Signal Processing: *Image Communication* 27 (2012) 113–125

Contents lists available at SciVerse ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

Resilient peer-to-peer streaming of scalable video over hierarchical multicast trees with backup parent pools[☆]

Muge Fesci-Sayit[a,*], E. Turhan Tunali[b], A. Murat Tekalp[c,1]

[a] *Int'l Computer Institute, Ege University, Bornova, Izmir 35100, Turkey*
[b] *Faculty of Engineering and Computer Sciences, Izmir University of Economics, Balcova, Izmir 35330, Turkey*
[c] *College of Engineering, Koc University, Sariyer, Istanbul 34450, Turkey*

ARTICLE INFO

Article history:
Received 10 August 2011
Accepted 23 November 2011
Available online 2 December 2011

Keywords:
Peer-to-peer streaming
Multicast trees
Backup parent pools
Scalable video coding

ABSTRACT

This paper introduces a new cross layer tree-based peer-to-peer design using hierarchical cluster layers and a new method for selection of "backup parent pools" for resilient streaming of scalable video to provide highest quality of experience for all peers. Backup parent pools are selected during the process of multicast tree construction based on information provided by the hierarchical clusters. The proposed tree construction method aims to minimize bottlenecks that may be caused by non-leaf nodes with low upload bandwidth. Performance of the proposed system is demonstrated by extensive test results using a wide range of simulation scenarios. Comparison of the results with those of some recent works indicates that the proposed system is clearly superior in several aspects.

1. Introduction

Various peer-to-peer (P2P) video streaming solutions have been proposed by both academia and industry in order to meet the growing demand for streaming video services over the Internet. However, providing QoS (Quality of Service) is still highly challenging in P2P streaming because of peer churn and network congestion. Furthermore, false network topology information may exacerbate the situation. Different types of overlay network structures are proposed in the literature to sense network topology [1]. Two main P2P overlay system architectures are mesh-based and tree-based systems.

Mesh based systems achieve better performance under different failure scenarios when compared to tree based systems. To solve the scalability problem, clusters are introduced in mesh-based systems. However, initial waiting time and playback lag between nodes sending data to each other is still a major problem [2,3]. If nodes are placed into a multicast tree by minimizing bottlenecked links on the tree and by constructing parent–child relation between nearby nodes in terms of delay, tree-based systems have lower initial wait time, lower playback lag between parent and children, lower complexity and lower coordination and communication overhead, but resilience becomes a major issue. For a tree-based P2P overlay system; the most challenging part of the structure is to cope with churn during streaming session. There can also be node crash or link failure. Throughout the paper, we use the term "node failure" to represent "churn", crashing node or link failure. In order to provide seamless streaming and make tree resilient against node failure, different types of mechanisms are proposed in the literature. Resilient multicast techniques can be classified as reactive and proactive [4]. In the reactive approach, node

[☆] This work is supported by the Scientific and Technical Research Council of Turkey (TUBITAK), Project number: 108M411.
* Corresponding author.
E-mail addresses: muge.fesci@ege.edu.tr (M. Fesci-Sayit), turhan.tunali@ieu.edu.tr (E.T. Tunali), mtekalp@ku.edu.tr (A.M. Tekalp).
[1] A. Murat Tekalp acknowledges support from Turkish Academy of Sciences (TUBA).

0923-5965/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.image.2011.11.004

experiencing parent failure tries to find a new parent after parent failure detection. In the proactive approach, before parent failure, each node has a predefined backup plan to minimize the effect of the loss of its current parent.

NEMO [5] and NICE [6] systems provide resilience by repairing the tree after failure, hence they are examples of systems using reactive approach. These works are also examples of peer to peer systems running over a hierarchically clustered network. The multicast tree is constructed from the leaders of clusters to convey video data to their respective clusters. After the failure of any leader, application layer protocol must choose another leader to continue video streaming. Since tree repair is based on selection of a new leader after failure, this approach is reactive. Unfortunately, these works suffer from increased delay due to tree construction. Furthermore, if more children are connected to the cluster leader than the capacity of leader allows, nodes in that cluster cannot receive the number of necessary data packets to play the video properly.

Both multiple description coding (MDC) and scalable video coding (SVC) have been proposed to provide robustness and quality adaptation for tree and mesh based P2P systems [7–11]. MDC can be considered as a proactive approach [12,13]. A node participating in more than one tree can still continue to receive video packets from another tree even if its parent fails in one tree. Even though MDC may provide more robustness in the event of node failures, it has significantly higher redundancy compared to SVC. For systems using a non-scalable standard codec, adding extra links to the multicast tree [5,14] is another approach. However, in such a case, optimal bandwidth utilization is lost due to redundant packets [15]. [14] utilizes another redundancy in which a distributed randomized forwarding algorithm forwards data packets randomly to a subset of the multicast tree. [4,16,17] also use a proactive method and implement a failure recovery algorithm for nodes affected by a parent failure and assign a backup node to each node. However, they do not address the problem of scalability. [18] proposes using backup parents in mesh networks in order to provide seamless streaming. For tree overlay multicasting, placing nodes with high upload capacity close to the root yields better performance in terms of overall bandwidth utilization and received video quality [19]. Both proactive and reactive techniques have some disadvantages. For tree repair after failure, reactive techniques require more time than that of proactive techniques. On the other hand, proactive techniques are not in general adaptive, which is an essential requirement for fast tree recovery.

In this paper, we propose a new cross-layer design for a P2P video streaming system, which utilizes scalable video coding and constructs multicast trees with backup parent pools over hierarchical clusters. While hierarchical clusters provide service scalability, each part of the multicast tree is formed in each cluster for all hierarchical layers in a distributed manner. Capacity aware multicast trees are formed according to rates of scalable video layers. To construct resilient multicast tree, we propose a new type of resilience technique, which has both reactive and proactive components and thus provide seamless video streaming session despite of churn. Our design is distinct in many aspects from the ones streaming over multicast trees in literature: (i) Instead of prefixed single backup parent, our approach uses a backup parent pool set for every level of the tree, which forms the proactive part of our system. (ii) Backup-out-degree of each backup parent is monitored dynamically. In case a backup parent unexpectedly leaves during streaming, this can be detected by heartbeat messages leading to increased probability of finding a suitable parent in the first attempt. (iii) The reactive part of our system selects the backup parent after parent failure according to tree level, which they belong to. This guarantees a responding backup parent and receiving video at maximum possible rate. (iv) By allocating the nodes of higher out-degree to upper layers of multicast tree, our approach guarantees the existence of such back-up parent sets.

Forming backup parent pools that contain more than one node and making this selection by considering the bandwidth distribution of the nodes is the proactive component of the proposed system. Our approach guarantees that no bandwidth is tied up for a particular node. During a streaming session, nodes experiencing parent failure try to find a new parent from their backup parent pool, which is the reactive component of the proposed system. Since backup parent pool is constructed from nearby nodes in advance and the number of nodes in the list is limited, finding new parent procedure may be completed in a short time. Combination of proactive and reactive feature makes our work distinct from others in the literature.

The paper is organized as follows: In Section 2, we present the overlay architecture consisting of the control protocol and streaming protocol, where the control protocol addresses hierarchical clustering of peers and the streaming protocol deals with multicast tree construction and streaming. In Section 3, we introduce backup parent pool selection and highlight streaming essentials using the backup parents. Section 4 discusses the performance of proposed P2P system by comparing it with an existing system from the literature. Finally, we provide some concluding remarks in Section 5.

2. Hierarchical overlay architecture

The proposed hierarchical overlay architecture consists of a control protocol and a streaming protocol, which are described below.

