Tuning Performance of P2P Mesh Streaming System Using a Network Evolution Approach[*]

Rui Wang, Depei Qian, Danfeng Zhu, Qinglin Zhu, and Zhongzhi Luan

Beihang University,
Xueyuan Road. 37, 100191 Beijing, China
{rui.wang,depei.qian,danfeng.zhu,qinglin.zhu,
zhongzhi.luan}@jsi.buaa.edu.cn

Abstract. Resilience and startup delay are the most important performance metrics to evaluate the P2P streaming systems. To simultaneously improve the two metrics, we propose several mechanisms at different system evolution stages. At the first stage, media server encodes the stream into multiple sub-streams of the same length. Redundancy is introduced by using Reed-Solomon (RS) coding before distributing the sub-streams to different successors. Each peer in the network establishes a cooperative relationship with others to obtain all required sub-streams. At the stage of new peer arrival, a parent selection algorithm with relatively lower complexity is proposed which takes full advantage of redundant coding. After the peer builds up streaming transmission, it replaces some parents with a latency-based decision mechanism. In case of node failure, a swap-in-turn repairing algorithm between different sub-stream sources is proposed to ensure the high continuity of steaming transmission. Simulation results show that 1) the redundant coding and the parent replace algorithm in case of node failure can effectively reduce interruption of data streams; 2) the codes with higher redundant degree can adapt to more dynamic scenario. Meanwhile, the codes with redundancy does not significantly decrease the effective transmission ratio when network is dynamic; 3) transmission achieves higher performance when the number of substreams is between 8 and 16; and 4) the parent switching mechanism can significantly decrease the startup latency for a big proportion of peers.

Keywords: P2P, Streaming, Mesh, Redundant Coding, Churn.

1 Introduction

Streaming service on the Internet has drawn significant attentions recently for its more interesting content than texts and pictures in web pages. As the participating peers contribute their upload bandwidth capacities to serve the others, P2P streaming system can sustain much more users than that with traditional Client/Server mode under

[*] This work was supported in part by a grant from NSFC under the contract 60673180 and 90612004, and by grants from National High- Tech Program of China (863 program) under the contracts 2006AA01A106 and 2006AA01A118.

the constraint of server's outgoing bandwidth. Among the several types of P2P streaming systems, pull-based random mesh gains dramatic success due to its simplicity and robustness[1]. Many successful commercial systems such as PPLive [2]and UUSee [3] use this mechanism.

In some systems, streaming server splits data into multiple blocks, and delivers them to the participating peers. Each peer queries missing blocks from its neighbors. This block routing introduces great overhead and uncertainty to the transmission of peers.

Instead of using data block as the routing unit, some systems like CoolStreaming[4] uses sub-stream as the routing unit. The streaming server splits data into multiple sub-streams of equal length, and distributes them to different peers. Peers build neighborhood relationship with each other to obtain the complete set of sub-streams. Once the connection is built, the packets belonging to the same sub-stream will be delivered continuously. In this pattern, each peer needs to receive data from several relay peers. When a peer is disabled, the playback in its successor will be interrupted until another parent peer is determined. In a highly dynamic network, peers suffer greatly from the unpredictable user join/quit action.

Since the peer has to receive all the sub-streams before it can playback the streaming, the startup delay is determined by the slowest one. As both the P2P network organization and the neighbor selection are random, the arrival time of different sub-streams may vary significantly, and the slower ones will slow down the playback and increase the startup time.

According to the conclusion of CoolStreaming[4], the system dynamics is the most critical factor that affects the overall performance, and the critical performance problem in a P2P streaming system is the excessive startup time. Through the measurement and analysis to some commercial P2P streaming system, [5] found that in these systems a lot of important decisions, such as how to pick a parent, seem to follow a randomized greedy algorithm.

This paper uses a network evolution approach to optimize the system performance. We propose several mechanisms at the different stages of the system evolution in order to cope with the node dynamics and to decrease the startup delay.

The rest of this paper is organized as follows: Section 2 presents a concise review of solutions for the nodes dynamics and startup delay in P2P streaming networks. Section 3 models the system. Section 4 gives the algorithms for parent selection and adjustment, and node failure handling. Our simulation methodology and results are described in Section 5. Finally, section 6 gives the conclusion of the paper.

2 Related Works

To solve the problem of transmission interruption caused by nodes departure, PRIME [6] used the ratio of bandwidth and peer degree as a metric named bandwidth-degree condition, to evaluate the system performance. Once the ratio value changes, system can immediately detect the bottleneck and relocate the bandwidth.

Feng [7] and Zimu [8] found that some stable nodes in P2P streaming networks affect the performance greatly though their amount is few. So they tried to identify the stable nodes and enable them to play more important roles in the system.

Redundant Coding such as Reed-Solomon [9] is another solution to avoid transmission interruption. Encoded to multiple sub-streams redundantly, the data could be recovered at peers which received any subset containing a certain number of sub-streams of the streaming.

Damiano[10] analyzed the mesh streaming system with a stochastic graph theory and drew the relations between delay and the number of sub-streams. It demonstrated that the transmission with multiple sub-streams is necessary to the system performance. However, it does not improve the system stability with redundancy of coding, and does not concern the influence of nodes failure on the successors.

