptive to the node dynamics. On the other hand, the clustered topology allows multiple stable nodes in a single clique, which in turn allows maintaining a backup relay node for each channel. The use of backup relay nodes facilitates fast and localized recovery from failure of stable nodes.

We can determine the outage time due to a failure in terms of the average round trip delay between nodes in adjacent cliques (*RTT*). The outage time for a failure is the sum of the failure detection time and the repair time. As described in Section 3.5, the failure of a relay node is detected either by the relay node in the downstream clique or by the



(a) Difference in total latency from source

(b) Difference in number of hops from source

Fig. 8 Difference in playback latency increases with distance between two receiver nodes

backup relay node in the same clique. In the first case, it requires the downstream relay node to pass a timeout period of stream outage, to send an *isAlive* message to the parent and wait for another timeout period before it either receives an *alive* reply from the parent or detects the failure. A reasonable period for both the timeouts is 2RTT, hence the total detection time is 4RTT. It requires one more message transfer to the backup of the failed relay (the *recover* message), so the the backup relay is aware of the failure after 4.5RTT. In fact, the time before the backup relay node becomes aware of the failure is the minimum of 4.5RTT and the heartbeat message interval. The recovery time is $\frac{1}{2}$RTT for a backup relay node to notify the parent plus $2 \times \frac{1}{2} = 1$ RTT to send the stream from the parent to the failure detecting child through the backup relay. So, in total, failure detection and recovery time $t_{fr} = 6$RTT. For a 50 ms RTT, this amounts to 300ms only. The parent of the failed relay node is unaware of the failure until it receives the *handover* message from the backup relay node. So the video segments streamed before the end of the failure-recovery period will be lost.

CliqueStream is also quite efficient in terms of control message overhead induced by each stable node failure. Instead of every downstream node rejoining the tree, only the immediate children initiate the recovery process and only a one step recovery process is conducted by the backup relay node. Each of the downstream nodes exchanges a (*isAlive*, *alive*) message pair except the immediate children of the failed relay node. The recovery process takes only 2 messages from the backup relay node. Each of the immediate children may send 1 request to the backup parent to initiate the recovery process.

Now, let us try to determine what fraction of the stream is lost on average due to failures during a streaming session. Say, the depth of the streaming tree is k. For the nodes in the clique that is farthest from the source in the streaming tree, the availability of the stream is $A = \Pi_{i=1}^{k} A_i$, where A_i is the availability of the relay node in the clique i hops away from the source. We may model the life of a relay node as a birth-death Markov process. If we assume that the failure of a stable node is a Poisson process with rate λ, then the expected time before the next failure is $\frac{1}{\lambda}$. If the time needed for recovery from a failure is $\frac{1}{\mu}$, then the availability of a relay node is $\frac{\mu}{\lambda+\mu}$, ignoring the possibility of both the relay node and the backup relay node failing at the same time.

From the previous analysis, we got the average recovery time $\frac{1}{\mu} = 6RTT$. Therefore $A = (\frac{\mu}{\lambda+\mu})^k$. From the property of the routing substrate, the worst case value of k is $log_{2^b} N/c$. For example, in a CliqueStream system with 10^5 nodes, organized in 1000 cliques of 100 nodes each on average, with a fanout from a streaming node of 4, with an average life time of a stable node of 4 hours ($\lambda = \frac{1}{4*3600}$) and an average RTT between two overlay nodes of 250ms ($\mu = \frac{1000}{6*250} = \frac{2}{3}$), we obtain the availability of $(0.999896)^{4.9829} = 0.9995$. This means that the stream is available at the receiver at 99.95% of the session time. This amounts to 31ms of outage time in a 1hour streaming session.

The outage time due to the failure of a non-stable node in the mesh inside a clique depends on several factors, such as the size of the data-exchange buffer maintained by each node, the time interval between the exchange of buffer-maps and the number of mesh neighbors. Previous works have studied the trade-off between the improvement in packet loss by increasing the buffer size and the corresponding overhead of buffer-map exchange [9]. In CliqueStream, the existence of the stable relay node in each clique allows the non-stable nodes to retrieve time-critical stream packets from the relay node, in case none of its mesh neighbors can provide those packets, thus reducing the outage time due to non-stable node failures.