2.1. Control protocol

In order to provide a scalable tree-based P2P solution, peers are organized into hierarchical clusters [5,14,20,21]. Clusters are organized in hierarchical manner with layer 0 being the bottom layer. The main purpose of clustering is to group “closer” nodes together. Hierarchical clusters are formed using the delay metric. Lowest level clusters are formed of peers that have low communication delay. In each cluster, one node is chosen as cluster control leader that is responsible for cluster maintenance, i.e., node join/leave procedures, and establishing communication links between the nodes in the same cluster and the nodes in other clusters. The leader selection is done by considering the distance of the nodes to the other nodes in the same cluster. For fault tolerance, one node is also chosen as back-up leader in case the leader crashes. Back-up leader copies all necessary information from the leader. Control leaders of level $l-1$ clusters form a level l cluster. The essential rule followed in the design is to use exactly the same algorithm in clusters of different hierarchical layers. This provides simplicity and efficiency.

The formation of the system is initiated by declaring a Rendezvous Point (RP) to candidate nodes. Any node wishing to join the system sends a request to RP to obtain the addresses of the top layer cluster control leaders. After finding the closest (in delay) leader, the joining node sends a request message to this closest leader to obtain the addresses of the leaders of the respective lower layer. If distance is not close enough, a node can declare itself as a new cluster leader. Applying node travels among hierarchical layers until it reaches the bottom layer and joins a cluster in $\log_k N$ steps, where k is the number of nodes in a cluster and N is the total number of nodes in the system. k varies from m to $3m-1$ gradually as in [5,6]. RP monitors the number of hierarchical layers and as this depth in hierarchy increases, RP increases m. Node demanding a particular video sends a request message to control leader. Control leader collects video requests from its cluster for a certain time unit and multicast tree construction starts after video is searched and found in the system.

2.2. Streaming protocol

Streaming protocol deals with: (i) constructing multicast trees for a particular video streaming, (ii) dynamic tree maintenance during streaming and (iii) sending and receiving video packets to and from the related set of nodes. In order to save the control leader from the burden of streaming, we also introduce a streaming leader for each cluster that may be different than control leaders. The multicast trees form a higher level of abstraction on top of hierarchical clusters and it is exactly between these two levels where we introduce cross-layer design as far as streaming and backup are concerned.

2.2.1. Selection of streaming leaders

Streaming leaders are assigned to be root of a tree in one or more clusters, and one or more children are connected to streaming leader in each hierarchical layer. Therefore the capacity of streaming leader must be high enough to support the number of children that are connected to it in each layer. Note that a streaming leader is not necessarily a control leader of a particular cluster.

The selection process of streaming leaders starts in the clusters that have either requester or source nodes (or both) at the bottom hierarchical layer. Each requester and source node within a cluster at the lowest hierarchical layer calculates the distance between itself and all the other requester nodes by measuring the round trip time. This information will be used in children selection when multicast tree is formed. They also send their upload bandwidth value to the leader of their cluster. Cluster leader chooses the source node with the highest capacity if any such source exists or chooses the requester node with the highest capacity as streaming leader and sends this information to the upper hierarchical layer. This process continues until reaching the topmost hierarchical layer that is involved in search of the video. The following theorem a proof of which is given in the appendix establishes message complexity of forming streaming leaders.

Theorem 1. *In the worst-case, message complexity of selection of streaming leaders equals $O(kN)$, where k is cluster size and N is the total number of nodes in the system.*

2.2.2. Construction of capacity-aware trees

At the topmost hierarchical layer that is involved in search of the video, the control leader of the cluster sends a message to all source streaming leaders to start tree construction. On top of the hierarchy, some streaming leaders are sources to serve video to the set of requesters; and some streaming leaders are requesters to convey video data to their clusters. Multicast tree construction starts at each hierarchical layer with roots being either source nodes or streaming leaders. The topmost leader of the hierarchy informs streaming leaders to start the algorithm and gives them the address list of other streaming leaders in that hierarchical layer. Recall that streaming leaders have the distance in terms of delay, i.e. Round Trip Time (RTT) value between them and other streaming leaders. All source streaming leaders send these distance values to each other. They also exchange their upload bandwidth capacity. After that, children selection is done by considering the capacity of the source node and distance between source and requester nodes. Multicast trees that are constructed by considering node's capacity ensure that nodes having high capacity are located close to the root in the tree. Thus, a node having low upload bandwidth will not form a bottleneck on the paths of the tree. In each tree layer, each node selects closest node as children among nodes having high capacity. After completing the part of the tree at the upper layer, a streaming leader continues to construct the multicast tree at lower hierarchical layer. In this manner, streaming leaders distribute their capacity over hierarchical layers, which they belong to. Each leader reserves one slot for each hierarchical layer l since it has to select at least one child to convey video data to its cluster at hierarchical layer l. At the topmost hierarchical layer, after reserving the necessary number of slots for lower hierarchical layers, leaders use the leftover capacity for children and backup in this top hierarchical layer. This procedure is repeated until the leaders reach the hierarchical layer 0. Multicast tree is completed after the trees in clusters at hierarchical layer 0 are completed. Note that a hefty portion of slots of streaming leaders' capacity is allocated at the topmost hierarchical layer. This approach allows reserving more backup slots at higher layers of the multicast tree, hence the tree resilience increases. Nodes joining the system after formation of a multicast tree for a particular video are placed as a leaf so that the streaming of other nodes is not interrupted. Backup parent selection procedure is given in next section comprehensively. Performance of the algorithm without backup parents can be found in [19].

The following theorem a proof of which is given in the appendix establishes message complexity of forming a multicast tree in a P2P system.

Theorem 2. *If channel capacity of each node is* 2, *then in the worst case; the message complexity of the tree construction algorithm is bounded by* $O(kN)$, *where* k *is the number of nodes within a cluster, and* N *is the total number of nodes in the system.*

Fig. 1 illustrates the relationship between control and streaming leaders for five clusters. Control and streaming leaders of a cluster in hierarchical layer 0 can also be control or streaming leader of a cluster in hierarchical layer 1, such as "a" being the control leader of two clusters in hierarchical layers 0 and 1, or "1" being the streaming leader of two clusters. On the other hand, not all nodes having a leadership in hierarchical layer 0 be the leader of a cluster in hierarchical layer 1, such as node 2 being the streaming leader of the cluster in hierarchical layer 0 but being an ordinary node requesting video in hierarchical layer 1. Arrows in the figure represent the parent–child relationship in a multicast tree. Note that although it is not illustrated in Fig. 1, a control leader can also be the streaming leader of its cluster.

3. Backup parent pools

3.1. Selection algorithm of backup parents

In P2P systems, churn, node crashes as well as unpredictable dropouts are very frequent and resilience is a major issue. In the sequel, we will use the term "node failure" to represent all of these events. Unfortunately, many of the tree based approaches do not meet the resiliency requirements. In the literature, the proposed proactive techniques that use backup parent functionality utilize only one backup parent for a node and hence bandwidth of the backup node is reserved for that particular node throughout streaming. However, fixing such bandwidth for a particular node cannot be an efficient way to balance bandwidth utilization–backup slot reservation trade-off. Another disadvantage of this approach is that when a node detects its backup parent failure, it usually takes time to find a new backup parent and while searching for new backup parent, node may consume its buffer and may experience drained video buffer. We propose to use backup parent pools in order to improve resilience of multicast tree construction algorithm given in [19], and provide seamless video streaming for the nodes that experience parent node failure. The selection of backup parents is a proactive approach since a rescue plan against failure is done before failure happens. In our system, one or more nodes in a tree layer may be selected as backup parents but they cannot reserve all capacity as backup. The motivation of the design can be summarized as follows:

- Reserving all slots of a high capacity node for backup should be prohibited due to two reasons: First, it introduces inefficient bandwidth utilization. Second, it forms a single point of failure.
- In addition to repairing tree when a crash or dropout occurs, the backup slots of a node are used for maintaining quality of service during network congestion.

Backup parents are selected in the phase of multicast tree formation. In each level of the multicast tree, different set of backup parent pool is constructed. In other words, a backup parent in a tree level tl is a node in backup pool for any node in tree level $tl+1$. In Fig. 2, the algorithm for selection of the backup parents is given. In our multicast tree formation algorithm, in each hierarchical layer, some part of the tree is constructed. Multicast tree – or in case of multiple source, multicast trees – is completed after traversing among all hierarchical layers. Note that the algorithm is sketched in iterative manner in Fig. 2. Since all nodes participating in the tree run the distributed algorithm simultaneously, parts of the multicast tree is formed in different nodes.