Kumar[11] used buffer to alleviate the interruption when nodes are disabled. Through the stochastic fluid analysis to the mesh streaming system, it showed that buffer can dramatically improve the stability of the system, since peers with more buffered data will have longer time to find a substitute data source when a parent is disabled. But large buffer will significantly increase the startup delay.

Zhou [12] studied the greedy strategy and the rare first strategy used in data searching using stochastic model, and proposed a mixed strategy that can be used to achieve a good balance between the continuity and startup latency. It has similar intent with this paper, while it is in different approach, and does not make use of the redundancy of the coding.

S. Liu[13] derived the performance bounds for minimum server load, maximum streaming rate, and minimum tree depth under different peer selection constraints in P2P streaming networks. Though this work provides excellent insights, it ignores the dynamics of the network.

3 System Model and Assumption

In this system, following assumptions are taken:

Assumption 1: streaming server splits the data into S' sub-streams of the same length, and then encodes it with $RS(S, S')$ coding to S sub-streams, $S > S'$. Peers that have received any S' sub-stream can recover the data, as shown in Fig. 1.

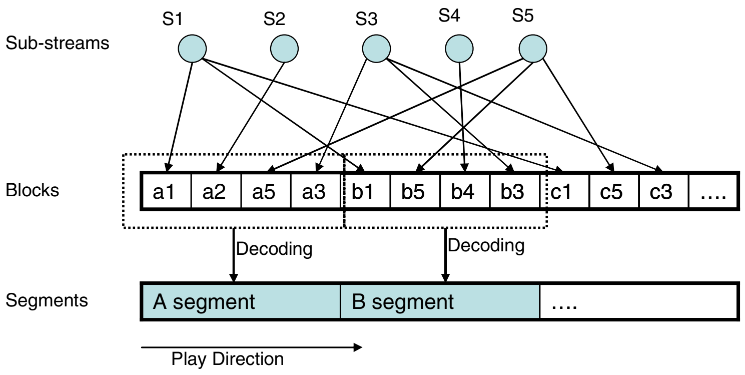

Fig. 1. Buffer filling in redundant sub-stream pattern (S'=4,S=5)

A set of existing peers will be presented to the new peer joining in the system randomly. The new peer selects a subset of this peer set as its neighbors, and requires the sub-streams from them. Once a peer gets a sub-stream, it can provide the sub-stream for other peers. Constrained by the limitation of outgoing bandwidth, it only provides sub-streams with lower latency.

Assumption 2: each node downloads a specific sub-stream from a single node, and each node downloads only a single stripe from a given parent, even if the parent could provide multiple sub-streams.

Assumption 3: each peer has B neighbors. Peer selects S neighbors as the data providers named as active parents, and selects D neighbors as substitutes which will be used in case of active parent failure.

We define the relationship of peers in the system:

Definition 1. Two peers are *neighbors* if they are connected with each other.

Definition 2. Peer i is an *active parent* of peer j about sub-stream k if peer i provides sub-stream k for peer j, and j is an offspring of peer i.

Definition 3. If peer i could provide sub-stream k for peer j but not really provide it at the moment, peer i is a *substitute parent* of peer j about sub-stream k, and j is an offspring of peer i.

In this paper, we assume that a substitute parent does not reserve bandwidth for its offspring.

We abstract the network as a graph, the vertices of the graph represent the peers, and edges connecting vertices represent the connections between peers.

We use a $n \times n$ connectivity matrix C to describe the relationship of peers in the network. The value of element c_{ij} in C represents the relationship of peer i and j. The definition of c_{ij} is as follows:

$$c_{ij} = \begin{cases} 0 & i = j \\ 0 & i \text{ and } j \text{ are not neighbors} \\ \eta & i \text{ and } j \text{ are neighbors} \\ \alpha_k & i \text{ is active parent of } j \text{ about } k \\ \beta_k & i \text{ is substitute parent of } j \text{ about } k \end{cases}$$

Note that the parent relationship is directed. Due to the limitation of outgoing bandwidth, each peer has limited offspring peers. This connectivity matrix maintains the global information, while for an arbitrary peer i, it only needs to maintain the information about the ith line, $L_i = [c_{i0}, c_{i1}, \cdots, c_{in}]$, which contains all the offspring of

i, and the ith column, $V_i = (c_{0i}, c_{1i} \cdots, c_{ni})^T$, which contains all the parents of i. In each of the vector the peer maintains, it omits the element whose value is 0, i.e., the peer does not maintain the disconnected peers. So this matrix is a distributed description structure that allows the peers to maintain its local information.

4 System Evolution

From the evolution's point of view, the state transfer of the network is driven by the change of relationship among nodes. In this section, we describe the processing of peer arrival and departure.