5 Related Work

There are quite a few approaches for streaming video over P2P overlays both in industry and in academia. Widely used commercial implementations, such as PPLive [8] and UUSee [20], use receiver-driven content pulling over unstructured mesh overlays with random neighborhood, which are variants of the CoolStreaming [21] protocol. The inefficiency of these platforms in terms of huge long-haul traffic burden is explained in [8].

There have been several efforts to create peer-to-peer overlays that select the overlay neighbors based on locality characteristics of the underlying physical network. CAN [13] has applied a landmark-based binning approach to assign d-dimensional coordinates to each node and routing is performed based on the proximity of the nodes in the coordinate space. Zigzag [16] is another architecture that organizes the nodes into locality-based clusters. It creates a hierarchy of clusters, grouping leaders of lower level clusters into higher level clusters and streaming the media content through this hierarchy.

Among the video streaming or group multicast topologies on structured peer-to-peer overlays, Scribe [3] is a prominent one. Scribe creates the multicast tree based on the reverse path of message routing in the Pastry [14] routing substrate. However, since Pastry assigns random ids to the nodes, the routing path is likely to have random hops between locally uncorrelated nodes. Some form of locality is however achieved by careful selection of routing table entries. In CliqueStream, the multicast streaming tree is constructed based on the forward paths of the messages from the source to the receivers on the clustered and structured peer-to-peer overlay, named eQuus. Because node id assignment in eQuus is strongly correlated with locality, the routing paths are more directionally controlled and have predictable locality properties.

In general P2P streaming platforms apply either content pushing over multicast trees or receiver driven content pulling over a mesh overlay. mTreeBone [19] has proposed a hybrid approach where more stable nodes constitute the internal nodes of the tree and more dynamic nodes are placed at the leaf level. The leaf nodes also participate in a mesh, and the stream is delivered using a combination of pushing and pulling. Our approach of using stable nodes to construct the tree backbone is similar to mTreeBone. However, the locality-based clustering was not considered in mTreeBone.

Making the overlay localized runs the risk of partitioning the network. In DAGStream [10] a DAG of nodes is created instead of a tree and content is delivered by receiver-driven pulling. Presence of multiple parents allow the system to work in the presence of node failure or departure. AnySee [11] maintains a set of backup paths for each active path over which it streams the data. When a stream in the active path is disrupted, one of the backup paths is selected and assumed as active path. However, switching the whole path takes much longer time than switching a single hop, as done in CliqueStream.

Zigzag [16] maintains a head and an associate head for each cluster. An associate-head receives the stream from the head of a foreign cluster and disseminates it inside the cluster. The head controls the resources within a cluster and can quickly elect a new associate-head in case the current one fails. Failure of the head is tolerated by selecting an alternative foreign head by the downstream associate-head. Unlike hierarchical clustering in Zigzag, CliqueStream creates disjoint clusters of nodes at the same level. CliqueStream maintains a backup relay node for each relaying stable node. The recovery procedure is initiated by the backup node of the same clique and it is contained locally.

6 Conclusion

In this paper we have exploited the features of a clustered distributed hash table overlay to create a network efficient transport overlay for video streaming. Our analysis shows that the clustered topology of the transport overlay provides good locality properties such as low stretch and low communication load compared to random topologies commonly used in existing peer-to-peer streaming systems. Also, we have introduced fast and localized failure recovery mechanisms to make the streaming platform robust against node dynamics. Relatively more stable nodes are used as internal nodes of the streaming tree so that their failure probability is minimal. Moreover, backup relay nodes are used to allow fast recovery. The localized clustering of the nodes allows an efficient election mechanism for the relay nodes and backup relay nodes.

To avoid the small disruptions in the streams that occur due to failure of tree nodes, use of multiple description coding and streaming different descriptions over different trees may be a good solution. However, the question how multiple node-disjoint trees can be constructed in the clustered peer-to-peer overlay, remains an open problem to be solved.

Acknowledgements We are grateful to Dr. Guy-Vincent Jourdan of University of Ottawa and the anonymous referees and the audience of P2P'08 conference for their valuable comments that helped significantly to refine the design of the CliqueStream system and to improve the presentation of the paper.