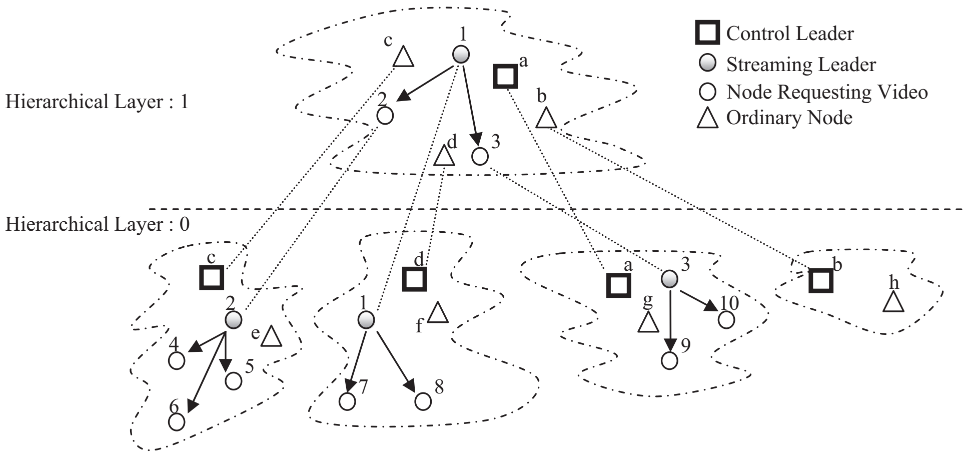

Fig. 1. Hierarchical clusters with control and streaming leaders.

```
for the top hierarchical layer do
    assign actual sources to current virtual sources
    reserve capacity from all of the nodes for lower layer clusters
for all of the hierarchical layers do
    for all the clusters of the current hierarchical layer do
        //determine maximum backup capacity of the cluster
        while (current virtual source set is not empty)
            sort the current virtual sources according to their channel capacities
            determine minimum backup capacity of the current tree layer as the
                capacity of the maximum capacity among the current virtual source nodes
            reserve minimum backup capacity from current virtual sources by reserving channels in a
                round-robin fashion without exceeding the maximum backup capacity of the
                cluster
            choose children for all the current virtual sources by invoking the consensus algorithm
            transfer all the leftover capacity of the current virtual sources to their respective clusters
                in lower hierarchical layer
            update current virtual source set as the current children set
        endwhile
    endfor
endfor
```

Fig. 2. Multicast tree construction algorithm with backup parent selection.

According to the algorithm, the nodes in each tree level act as virtual source. Clearly actual sources are assigned as virtual sources when algorithm starts and these nodes become root of the tree, which they belong to. In each tree level, the maximum number of children that can be connected to a node in that tree level is determined. This number specifies needed empty slots in that tree level, if the node having maximum capacity fails during streaming, then all children of that node can connect to these empty slots. A node can reserve at most 50% of its capacity. This restriction is determined after large number of simulations and it is seen that 50% gives the best performance in terms of dealing with congestion and tree recovery time. But nodes in the system generally reserve less than 50% of their capacity as backup. Furthermore, not every node in the system is also a backup parent (i.e. they do not reserve any backup slots). Algorithm design ensures that these backup slots belong to different nodes, so these nodes are added to backup parent pool for this level. This procedure is repeated until tree is formed, i.e. every node is connected to the tree in the lowest hierarchical layer.

Once backup pool is determined in a tree level *tl*, each node in *tl* sends *parent* message carrying selected backup parents as well as its own parent to its children. Thus, every node knows the address of grandparent in addition to backup parents. When a node fails, one slot of the parent of the failed node becomes available, and the closest grandchild connects to that available slot.

As an example, suppose there are three source nodes within hierarchical clustering given in Fig. 3. In hierarchical layer 1, there are 12 nodes representing their clusters in hierarchical layer 0 (bottom hierarchical layer). These 12 nodes are streaming leaders of 12 clusters of lower hierarchical layer and each of them becomes the root of the trees in their clusters. According to the consensus algorithm, each source node starts multicast tree

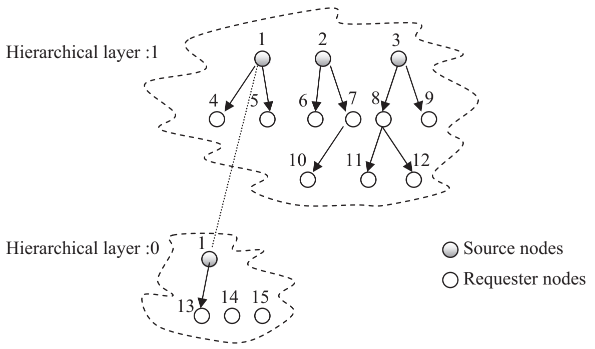

Fig. 3. An example tree structure.

construction algorithm and chooses the children nodes, which are the closest nodes among requester nodes. Suppose $c(1)=c(2)=5$ and $c(3)=4$, where $c(1)$ stands for the capacity of node 1, i.e. maximum number of children that can be connected to node 1. Each node in this cluster reserves one channel from its capacity for hierarchical layer 0 since these nodes are assigned as being root of their clusters in lower layer and has to have at least one child. After this allocation, for each tree layer, the number of necessary backup slots (δ) is determined. For tree layer 1, this value is set to 5 since maximum capacity among $c(1)$, $c(2)$ and $c(3)$ equals 5. In round-robin fashion, number of allocated backup parent slots equals 2, 2, 1 for node 1, 2 and 3, respectively, and backup parent list $BPL_{tl=1}=\{1, 2, 3\}$ is set to be sent to children. While sending backup parents' information, parent of the children is extracted from the list. Now suppose children selection is done as given in Fig. 3. New nodes will be having consensus on the set of nodes $S_{tl=2}=\{4, 5, \ldots, 9\}$. This new set is the set of new virtual source nodes, and algorithm starts again by determining δ, and these backup slots are chosen among new virtual source nodes. Note that backup parent list given to the nodes in $S_{tl=2}$ is $BPL_{tl=1}$. In this tree layer, suppose all nodes have capacity of 4. This time, the value δ equals 4, and new set of backup parents can be chosen as $BPL_{tl=2}=\{4, 5, 6, 7\}$. $BPL_{tl=2}$ and grandparent address information is send to nodes in tree layer 3, which consists of nodes 10, 11 and 12. The algorithm continues running at hierarchical layer 0 at the same time. In Fig. 3, only one cluster in hierarchical layer 0 is illustrated. Since node 1 has 2 backup slots and has one child connected to it from hierarchical layer 1, only one child can be connected to it (recall that the capacity of node 1 is 5). Backup parent list sent to node 13 is again $BPL_{tl=1}=\{2, 3\}$ since node 13 is in the second tree layer.

Note that one of the children of a failed node will always connect to grandparent. Hence, if capacities of nodes are limited, then the algorithm assigns less number of children to nodes and forms deeper trees. In particular, if all of the capacities of nodes are one, then the multicast tree takes the form of a chain with backup mechanism provided by the grandparents.

For backup parent selection, in addition to round-robin approach, we also implemented another approach based on what we call "greedy selection". In greedy approach, backup parent slots are chosen in greedy manner for a tree layer, i.e., all slots available as backup slots of the first node in consensus list are chosen as backup slots. Then all available backup slots of the second node in consensus list are chosen. This procedure continues until the number of needed backup slots is completed. The comparative performance results between round-robin and greedy approaches are given in Section 4.

Backup parent selection algorithm provides each node with a backup parent list that contains nodes from upper layers of the tree. By this way, timing constraints for streaming are satisfied. Children of root have backup parents that belong to other trees or sources.

3.2. Streaming protocol with backup parents

Determining backup parent pools and notifying related nodes about backup parents is proactive part of proposed resilient technique. Each node participating in the tree holds a set of node addresses in case of parent node failure. Note that even though the candidate backup nodes are known, the determination of which of them is to be connected to is done after parent failure.