4.1 Peer Arrival

At any time t, the instantaneous state of network could be represented by the connectivity matrix C. When a new peer b arrives, it contacts one existing node, and initializes its neighbor list. The matrix C will be added with a new row and a new column and turns to a $(n+1)\times(n+1)$ matrix with the value of each element of the new row and new column filled with η. The peer b asks its neighbors for available sub-streams, and gets a vector $V_{i,b}$ from each neighbor containing the sub-streams that the neighbor can provide. Each vector has S elements, each of them represents a sub-stream. The value of the kth element is a_k if the kth sub-stream is available in this neighbor, else it is 0. Within the set of all the vectors $\{V_{0,b}, V_{1,b}, \cdots V_{x,b}\}$, peer b selects the source for each sub-stream. So for arbitrary sub-stream k, b selects a neighbor as its active parent if in the vector the value of kth element is α_k. Besides the active parents, the peer b also needs to select a roughly equal number of substitute parents for each sub-stream. That is, in the connectivity matrix, we need to fill in the column vector $V_b = (c_{0b}, c_{1b} \cdots, c_{nb})^T$ of peer b, ensure that the V_b at least contains all elements in the set $\{\alpha_1, \alpha_2, \cdots, \alpha_n\}$, and contains the elements in $\{\beta_1, \beta_2, \cdots, \beta_n\}$ as uniformly as possible.

The parent selection problem above could be transferred to a bipartite graph matching problem and solved by Hungarian Algorithm [14] with complexity of $O(n^3)$. With the redundancy of the coding, we do not need a complete matching. The description of parent selection algorithm is as follows:

1) Let N be the neighbor set, here we set $|N| = n$. Let S_{substr} denote the sub-stream set. The set of sub-stream set which could be provided by each neighbor is denoted by $\Gamma = \{S_0, \cdots S_n\}$;

2) Sort the neighbors decreasingly according to the numbers of sub-streams they can provide, and then get the neighbor vector N_s;
3) Search the provider for each sub-stream in N_s. When all the sub-streams get their provider, stop searching and record the position. Then we get a sub vector N' of vector N_s;
4) Sort the sub-stream identifies increasingly to form a vector of Str;
5) For each sub-stream in Str, we select the node as its parent that can provide fewer sub-streams than other node . Once the node is selected as a parent, it is deleted from the neighbor set;
6) Search more substitute parents for each sub-stream in Str that does not have average number of providers until the numbers of its providers reaches the average value, or all the neighbors are checked.

The complexity of this algorithm is $O(n^2)$ which is better than the Hungarian Algorithm. The pseudo code is shown as follows:

Algorithm 1. Parent Selection Algorithm

```
Input: neighbor set N; sub-stream set S, sub-streams of
 each neighbor {S1,S2,…Sn}.
Output: active parents set Vact, substitute set Vsub.
N=SortBySubStreams(N);
foreach n in N do // count the providers of sub-stream
  if (Marked(S) and |Ns|>=|S|) break;
  foreach s in Sn { Mark(s, S); add n to SRCs} // SRCs
   represents the set of neighbor that can provide sub-
   stream s.
  add n to Ns;
end foreach
Str= SortSubstreamBySourceNumber(S);
foreach s in Str do //select parent
  foreach n in SRCs do
    vi = the last n;
    add vi to Vact;
    delete vi from Ns;
  end foreach
end foreach
foreach s in S do // add sub parents
  if (|SRCs|<Mean{|SRC0,SRC1,…,SRCs|})
    foreach n in N do
      if ((s in Sn) and (n notin Ns))  add n to Vsub;
    end foreach
  endif
end foreach
```

Following is a simple example for the peer arrival processing. We assume that the number of sub-streams $S = 3$, $S' = 2$, so the possible value of α_k is in $\{\alpha_1, \alpha_2, \alpha_3\}$. There are 5 peers in the network besides the source server. The network connectivity is shown in Fig. 2(a).

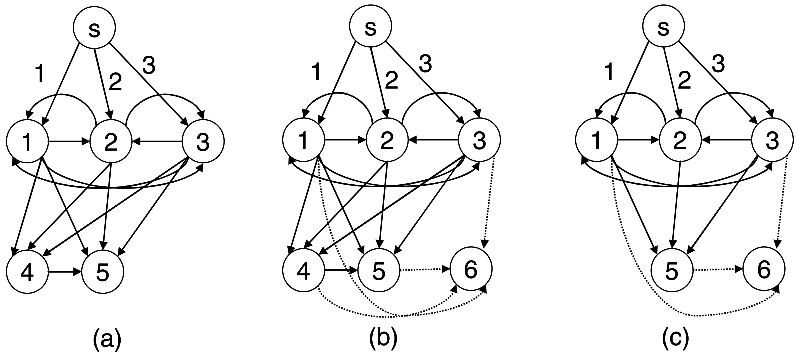

Fig. 2. Peer arrival and departure (a) Original network (b) peer 6 arrival (c) peer 4 quit

We assume that the connectivity matrix of Fig. 2 (a) is:

$$C = \begin{bmatrix} 0 & \alpha_1 & \alpha_1 & \alpha_1 & \alpha_1 \\ \alpha_2 & 0 & \alpha_2 & \alpha_2 & \beta_2 \\ \alpha_3 & \alpha_3 & 0 & \alpha_3 & \alpha_3 \\ \eta & \eta & \eta & 0 & \alpha_2 \\ \eta & \eta & \eta & \eta & 0 \end{bmatrix}$$