References

1. M Castro, P Druschel, Y Hu, and A Rowstron. Exploiting Network Proximity in Peer-to-peer Networks. Technical Report MSR-TR-2002-82, Microsoft Research, 2002.

2. M Castro, P Druschel, A Kermarrec, A Nandi, A Rowstron, and A Singh. SplitStream: High-bandwidth Multicast in Cooperative Environments. In *19th ACM Symp. on Operating Systems Principles (SOSP)*, pages 298–313, 2003.
3. M Castro, P Druschel, A M Kermarrec, and A Rowstron. Scribe: a Large-scale and Decentralized Application-level Multicast Infrastructure. *IEEE J. Selected Areas in Communications*, 20:1489–1499, 2002.
4. M Castro, M B Jones, A M Kermarrec, A Rowstron, M Theimer, H Wang, and A Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *IEEE INFOCOM*, Apr. 2003.
5. M Chay, P Rodriguez, S Moony, and J Crowcroft. On Next-Generation Telco-Managed P2P TV Architectures. In *7th IPTPS*, Feb. 2008.
6. F Dabek, R Cox, F Kaashoek, and R Morris. Vivaldi: a Decentralized Network Coordinate System. In *ACM SIGCOMM '04*, pages 15–26, 2004.
7. J Dilley, B Maggs, J Parikh, H Prokop, R Sitaraman, and B Weihl. Globally Distributed Content Delivery. *IEEE Internet Computing*, 6(5):50–58, 2002.
8. X Hei, C Liang, J Liang, Y Liu, and K W ross. A Measurement Study of a Large-Scale P2P IPTV System. *IEEE Tans. Multimedia*, 9(8):1672–1687, Dec. 2007.
9. B Li, S Xie, Y Qu, G Y Keung, C Lin, J Liu, and X Zhang. Inside the New Coolstreaming: Principles, Measurements and Performance Implications. In *The 27th IEEE INFOCOM*, pages 1031–1039, Apr. 2008.
10. J Liang and K Nahrstedt. DagStream: Locality Aware and Failure Resilient Peer-to-peer Streaming. In *Intl. Conf. on Multimedia Computing and Networking (MMCN)*, 2006.
11. X Liao, H Jin, Y Liu, L M Ni, and D Deng. AnySee: Peer-to-Peer Live Streaming. In *IEEE INFOCOM*, Apr. 2006.
12. T Locher, S Schmid, and R Wattenhofer. equus: A provably robust and locality-aware peer-to-peer system. *Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on*, pages 3–11, 06-08 Sept. 2006.
13. S Ratnasamy, P Francis, M Handley, R Karp, and S Shenker. A scalable content-addressable networks. In *ACM SIGCOMM-01*, pages 161–172, Aug. 2001.
14. A Rowstron and P Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *IFIP/ACM Intl. Conf. on Distributed Systems Platforms*, pages 329–350, 2001.
15. T Silverston and O Fourmaux. P2P IPTV Measurement: A Case Study of TVAnts. In *2nd Conference on Future Networking Technologies (CoNEXT)*, Dec. 2006.
16. D A Tran, K A Hua, and T T Do. A Peer-to-peer Architecture for Media Streaming. *IEEE J. Selected Area in Communications*, 22(1):121–133, 2004.
17. V Venkataraman, P Francisy, and J Calandrinoz. Chunkyspread: Multitree Unstructured PeertoPeer Multicast. In *6th IPTPS*, 2006.
18. F Wang, J Liu, and Y. Xiong. Stable Peers: Existence, Importance, and Application in Peer-to-Peer Live Video Streaming. In *IEEE INFOCOM*, Apr. 2008.
19. F Wang, Y Q Xiong, and J C Liu. mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast. In *27th IEEE ICDCS*, page 49, 2007.
20. C Wu, B Li, and S Zhao. Magellan: Charting the Large-Scale Peer-to-Peer Live Streaming Topologies. In *IEEE ICDCS*, Jun. 2007.
21. X Zhang, J Liu, B Li, and Y Yum. CoolStreaming/DONet: a Data-driven Overlay Network for Peer-to-peer Live Media Streaming. In *The 24th IEEE INFOCOM*, Mar. 2005.