3.2.1. Reactive component

When failure is detected, repair of tree by replacing the failed node with a backup parent is reactive part of the proposed resilient system. During this process, the following rules are applied by the nodes:

Rule 1: Do not proceed with parent replacement if the current parent sends a delayed "I am alive" message.

Rule 2: If there is no sibling, try to connect grandparent node. In case of siblings, the closest one connects to grandparent whereas the others find closest backup parent from the pool.

Rule 3: Among the backup parents, the ones that take part in lower layers of the multicast tree should be chosen first so that the streaming performance does not deteriorate. Note that a node in backup pool of a certain tree layer is guaranteed to have received the most recent video since it takes part in tree layers that are above the nodes to be backed up. This is essential for seamless streaming that imposes stringent time limits for tree repair process during which the node without parent consumes video from its buffer.

Note that only the node that has lost its parent runs the repair process and hence the message complexity is considerably lower as compared to grandparent-all and root-all approaches [4] of the literature. In order to implement Rule 2, all nodes try to connect to a backup parent node from their pool in turn. Consequently, if there is more than one sibling, all siblings try to connect to their grandparent by signaling their parent have left. Grandparent accepts to connect to the node from which it has received the first message. Same strategy applies to nodes competing to connect to a backup parent from the pool. For each sibling, backup parent list is exactly the same and ordered. Sending backup request to backup parents in order is important since backup parents are ordered according to the tree level, which they belong to. For example, a node in tree level n has a backup parent list consisting of nodes from tree level $n-1$ to tree level 0. Obviously, if a node is tree level n, it has not enough capacity to locate in upper tree levels. After detecting parent failure, it should first try to connect to a backup parent from tree level $n-1$ since bandwidth constraints using when multicast tree construction. If it cannot find a suitable backup parent to connect in tree level $n-1$, it then tries to connect a new parent from upper levels.

In order to meet the timing constraints, each node maintains a connection trial number ν_{ct} and sends this value to each backup parent that it tries to connect. A backup parent node makes its decision about accepting the connection by looking at the value ν_{ct}. If α and β are the identifiers of two predetermined thresholds, the heuristics running by a backup node on accepting a connection request is given in Fig. 4. The input of the algorithm is ν_{ct} and output is *ACCEPT* or *REJECT*.

On the 9th step of the algorithm, a procedure called *pushing_down* is run. This is the situation of parent search ending up with no backup parent. In such a case, pushing down method pushes the node down the tree. Backup node sends the address of its child having maximum capacity to backup searching node. Node receiving that address then tries to connect to this new candidate parent. Note that pushing down operation is done only when buffer level of the node falls under a pre-defined threshold and still no backup parent is found. Usually, pushing down operation provides the node with an address, which can accept a new connection. Thus, the algorithm guarantees that node detecting parent failure can reconnect a new node before it consumes its buffer.

After streaming starts, all nodes monitor incoming packet rate while they receive video packets. Nodes that have not received any packet for a while signal their

```
1....**if** ( $v_{ct} <= \alpha$)
2......**then** *ACCEPT* if there is a slot to support one more child,
3.............*REJECT* if there is no slot empty
4....**else if** ( $\alpha < v_{ct} <= \beta$)
5......**then** *ACCEPT* if there is a slot to support one more child but this time calculate the
                number of slots with the lowest quality bitrate,
6.............*REJECT* otherwise
7.....**else if** ($v_{ct} > \beta$)
8........**then** *ACCEPT* if there is a slot to support one more child but this time calculate the
                number of slots with the lowest quality bitrate,
9...............*REJECT* by *pushing_down* otherwise
10...**end if**
```

Fig. 4. Connection establishment algorithm.

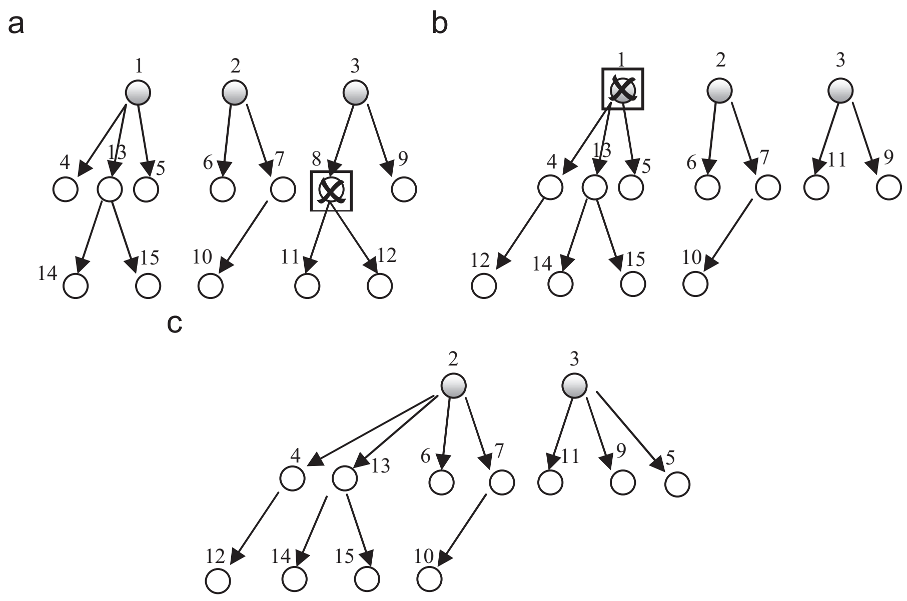

Fig. 5. An example of failure scenario. (a) Tree structure before failure of node 8. (b) Tree structure after failure of node 8. (c) Tree structure after failure of node 1.

parent to check if their parents are alive or not. If parent failure is detected, the node starts backup parent search and for each connection trial, it increments the value of v_{ct}. In a round robin fashion, the backup parents in the pool are polled. If buffer level of the node decreases below a threshold, the node moves itself to urgent state and sets the value of v_{ct} to β.

In Fig. 5, one possible failure scenario is illustrated for the tree given in Fig. 3. During streaming, suppose node 8 fails and its two children 11 and 12 starts backup parent search. They both have the list {3, 4, 5, 6, 7}, where node 3 is their grandparent. First, they try to connect to node 3, and the closest grandchild connects to its grandparent, which is node 11. Node 12 then tries to connect to node 4 and since node 4 has available backup slots, it accepts the connection. If node 1 fails together with node 8, children of node 1 connect to backup parents of upper tree layer, and one possible tree structure after failure is given in Fig. 5c.

Each node periodically sends heartbeat messages to its parent to inform its aliveness. If a parent node notices that a particular child is not alive, it cancels the connection for that child and this empty slot serves as backup slot. Another way to detect left children is to look at the message coming from a grandchild. If a node detects parent failure, it first signals the grandparent to check if it is available for connection and sends a message by notifying its parent failure. Thus, grandparent node can easily detect its failed child by checking this message.

3.2.2. Quality adaptation

In a video streaming session, one of the most challenging issues is dealing with network congestion. In our system, each node monitors its incoming packet rate and if it detects that the rate is below the rate of base video rate, it signals the parent by notifying bad connection it experiences. The node then waits for parent to adapt connection quality and in this duration, it plays the video data in its buffer.

Under persisting network congestion, the parent node applies the following policies to adapt connection quality:

i. *Decrease the video quality:* This is the first attempt to cope with congestion. During streaming, each node having children sends video with the highest quality that the connection quality and upload bandwidth of the node allows. But if it receives a message about bad connection from any of its children, it extracts enhancement video layers to adapt video quality.

ii. *Choose a victim child:* In case of it is not enough to extract enhancement layers to improve video quality received by children, parent chooses a victim child and drops the connection to this victim child that in turn starts backup parent search and connects to a new parent. Chosen victim child is generally one of its children that sense bad connection quality because of bottleneck between itself and its parent. Bottleneck may occur between a parent node and a child although upload bandwidth of the parent node is high. In this situation, it is the best choice to drop the child before it consumes its buffer.