Note that for a given network connections, the connectivity matrix is not unique. Assuming that a new peer 6 arrives and builds the connection relationship as Fig. 2 (b), the matrix C turns to be

$$C = \begin{bmatrix} 0 & \alpha_1 & \alpha_1 & \alpha_1 & \alpha_1 & \eta \\ \alpha_2 & 0 & \alpha_2 & \alpha_2 & \beta_2 & 0 \\ \alpha_3 & \alpha_2 & 0 & \alpha_3 & \alpha_3 & \eta \\ \eta & \eta & \eta & 0 & \alpha_2 & \eta \\ \eta & \eta & \eta & \eta & 0 & \eta \\ \eta & 0 & \eta & \eta & \eta & 0 \end{bmatrix}$$

Among the matrix, the sixth row represents all the offspring of peer 6, and the sixth column represents all the parents of peer 6. Then peer 6 asks its neighbors what resource they can provide. Let's assume that peer 6 receives the following resource vectors:

$$V_{1,6} = [\alpha_1, \alpha_2, 0]$$
$$V_{3,6} = [0, \alpha_2, \alpha_3]$$
$$V_{4,6} = [0, 0, \alpha_3]$$
$$V_{5,6} = [\alpha_1, 0, 0]$$

After the selection algorithm, the matrix turns to

$$C = \begin{bmatrix} 0 & \alpha_1 & \alpha_1 & \alpha_1 & \alpha_1 & \alpha_1 \\ \alpha_2 & 0 & \alpha_2 & \alpha_2 & \beta_2 & 0 \\ \alpha_3 & \alpha_2 & 0 & \alpha_3 & \alpha_3 & \alpha_2 \\ \eta & \eta & \eta & 0 & \alpha_2 & \alpha_3 \\ \eta & \eta & \eta & \eta & 0 & \beta_1 \\ \eta & 0 & \eta & \eta & \eta & 0 \end{bmatrix}$$

We can see that there is no α_k or β_k in the last row of C. When a new peer joins in the system, the row corresponding to the contribution of the new peer will be formed this way, which means, the new peer has not contributed its bandwidth to the existing nodes in the network yet. So we apply a feedback process in the network. New peers send the sub-stream identities that it can provide to some of the originally existing peers, so that the new peers could be the substitute parents of the existing ones. Peers with not enough substitute parents will make use of the feedback periodically. The procedure is similar to the parent selection illustrated above. The following matrix is a possible result of the feedback action.

$$C = \begin{bmatrix} 0 & \alpha_1 & \alpha_1 & \alpha_1 & \alpha_1 & \alpha_1 \\ \alpha_2 & 0 & \alpha_2 & \alpha_2 & \beta_2 & 0 \\ \alpha_3 & \alpha_2 & 0 & \alpha_3 & \alpha_3 & \alpha_2 \\ \beta_2 & \eta & \beta_2 & 0 & \alpha_2 & \alpha_3 \\ \beta_3 & \beta_1 & \eta & \eta & 0 & \beta_1 \\ \eta & 0 & \eta & \beta_2 & \beta_3 & 0 \end{bmatrix}$$

4.2 Parent Adjustment

A peer needs to receive at least S' sub-streams of one streaming segment before the segment can be decoded and played. So the playback delay of each segment depends on the S'th received sub-stream. Due to the random neighbor assignment, peers select the parents without considering the delays of the corresponding sub-streams. The delays of sub-streams in one segment will be out-of-order. In this way, the last (S'th) required sub-stream is very likely the transmission bottleneck. In order to decrease the startup delay, we propose a parent adjustment mechanism to replace the parent of the bottleneck sub-stream. Our approach is focused on checking closeness of the arrival time of each sub-stream which belongs to the same segment.

Let $d_j(i)$ denote the delay of data block belonging to sub-stream j in segment i .We give the definition of the delay of sub-stream in a segment as follows:

Definition 4. Within each segment, the delay of the first arrived data block is 0; the delays of following blocks are represented in the difference of arrival time; the delay of a sub-stream equals to the delay of corresponding data blocks in each segment.

Then we get the delay vector of arbitrary segment i as follows:

$$L(i) = \{d_0(i), \cdots, d_{S'}(i)\}^T$$

According to the definition 4, we compute the variance $D(d)$ of delays of sub-streams in each segment. With the value of $D(d)$, we can judge that whether or not the sub-streams in each segment have similar delays. We introduce a variance threshold δ which could be used as the criterion to determine whether or not a peer needs to adjust its parents. The decision is made following the rules below:

1) When $D(d) > \delta$, the peer needs to adjust some of the sub-streams. Let s' denote the identity of the S'th arrival sub-stream, let $\overline{d}$ represent the mean delay of the first S' sub-streams of all, and let $\Delta\overline{d}_i$ denote the difference between the delay of each sub-stream and the mean delay $\overline{d}$.
2) If the peer only has S' active parents, calculate the mean value of this difference $E(\Delta\overline{d}_i)$. If $\Delta\overline{d}_{s'} > E(\Delta\overline{d}_i)$, it implies that the sub-stream s' becomes the bottleneck of this peer, so we replace the parent providing s' with an randomly selected neighbor.
3) If the peer has more than S' active parents, not only the parent for sub-stream s' needs to be replaced as mentioned in 2), but also the parents whose sub-stream slower than s' need to be replaced.

