Dynamic Bi-Overlay Rotation for Streaming with Heterogeneous Devices

Dongyu Liu[a], Songqing Chen[a], and Bo Shen[b]

[a]George Mason University, Fairfax, VA, USA
[b]Hewlett-Packard Laboratory, Palo Alto, CA, USA

ABSTRACT

Recently Internet P2P/overlay streaming has gained increasing popularity. While plenty of research has focused on streaming performance study, it is not quite known yet on how to efficiently serve heterogeneous devices that have different limitations on display size, color depth, bandwidth capacities, CPU and battery power, than desktop computers. Although previous work[1] proposes to reuse intermediate information (metadata) produced during transcoding to facilitate runtime content adaption to serve heterogeneous clients by reducing total computing load, unbalanced resource contribution may pre-maturely exhaust the limited power of mobile devices, and adversely affect the performance of participating nodes and subsequently threaten the robustness of the whole system. In this work, we propose a Dynamic Bi-Overlay Rotation (DOOR) scheme, in which, we further consider resource consumption of participating nodes to design a dynamic rotation scheme that reacts to dynamic situations and balances across multiple types of resources on individual nodes. Based on the computing load and transcoding quality parameters obtained through real transcoding sessions, we drive large scale simulations to evaluate DOOR. The results show clear improvement of DOOR over earlier work.

Keywords: P2P/overlay Streaming, Meta-transcoding, Heterogeneity, Fairness

1. INTRODUCTION

The Internet has witnessed the sheer increase of Internet streaming traffic in recent years, among which a significant portion is delivered through various P2P/overlay streaming systems, such as PPLive,[2] PPStream,[3] TVAnts,[4] ESM,[5] AnySee,[6] and CoolStreaming.[7] Today, TVAnts[4] provides daily accesses to more than 400 channels. On the other hand, ESM,[5] built upon overlay multicast technology, has been used for over 30 different events and by over 10,000 users across the world.

The increasing popularity of P2P/overlay streaming services has attracted a wide scope of participation from heterogeneous mobile devices, owing to the technology advancements in the wireless[8] and the third generation (3G) network. Currently, the 3G wireless networks based on CDMA 2000[9] (with more than 300 million subscribers worldwide by December 2005) and UMTS[10] have been widely deployed. According to Telephia,[11] 17% of the total 206 million US mobile phone subscribers accessed the Internet and it is estimate that users worldwide will exceed 1 billion before 2010.[12]

However, the increasing participation of heterogeneous mobile devices challenge existing P2P/overlay live streaming systems in at least two aspects. First, although some previous research has considered the bandwidth heterogeneity of different peers, which is not uncommon in residence if clients use ADSL connections, it is not quite known yet on how to efficiently serve heterogeneous devices that have different quality requirements on display size, color depth, connection bandwidth while they have limited CPU cycles and battery power when compared with traditional desktops. Some research has been conducted on layered coding approaches.[13,14] But it is difficult for these approaches to provide fine-grain video qualities and flexible adaptations.[1] In addition, in practice most of existing devices do not support these codecs. Second, although previous research has considered the unfairness of node contributions due to the node positions in the system, the unfair contribution

Further author information: (Send correspondence to Songqing Chen)
Dongyu Liu: E-mail: dliu1@cs.gmu.edu, Telephone: 1 703 993 1536
Songqing Chen: E-mail: sqchen@cs.gmu.edu, Telephone: 1 703 993 3176
Bo Shen: Email: bo.shen@hp.com, Telephone: 1 650 236 2429

of different nodes in a heterogeneous environment may exhaust their limited resources quickly, resulting in the early termination of these nodes, and subsequently threatening the robustness of the system. In our previous work,[1] in order to facilitate the CPU-intensive content adaptation at runtime, we have proposed to construct an additional overlay, called metadata overlay, to instantly share intermediate results of a content adaptation process to reduce runtime content adaptation cost significantly. This extra overlay, however, incurs unfair CPU contribution among transcoding nodes and thus aggravates the concern on fairness and robustness of the system.

In this paper, aiming to more efficiently address the heterogeneity and fairness problems, we propose DOOR (Dynamic Bi-Overlay Rotation) to improve existing P2P/overlay systems so that heterogeneous devices can receive their desired streaming qualities in a fair fashion by leveraging the online content adaptation without any additional infrastructure support. Dynamic rotation is conducted on both overlays by considering the resource contribution and consumption of heterogeneous nodes, aiming to improve the robustness and fairness of the system. Based on the computing load and transcoding quality parameters obtained through real transcoding sessions, we drive large scale simulations to evaluate our proposal. The results show clear improvement of the proposed scheme over earlier work.