3.2.3. *Buffer management*

After video streaming is started, all nodes participating in the tree wait for initial buffering time. After initial waiting time t_i is completed, node starts to play video, and there is t_i seconds of video in its buffer. For a packet incoming to node's buffer, the deletion time of the packet is important design issue. In our system, each node implements store and forward mechanism for an incoming packet. For video streaming systems, a node stores the video packet until displaying or forwarding it to the children. However, in our system, deletion of the packet after forwarding it may cause a problem since the node should keep the packet for further children, which may request a connection in the future.

In Fig. 6, appearance of a node's buffer is given. In the figure, after a packet is received by a node, it is placed according to its playout timestamp in preprocessing time. Obviously, each packet should wait in the buffer until it is displayed or forwarded. In the figure, this time is denoted as t_w, and the time between displaying and forwarding the packet is denoted as t_p. According to our system, the need for keeping packet for future usage is denoted as t_f. In order to determine t_f, so the time limit of a node buffer, the height of the tree in terms of delay is considered. When a packet is delivered by a root node at the time t_i, it traverses the tree and at the time $t_i+\varepsilon$, it is received by all

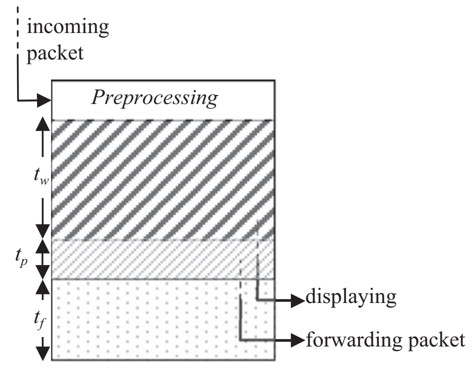

Fig. 6. A Node's buffer.

leaf nodes. If each node keeps the number of packets equal $(t_i+\varepsilon)-t_r$ playout time in the buffer, where t_r is the time of the packet received by the node; even if one of the leaf nodes connect to a node in tree layer 1, it can receive all video packets streamed while it is not connected to any node. As a result of this calculation, t_f should equal to $\varepsilon-(t_r-t_i)$.

4. Performance evaluation

In order to evaluate the proposed algorithm, we have carried out a set of simulations using NS2 environment [22]. In these simulations, we compare two approaches used in selecting backup parents of our algorithm, namely round-robin selection and greedy selection. We analyze tree structure constructed with and without backup parents in order to measure overhead of backup parents. We also compare our proposed algorithm with the algorithm proposed by [16].

4.1. *Simulation setup*

Network infrastructure used in the experiments is generated by GT-ITM module [23]. Underlying physical network is generated as flat random graph consisting up to 250 nodes. The first overlay formed above the physical network is hierarchical clusters that form two or three layers from these 250 nodes. In comparisons with [16], the degree of nodes is set to 4 as used in [16]. In other experiments, the degree of nodes is uniformly distributed between 1 and 5. For all experiments, there is only one source and the source node has available degree of 5. Bitrate values of scalable video used in the experiments are 500, 800 and 1000 Kbps for base and two enhancement layers. Upload bandwidth value of a node having one degree that allows use of one slot may have 500, 800 or 1000 Kbps, a node having more than one degree has upload value of 1000 Kbps for each degree it has. Delay between two nodes having a physical link is between 18 ms and 270 ms according to the distribution of generated by GT-ITM. As streaming starts, each node waits for a time of 5 s for initial buffer filling. Simulations are repeated over 100 times and results are summarized in graphs.

4.2. *Experimental results*

In the first set of experiments, we measure the overhead introduced by reserving backup slots, and compare three type of tree structures, tree without backup parents, tree with backup parents selected in round-robin fashion and tree with backup parents selected in greedy manner. For comparison, distance from source to destination in terms of hop count and delay for each node in the tree are calculated and illustrated in a graph. Repeated simulations are also done using Jeon et al. algorithm in [16] and we compare it with our proposed algorithm. According to [16], when a new node joins to the system, it contacts to the root of multicast tree and obtains nominated node addresses, which have enough capacity to have a new child in the tree. The new node then measures the distances between itself and nominated nodes to find a suitable parent. After placing itself in the multicast tree, it finds a backup parent using distance measure. Thus, each node has one backup parent. For siblings, the closest sibling to the grandparent chooses its grandparent as backup parent.

In Figs. 7 and 8, cdf graph of total delay and hop count for the nodes in the multicast tree are given, respectively. As it can be seen from the figures, proposed algorithm gives better results in terms of total delay and hop count than Jeon's algorithm [16]. On the other hand, proposed algorithm achieves similar performance when compared with the tree constructed without backup parents. This shows that the overhead introduced using backup parents is within reasonable boundaries.

Figs. 9 and 10 show average hop count and average total delay for single and multiple trees (multiple sources). When Figs. 9 and 10 are examined, it is observed that greedy approach yields slightly better performance. Since the number of selected backup parents in round robin approach is greater than that selected in greedy approach, the number of selected children nodes in round robin approach is smaller than that of greedy approach. This causes additional height in the tree constructed using round robin. The overhead in tree height introduced by backup parent selection can also be seen when number of tree is increased from one to two. If there is one tree, i.e. there is one source, no backup parent is chosen, hence no backup slots are reserved and children candidates can connect to all slots to the source. But if there is more than one source, a subset of the sources reserves some slots as backup and this causes some slots of the sources staying idle for backup.

In the second part of experiments, we repeat the simulations over congested and error prone network. Errors can happen by a node failure (includes churn) or link failure and results are summarized in graphs according to error rate on network. Nodes are randomly failed with uniform probability p during each tree update interval of one second. Failure rate is classified as low if the failure probability of a node, p, is under 0.2, medium if the failure probability of a node is between 0.2 and 0.4 and high if the failure probability of a node is more than 0.4. It has been observed that more than half of the failed nodes have two or more than two children, going up to eight children. This shows that nodes searching for backup excessively utilize backup parent pool and simple "connect to grandparent" approach is not adequate in many cases. In Fig. 11, comparative CDF graph of time gap, i.e., the necessary time for connecting to a new parent after a parent failure, continuity index (ratio of video packets received that arrive before or on its playback deadline to total video) and average received quality by nodes on the tree is given in an environment with low

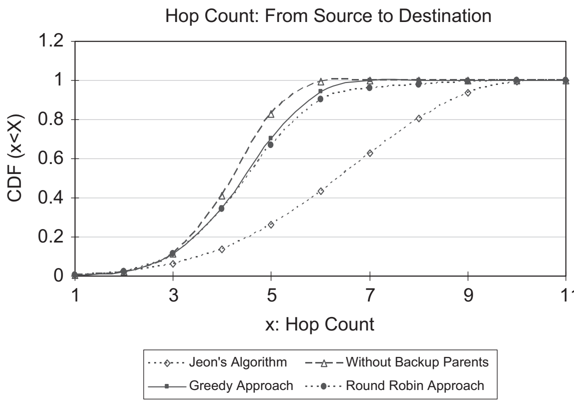

Fig. 8. CDF graph of hop count.

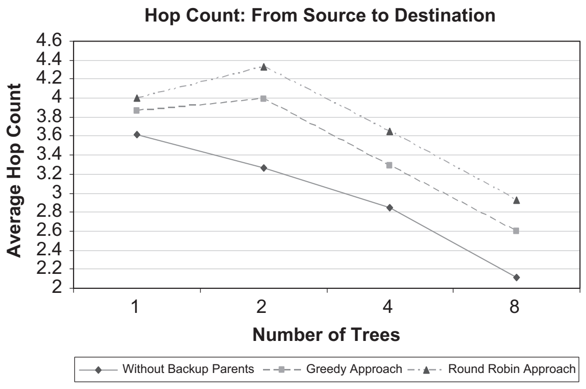

Fig. 9. Average hop count.