4.3 Peer Failure

When a peer d quits, the connections related with it are canceled. So in the connectivity matrix, the d th column and d th row is filled with 0. If peer d is the substitute parent of another peer, that peer does not need to react immediately. The node needs to add a new substitute parent in the next feedback process. If peer d is the active parent of some node, that peer needs to find a new active parent from the substitute parent nodes as soon as possible.

Normally, the system search only the backup peers for the missing sub-stream, the backup peers probably fails to recover the transmission. If the peer does not have any available backup parents or the backup parent does not provide the required sub-stream in time, the peer will encounter data missing, and the successors of this peer will also suffer the same problem consequently.

When the backup parent can not provide the required resource directly, we propose to search the available resource in the current active parents, and find an adaptable match in all the available neighbors with a swap-in-turn pattern. The algorithm is shown as follows:

Algorithm 2. Peer Failure Handling Algorithm

```
input: active parents collection Vi, the failed stripe
 id :x; substitute parents collection: Sub.
output: final SubStream collection Vi.
foreach p in Sub do
  subnode= find_Substream(ak, x, p);
  if subnode not null
    add subnode to Vi; return;
  endif
end foreach
foreach ni in Vi do
  add ni to V;
end foreach
Vi=Algorithm1(V);
```

Using the example in Fig. 2, we assume the node 4 fails, and then the connection is as Fig. 2 (c). According to the set of resource vector $\left\{V_{1,6},V_{3,6},V_{4,6}V_{5,6}\right\}$, peer 6 adjusts the data source for the missing sub-stream. The changing of the sixth column of matrix C is shown in Fig. 3.

$$\begin{bmatrix}\alpha_1\\0\\\alpha_2\\\alpha_3\\\beta_1\\0\end{bmatrix}\rightarrow\begin{bmatrix}\alpha_2\\0\\\alpha_3\\0\\\alpha_1\\0\end{bmatrix}$$

Fig. 3. Parents swap-in-turns

In the sixth column, peer 6 loses the source of sub-stream 3, and none of its substitute parents could provide sub-stream 3 at this moment. But peer 3, original source of sub-stream 2, could provide sub-stream 3 also. So we set peer 3 as the source of sub-stream 3, peer 1 as the source of sub-stream 2, and change the state of peer 5 as the active parent for sub-stream 1. This switch mechanism can avoid the situation that the substitute parents can not provide some sub-streams. Then the matrix C becomes

$$C = \begin{bmatrix} 0 & \alpha_1 & \alpha_1 & 0 & \alpha_1 & \alpha_2 \\ \alpha_2 & 0 & \alpha_2 & 0 & \alpha_2 & 0 \\ \alpha_3 & \alpha_2 & 0 & 0 & \alpha_3 & \alpha_3 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \beta_3 & \beta_1 & \eta & 0 & 0 & \alpha_1 \\ \eta & 0 & \eta & 0 & \beta_3 & 0 \end{bmatrix}$$

Beside peer 6, peer 5 also changes peer 2 to the active parent for sub-stream 2 due to the quit of peer 4.

5 Simulation

In order to evaluate the parent selection and adjusting algorithm, and the peer failure handling algorithm, we developed a simulation with Peersim[15] simulator.

5.1 Methodology and Metrics

We initialize the topology in which nodes degree follows power-law. At the beginning, we insert 1000 overlay nodes to the network, and set the link latency using DS^2 [16] of Rice University. Nodes join in and leave the system randomly. The interval of both action subjects to a Poison process with mean of λ. So both the node arrival and departure actions have an average rate of $1/\lambda$. We can get different average rate by varying the value of λ.

The streaming bit rate is set to 400kbps according to the setup in CoolStreaming4, which must be satisfied at all peers during their streaming playback. Each segment has 50KB data, represents 1 second of playback, and is divided into 400 blocks. The data in each segment is redundantly encoded. For example, $RS(9,8)$ coding splits the data into 8 sub-streams, and encodes them into 9 sub-streams, each with a bit rate of 50kbps.

We define the parameters and performance metrics used in the simulation as follows: (1) Churn rate: the ratio of the number of node joining in or left to the average number of nodes in the system. Bigger churn rate means nodes change more frequently. For instance, in a system with churn rate of 2, the value of λ is 0.05, the average rates of both nodes arrival and nodes leaving are 20 per second. So in 100 seconds, the number of nodes leave the system is about 2000, which is about twice of average number of nodes in system. (2) Transmission interruption: we check every offspring node of the peer when it leaves the system. For each offspring that fails to find the substitution for the missing sub-stream in the current neighbors, we count a transmission interruption. (3) Steady degree: the ratio of the number of peers with no transmission interruption to the number of all its brothers when their active parent failed. (4) Effective transmission ratio: the ratio of the total amount of received and decoded data by all peers to the total amount of data transmitted in the network.

5.2 System Resilience

First we evaluate the resilience of this system. To compare with non-redundant coding, we use $RS(12,8)$ redundant coding. By varying the churn rate, we get the result shown in Fig. 4.