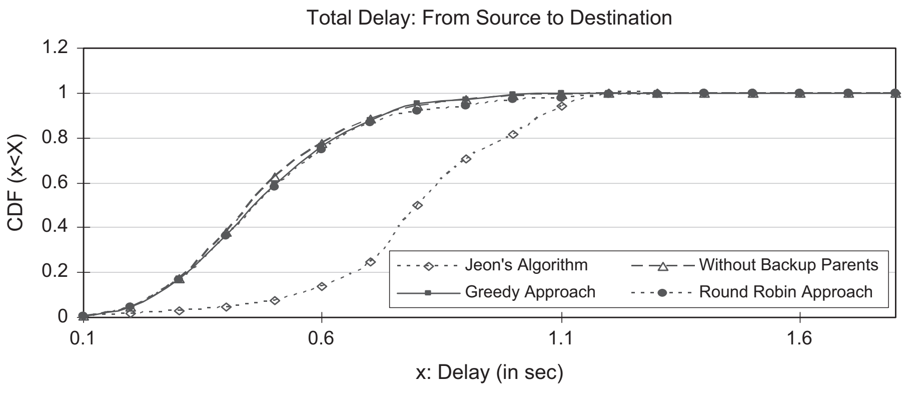

Fig. 7. CDF graph of total delay.

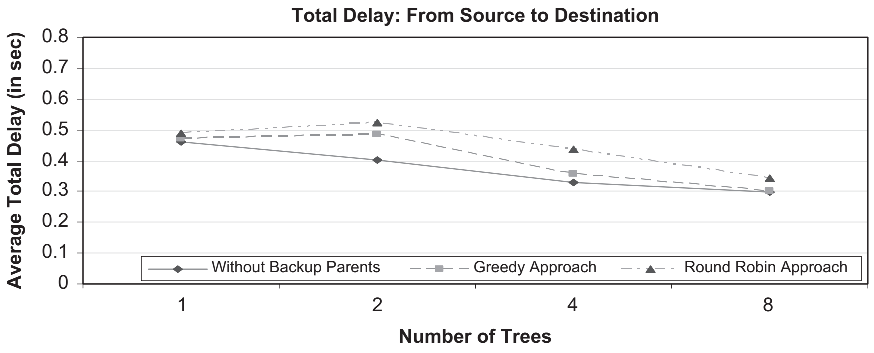

Fig. 10. Average total delay.

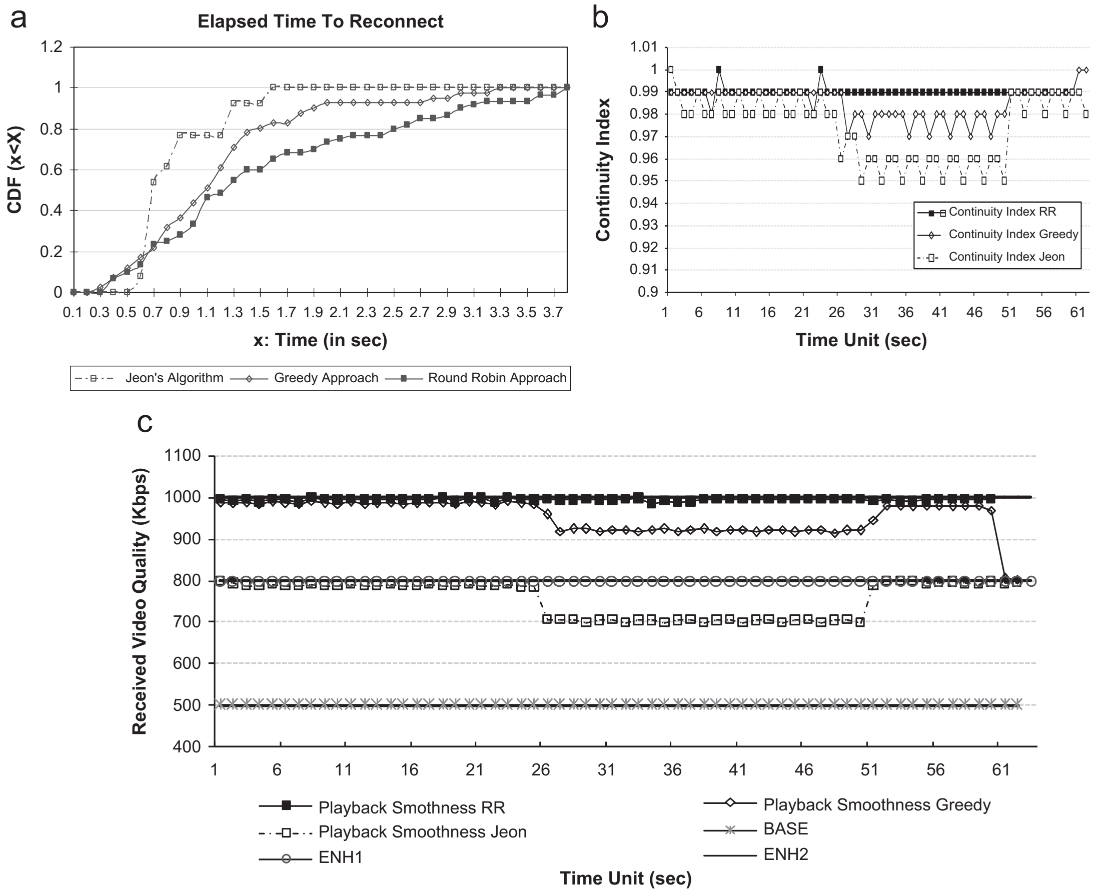

Fig. 11. Low failure rate (p is uniform in $p<0.2$). (a) CDF graph of time gap. (b) Average continuity index over time. (c) Average received quality.

failure rate. Average received quality also represents the playback smoothness observed by peers in the system over time. For low failure rate, Jeon's algorithm [16] gives the best performance in terms of time gap since backup node to be connected after parent failure is known in advance and requires only one message to connect a new parent. Greedy approach has better performance than round robin approach. But for average received quality, round robin approach is slightly better than greedy approach. Since time gap is always under 5 s, no node experiences frozen display and average received quality is never less than 500 kbps. For round robin approach, it has been observed that almost all nodes in the system receive video with highest bit-rate. If the continuity index is more than 0.95, it is accepted that peers play the video at a good rate [24]. The continuity index measured for all algorithms is always more than 0.95 for low failure rate.

In Figs. 12 and 13, time gap, continuity index and average received quality are given when failure rate is at medium level and high level, respectively. We observed that when failure rate increases, some nodes experience frozen display in Jeon's algorithm [16]. When failure rate increases, nodes detecting backup node's failure start to search for finding another backup node. During backup parent search operation, backup node's parent may fail since failure rate is high. In this scenario, the lower level backup searching node has to wait until its backup parent completes backup search procedure. When failure time of the parent node and the backup node of a node is close to each other, time gap required to find a new parent–backup parent pair increases and this may cause the node detecting failure experience display blackout. In Figs. 12a and 13a, percentage of nodes experiencing this scenario and having time gap above 5 s can be seen. This scenario also decreases average received quality. In Fig. 13c, we observe that the proposed algorithm with round robin approach gives the best performance, since it is more resilient to high failure rates because of larger backup parent set. With the proposed algorithm, for medium and high failure rate, nodes in the system can receive video packets without any quality degradation, and most of the nodes continue to receive packets with at least one enhancement layer. In all these experiments, average received quality degrades after tree reconstruction in Jeon's algorithm [16] since it does not consider placing nodes having highest capacity near root. We also observe that such degradation does not happen in our proposed algorithm.

For message complexity comparison of both systems, during streaming, the average number of maintenance and heartbeat messages sent by any node in 1 s are calculated and summarized in Table 1. In [16], maintenance messages include tree maintenance messages such as reporting forbidden nodes to choose as backup parent, determining the closest grandchild to its grandparent, connection request and necessary RTT measurements during streaming. In the proposed system, maintenance messages consist of connection request and connection reject. When we compare the maintenance messages, we see that our proposed system has negligible overhead as compared to that of [16]. On the other hand, since we use multiple backup parents, our heartbeat message rate is almost double that of [16].

5. Conclusions

This work proposes a novel resilient cross layer design in hierarchical clusters and multicast trees using "backup parent pools". The tree formation procedure considers number of sources, scalability layer rates available at the source, as well as delay and available bandwidth over links in an attempt to maximize the quality of received video at each peer. Simulations are performed on NS2 with 250 nodes to demonstrate that the overall performance of the system in terms of average received video quality of all peers is significantly better if peers with higher available bandwidth are placed higher up in the

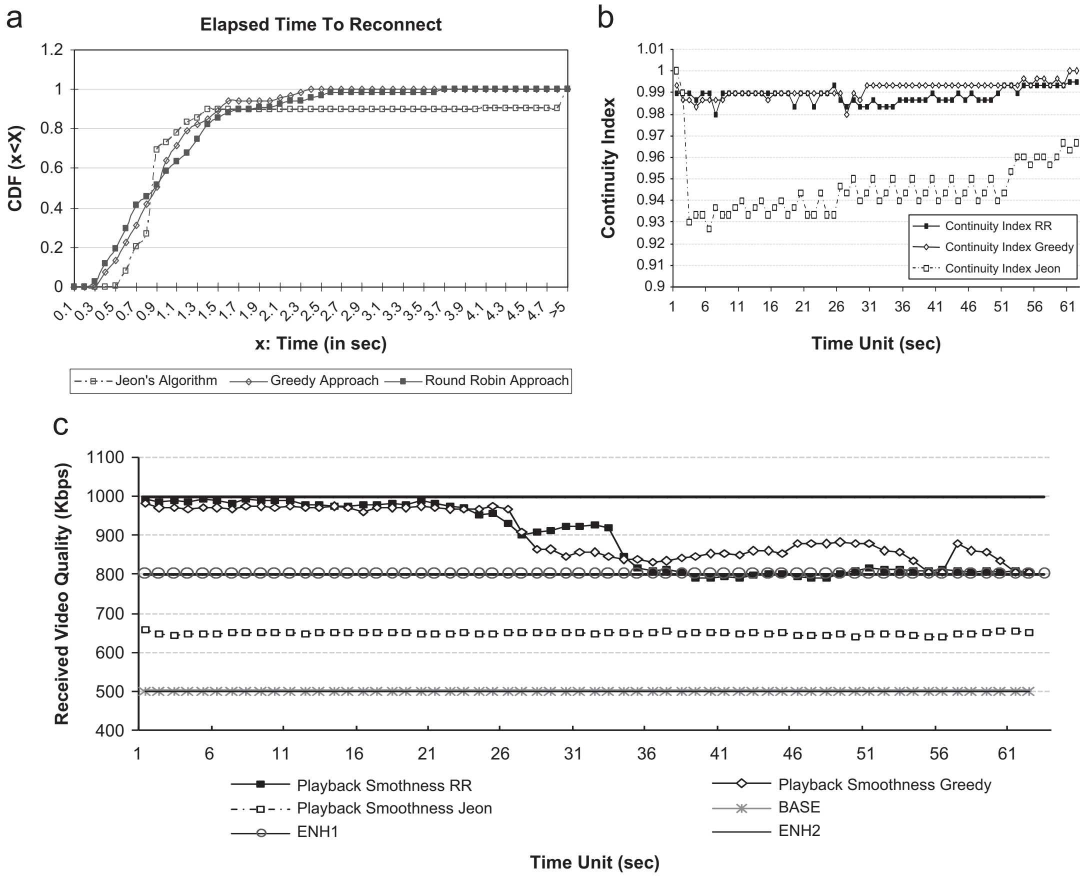

Fig. 12. Medium failure rate (p is uniform in $0.2 \le p < 0.4$). (a) CDF graph of time gap. (b) Average continuity index over time. (c) Average received quality.

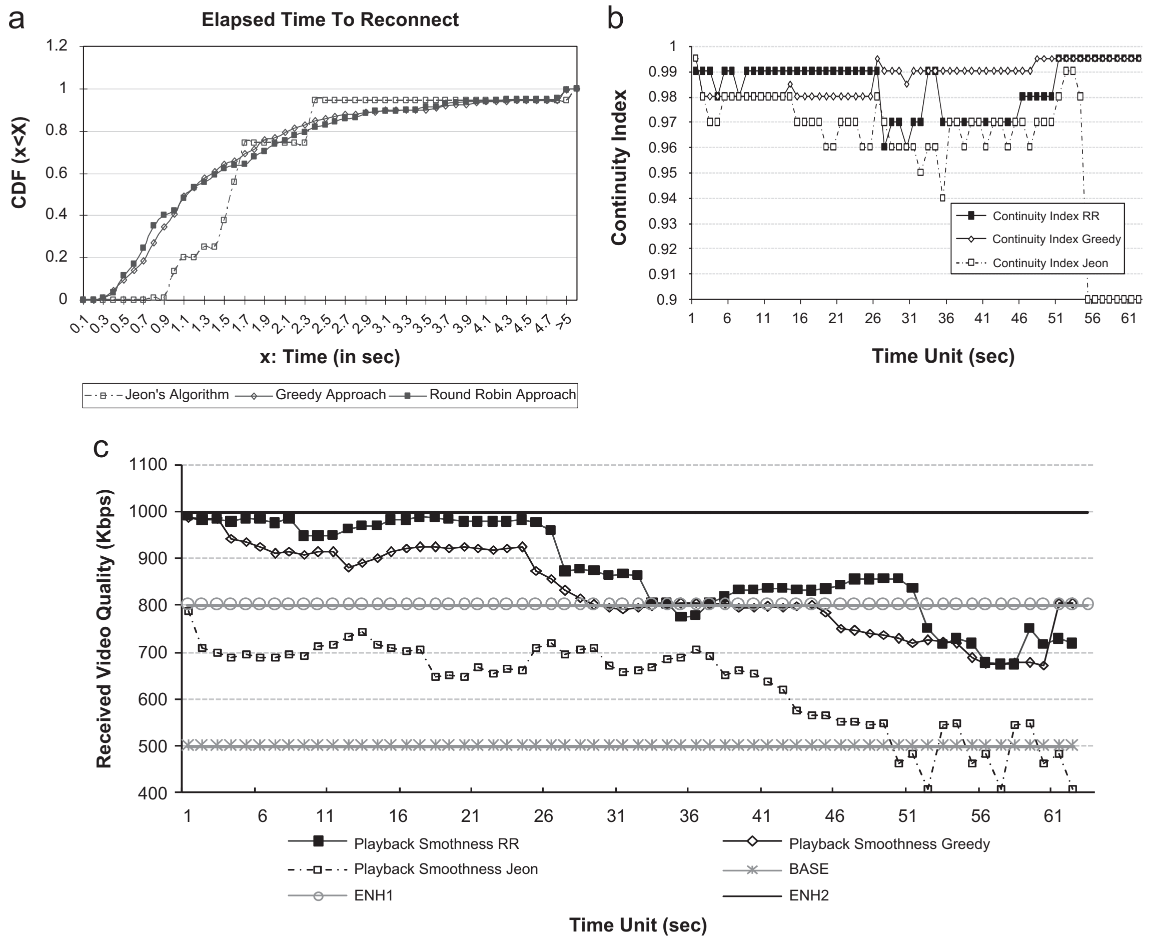

Fig. 13. High failure rate (p is uniform in $0.4 \le p < 1$). (a) CDF graph of time gap. (b) Average continuity index over time. (c) Average received quality.

Table 1
Comparative average number of messages of different types (per sec).

	Average number of maintenance messages	Average number of heartbeat messages
Jeon's algorithm [16]	3.58	0.88
Proposed algorithm	0.01	1.69

trees and peers with lower bandwidth are near the leaves. It has also been shown that the performance of the proposed algorithm in scalable video streaming under different node and link error rates in congested networks is superior to its alternatives in literature. Simulations show that, for each node, selecting more than one backup parent is more resilient than selecting only one backup parent especially in peer to peer systems where node dropout rate is considerably high.