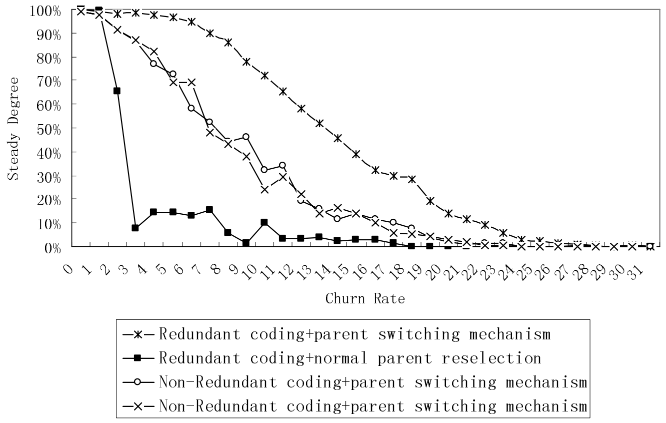

Fig. 4. The effect of redundant coding and parent swap-in-turns algorithm

We can see that both the redundant coding and parent switch algorithm is effective in improving the system resilience. The combinational use of these two mechanisms can ensure high quality for more than 80% users when churn rate is below 10.

In order to observe the sensitivity of the transmission quality to the number of sub-streams (value of S'), we varied the value of S' from 4 to 16, set the neighbor number $N = 30$, and get the result of continuity, startup latency, average hops and bandwidth usage shown in Fig. 5(a) (b) (c) (d), respectively. Here we use a non-redundant coding, that is, $S = S'$. We can see that when the number of sub-streams is between 8-16, the performance is acceptable.

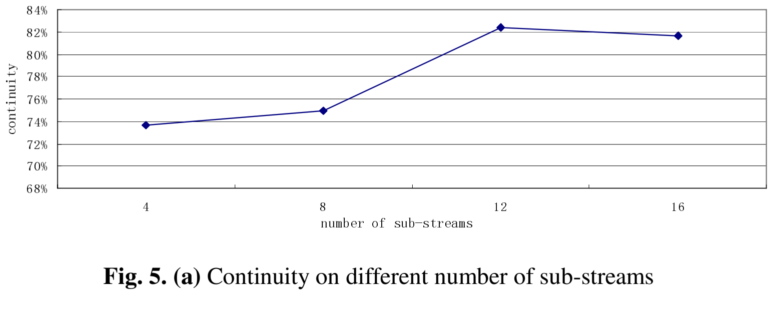

Fig. 5. (a) Continuity on different number of sub-streams

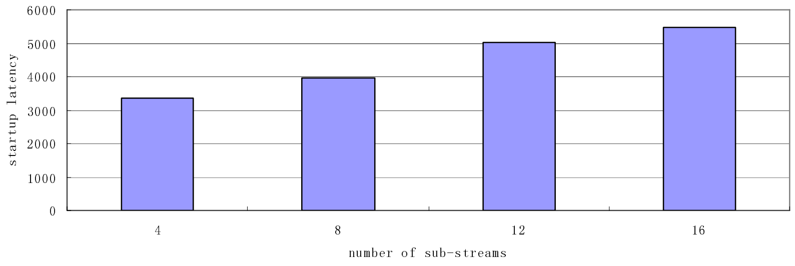

Fig. 5. (b) Startup latency on different number of sub-streams

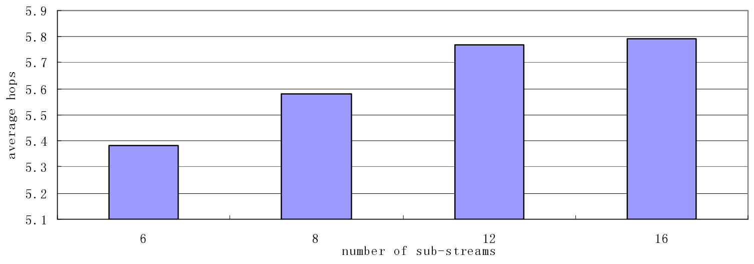

Fig. 5. (c) Average hops on different number of sub-streams

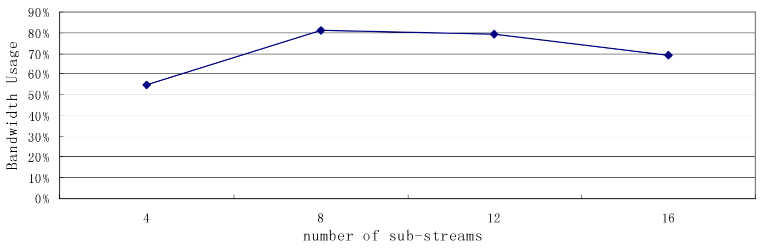

Fig. 5. (d) Bandwidth usages on different number of sub-streams

We then observe the sensitivity of transmission quality to coding redundancy by varying the churn rate and redundancy degree. The result is shown in Fig. 6. As the redundancy increase, the transmission quality becomes better.