It is well known that the faster the system responds to any problem in a multicast tree, the better the average received video quality. Backup parent pool provides more flexibility for relocation in the multicast tree, which makes dynamic tree maintenance during streaming considerably easier to handle. Developing better and faster algorithms to repair a multicast tree has always been a challenging problem and this study has shown that carefully crafted heuristics involving node capacities as well as other integrity constraints may provide a solution to this problem.

Appendix

Proof of Theorem 1. Considering worst-case scenario, suppose all nodes in the system demand video simultaneously. In each cluster, since every node demands to participate in the multicast tree to be formed, k nodes calculate the distance between each other. This requires $k(k-1)$ message exchange.

If the video is found at the topmost hierarchical layer, streaming leader selection is done in $\log_k(N/k)$ hierarchical layers and all clusters in each layer. From hierarchical layer 0 to hierarchical layer $\log_k(N/k)$, there are $k^{\log_k N/k},\dots,k^2,k,1$ clusters, respectively. Since $k(k-1)$

number of messaging is done in each clusters, we have

$$k^{\log_k N/k} k(k-1)+\cdots+kk(k-1)+1k(k-1) \tag{1}$$

total number of messaging throughout the system. By arranging Eq. (1),

$$k(k-1)k^{\log_k N/k}+\cdots+k+1)=k(k-1)\sum_{i=0}^{\log_k N/k} k^i$$

$$=k(k-1)\left(\frac{k^{\log_k N/k+1}-1}{k-1}\right) \tag{2}$$

which gives us the message complexity of $O(k^{2+}\log_k N/k)=O(kN)$, the theorem is proved. □

Proof of Theorem 2. Suppose that all nodes demand video simultaneously. Then, within a cluster, k nodes participate in a multicast tree. Suppose that the height of the multicast tree is h, and x_0, x_1, …, x_h denote the number of nodes in tree layers 0–h, respectively, with tree layer 0 being the root. Then x_0 is the number of source nodes in the system. For the worst-case scenario, there is one source, hence x_0 equals 1. x_0 chooses x_1 nodes as children, informs them by sending x_1 parent messages and these x_1 nodes send each other necessary information. This requires $x_1(x_1-1)$ messages. In the tree layer 2, x_2 nodes are chosen as children set of x_1 nodes. This process continuous until all nodes connected to the multicast tree. If the total number of messaging (M) is calculated, the following summation is obtained:

$$M=x_1+x_1(x_1-1)+x_2+x_2(x_2-1)+\cdots+x_{h-1}(x_{h-1}-1)+x_h \tag{3}$$

By arranging the formula (3),

$$M=\sum_{i:1}^{h-1} x_i^2+x_h \tag{4}$$

If all nodes have the same capacity of 2, then the obtained tree is a binary tree and in each tree layer i, there are 2^i nodes. So, the formula (4) can be rewritten as below:

$$M=\sum_{i:1}^{h-1}(2^i)^2+2^h$$

$$M=(4^h-1)/3-1+2^h \tag{5}$$

In each cluster M messaging is done,

$$M(1+k+k^2+\cdots+\log_k(N/k))=M\sum_{i:0}^{\log_k N/k} k^i=M\frac{k^{\log_k N/k}-1}{k-1} \tag{6}$$

The message complexity is bounded by $O(4^{\log_2 k} k^{\log_k N/k})=O(kN)$, the theorem is proved. □

References

[1] J. Liu, S.G. Rao, B. Li, H. Zhang, Opportunities and challenges of peer-to-peer Internet video broadcast, Proceedings of the IEEE, Special Issue on Recent Advances in Distributed Multimedia Communications 96 (2008) 11–24.

[2] Z. Chen, B. Li, G. Keung, H. Yin, C. Lin, Y. Wang, How scalable could P2P live media streaming system be with the stringent time constraint? in: Proceedings of the IEEE International Conference on Communications, Germany, 2009, pp. 1–5.

[3] X. Hei, C. Liang, J. Liang, Y. Liu, K.W. Ross, A measurement study of a large-scale P2P IPTV system, IEEE Transactions on Multimedia 9 (2007) 1–15.

[4] Z. Fei, M. Yang, A proactive tree recovery mechanism for resilient overlay multicast, IEEE/ACM Transactions on Networking 15 (2007) 173–186.

[5] S. Birrer, F.E. Bustamante, Resilient Peer-to-Peer Multicast Without the Cost, MMCN, USA, 2005.

[6] S. Banerjee, B. Bhattacharjee, C. Kommareddy, Scalable application layer multicast, ACM SIGCOMM, USA, 2002 (pp. 205–217).

[7] V.N. Padmanabhan, H.J. Wang, P.A. Chou, Resilient peer to peer streaming, in: Proceedings of the IEEE International Conference on Network Protocols, USA, 2003, pp. 16–27.

[8] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, A. Singh, SplitStream: High-bandwidth Multicast in a Cooperative Environment, SOSP, New York, 2003.

[9] Y. Ding, J. Liu, D. Wang, H. Jiang, Peer-to-peer video-on-demand with scalable video coding, Computer Communications 33 (2010) 1589–1597.

[10] R. Rejaie, A. Ortega, PALS: Peer-to-Peer Adaptive Layered Streaming, NOSSDAV, USA, 2003 (pp. 153–161).

[11] Y. Okada, M. Oguro, J. Katto, S. Okubo, A new approach for the construction of ALM trees using layered video coding, ACM Workshop on Advances in Peer-To-Peer Multimedia Streaming, Singapore, 2005, pp. 59–68.

[12] E. Setton, J. Noh, B. Girod, Rate-distortion optimized peer-to-peer multicast streaming, ACM Workshop on Advances in Peer-to-Peer Multimedia Streaming, Singapore, 2005.

[13] P. Baccichet, T. Schierl, T. Wiegand, B. Girod, Low-delay peer-to-peer streaming using scalable video coding, in: Proceedings of the 16th IEEE PV, Switzerland, 2007, pp. 173–181.

[14] S. Banerjee, S. Lee, B. Bhattacharjee, A. Srinivasan, Resilient multicast using overlays, IEEE/ACM Transactions on Networking 14 (2006) 237–248.

[15] S. Birrer, F.E. Bustamante, A comparison of resilient overlay multicast approaches, IEEE Journal on Selected Areas in Communications 25 (2007) 1695–1705.

[16] J.H. Jeon, S.C. Son, J.S. Nam, Overlay multicast tree recovery scheme using a proactive approach, Computer Communications 31 (2008) 3163–3168.

[17] T. Kusumoto, Y. Kunichika, J. Katto, S. Okubo, Proactive route maintenance and overhead reduction for application layer multicast, in: Proceedings of the ICAS/ICNS, Singapore, 2005, pp. 49–58.

[18] Y.-T.H. Li, D. Ren, S.-H.G. Chan, A.C. Begen, Low-delay mesh with peer churns for peer-to-peer streaming, IEEE ICME (2009).

[19] M. Fesci Sayit, E.T. Tunali, A.M. Tekalp, Bandwidth-aware multiple multicast tree formation for P2P scalable video streaming using hierarchical clusters, in: Proceedings of the IEEE ICIP, Egypt, 2009, pp. 945–948.

[20] Y.-H. Chu, S.G. Rao, H. Zhang, A case for end system multicast, IEEE Journal on Selected Areas in Communications 20 (2002) 1456–1471.

[21] X. Zhang, Q. Zhang, Z. Zhang, G. Song, W. Zhu, A construction of locality-aware overlay network: mOverlay and its performance, IEEE Journal on Selected Areas in Communications, Special issue on Recent Advances in Service Overlay Networks 22 (2004) 18–28.

[22] 〈http://www.isi.edu/nsnam/ns/〉.

[23] 〈http://www.cc.gatech.edu/projects/gtitm/〉.

[24] A. Habib, J. Chuang, Service differentiated peer selection: an incentive mechanism for peer-to-peer media streaming, IEEE Transactions on Multimedia 8 (2006) 610–621.