Though redundancy improves the transmission quality, we have to answer the following question: Does redundancy cause inefficient transmission? We compared the transmission efficiency of the redundant and non-redundant coding and the result is shown in Fig. 7. When the churn rate is relatively lower, the coding schemes with higher redundancy achieve lower transmission efficiency. While when the churn rate becomes higher (beyond 8), the transmission efficiency of redundant coding is not significantly lower than that of non-redundant coding.

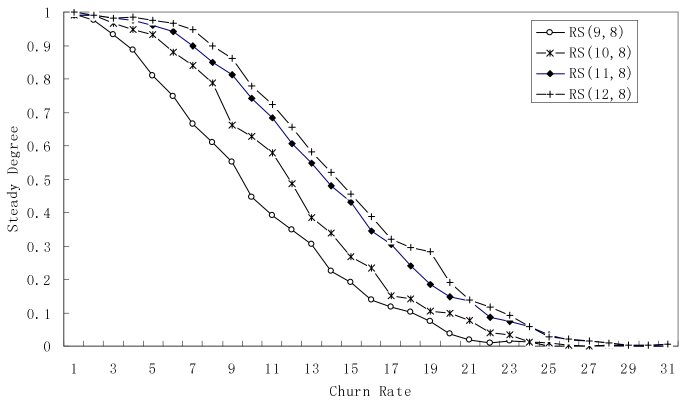

Fig. 6. Comparison of different redundancy of coding on resilience

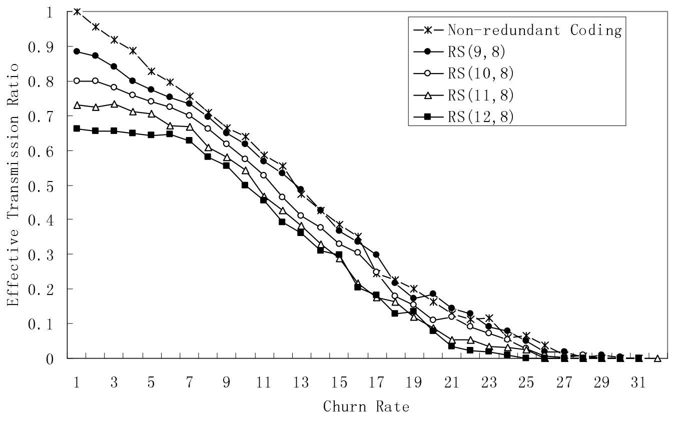

Fig. 7. Comparison of different redundancy of coding on effective transmission ratio

5.3 Startup Latency

In this section, we observe the efficiency of parent switching algorithm. We set $S' = 8$, and get the result shown in Fig. 8 by varying the delay variance threshold.

The best result by adjustment is achieved when the variance threshold is set to a medium value, like 5000ms-9000ms. Most peers that need a adjustment can achieve an improvement in delay performance and the whole network delay is reduced by 30%. When the threshold is beyond 9000ms, some peers that do need an adjustment will be missed.

We set $S' = 16$ and repeat the experiment. The result is shown in Fig. 9. Besides the conclusion drawn when $S' = 8$, we can further conclude that when the number of sub-streams becomes larger, the number of peers that need adjustment decreases, and the appropriate range of variance threshold become narrower. So in this situation, we

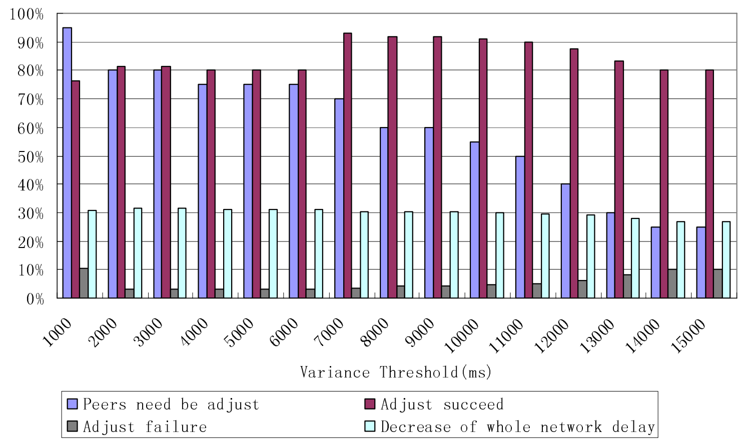

Fig. 8. Adjust effect (S'=8)

should be more cautious in determining the threshold so as to achieve the best result. For example, when the threshold is 5000ms, only 24% of peers need to be adjusted, and we can achieve almost 100% success rate. Note that the link latency parameter used in simulation is the practically measured value from the Internet, so the setup of variance threshold is meaningful in the practical systems.

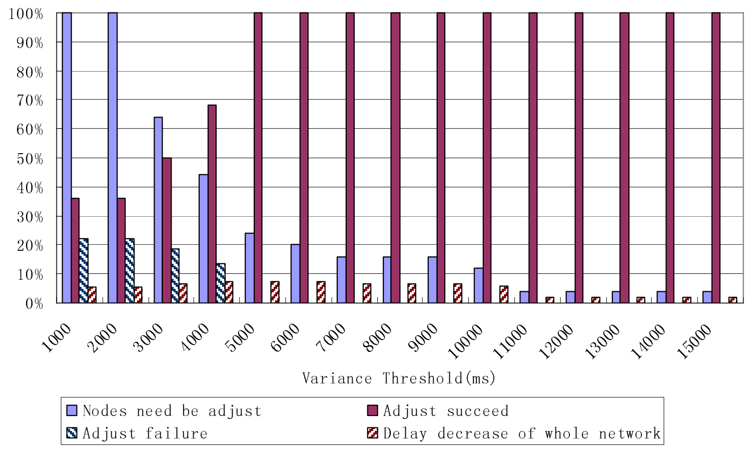

Fig. 9. Adjust effect (S'=16)

In order to find how many peers still need to be adjusted after the first adjustment, we compare the result between $S'=8$ and $S'=16$ in Fig. 10 and can see that the system with more sub-streams could be adjusted more quickly.

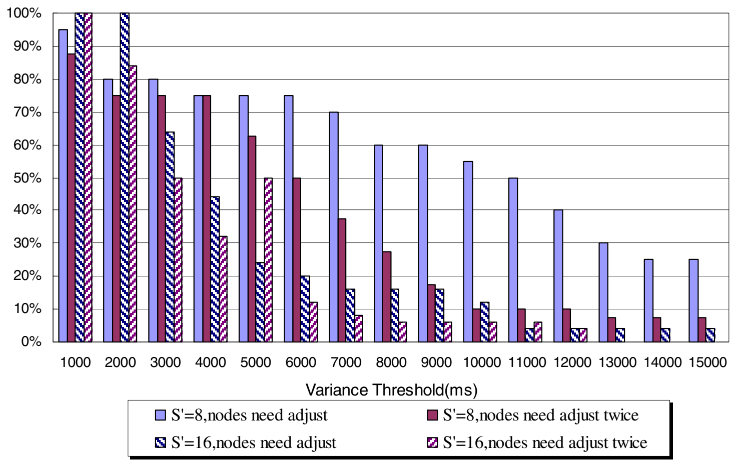

Fig. 10. Comparison of situation after the first adjust between S'=8 and S'=16

6 Conclusion

In this paper, we propose several mechanisms to improve the performance of P2P streaming systems at the different network evolution stages. Our purpose is to alleviate the influence of network dynamics to the system stability, and to decrease the peer startup delay. Simulation results show that 1) the redundant coding and the parent switching algorithm in case of node failure can effectively reduce interruption of data streams; 2) the codes with higher redundant degree can adapt to more dynamic scenario and the codes with redundancy does not significantly decrease the effective transmission ratio when network is dynamic; 3) transmission achieves higher performance when the number of substreams is between 8 and 16; and 4) the parent switching mechanism can significantly decrease the startup latency for a big proportion of peers.

References

1. Magharei, N., Rejaie, R., Guo, Y.: Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In: Proc. of IEEE INFOCOM (2007)
2. PPLive, `http://www.pplive.com/`
3. UUSee, `http://www.uusee.com/`
4. Li, B., Xie, S., Qu, Y., et al.: Inside the New CoolStreaming: Principles, Measurements and Performance Implications. In: Proc. of IEEE INFOCOM (2008)
5. Ali, S., Mathur, A., Zhang, H.: Measurement of commercial peer-to-peer live video streaming. In: Proc. Workshop on Recent Advances in P2P Streaming, Waterloo, ON, Canada (August 2006)
6. Magharei, N., Rejaie, R.: PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming. In: Proc. of IEEE INFOCOM 2007 (2007)

7. Wang, F., Liu, J., Xiong, Y.: Stable Peers: Existence, Importance, and Application in Peer-to-Peer Live Video Streaming. In: Proc. of IEEE INFOCOM 2008 (2008)
8. Liu, Z., Wu, C., Li, B., Zhao, S.: Distilling Superior Peers in Large-Scale P2P Streaming Systems. To appear in Proc. of IEEE INFOCOM 2009 (2009)
9. Koetter, R., Vardy, A.: Algebraic soft-decision decoding of Reed-Solomon codes. IEEE Transactions on Information Theory 49(11) (2003)
10. Carra, D., Cigno, R., Biersack, E.: Graph based analysis of mesh overlay streaming systems. IEEE Journal on Selected Areas in Communications 25(9), 1667–1677 (2007)
11. Kumar, R., Liu, Y., Ross, K.: Stochastic Fluid Theory for P2P Streaming Systems. In: Proc. of IEEE INFOCOM 2007 (2007)
12. Zhou, Y., Chiu, D., Lui, J.C.S.: A Simple Model for Analyzing P2P Streaming Protocols. In: Proc. of IEEE ICNP 2007 (2007)
13. Liu, S., Zhang, R., Jiang, W., Rexford, J., Chiang, M.: Performance bounds for peer-assisted live streaming. In: Proceedings of the 2008 ACM SIGMETRICS (2008)
14. Edmonds, J., Karp, R.: Theoretical improvements in algorithmic efficiency for network flow problems. Journal of ACM 19(2), 248–264 (1972)
15. PeerSim, `http://peersim.sourceforge.net/`
16. DS2, `http://www.cs.rice.edu/~bozhang/ds2/`