The remainder of the paper is organized as follows. Section 2 describes some related work. We present the overlay construction protocol and its associated data structures that are critical to rotation in Section 3. The details of dynamic bi-overlay rotation scheme are discussed in Section 4. Experimental results are presented in Section 5. We conclude in Section 6.

2. RELATED WORK

Research on P2P/overlay streaming systems has attracted considerable attention.[2,6,7,14–19] Some of these studies aim to address the robustness and unfairness problems for interior nodes. For example, Splitstream[15] distributes the forwarding bandwidth load among all the peers and can accommodate the peers with different bandwidth. Considering streaming from one source to many peers with low delay, ZigZag[18] constructs an efficient multicast tree and its end-to-end delay is kept small because the number of processing hops on the delivery path is decreased. In order to satisfy large and highly dynamic host population, Dagster[17] encourages nodes to donate more bandwidth to the system by preempting other nodes that contribute less bandwidth. However, it is not clear how a node could efficiently conduct transcoding at runtime although transcoding is proposed.

These existing research efforts mainly focus on reducing one type of resource consumption. In our previous work,[1] we have systematically evaluated the trade-offs of using runtime transcoding to handle content adaption that is required in a heterogeneous P2P system. We have further proposed to use intermediate information (metadata) produced during transcoding. Nevertheless, transcoding is still considered a computing intensive task, especially for mobile devices. The unbalanced use of various types of resources can adversely affect the performance of participating nodes and subsequently threaten the robustness of the whole system. DOOR aims to improve fairness and leads to more robust P2P systems.

3. BI-OVERLAY CONSTRUCTION

In this section, we present the bi-overlay construction, including a data overlay and a metadata overlay. On each overlay, different interchangeable sets are constructed to facilitate rotation in the streaming service in order to balance resource consumption on heterogeneous nodes.

3.1 Interchangeable Set

An interchangeable set is a node cluster in which the positions of the nodes are interchangeable. In DOOR, there are two types of interchangeable sets: Data Interchangeable Set (DIS) and Metadata Interchangeable Set (MIS). A DIS is formed by non-transcoding nodes on the data overlay and a MIS is formed by transcoding nodes on the metadata overlay. We describe their formation in section 3.1.1 and section 3.1.2, respectively.

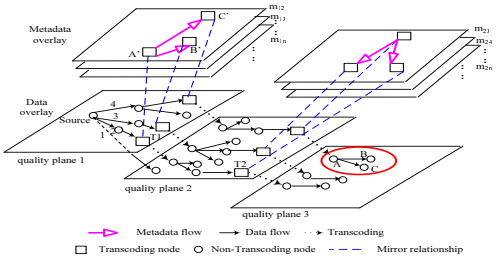

Figure 1. A bi-overlay example

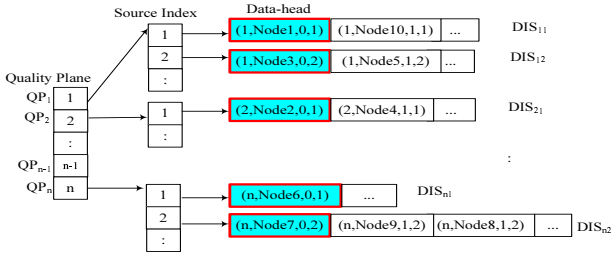

Figure 2. A QP-Array example. The fields of an element in DIS-Array are planeID, NodeID, FunctionProperty (0:regular peer, 1:transcoding-capable peer), and source index

3.1.1 DIS and QP-Array

On the data overlay, *the nodes that demand the same streaming quality form a quality plane*. Different quality planes are connected through transcoding nodes. On a quality plane, there are different subtrees. For example, in Figure 1, on quality plane 1, there are three subtrees on different branches originating from the source node. Each subtree may contain both transcoding and non-transcoding nodes. We use source index to indicate the subtree branch originating from the source node. For example, in the quality plane 2 of Figure 1, there are 4 subtrees originating from the source node. *Non-transcoding nodes from the same subtree on a quality plane forms a DIS.* For example, nodes A, B, and C on quality plane 3 in Figure 1 belong to the same DIS in the circle. In DOOR, the quality plane is managed through an array called *QP-Array*, which is a n-element array if the entire number of different quality versions is n. Figure 2 shows an example. Apparently, each element in the array is associated with one quality plane that is represented by an array called DIS-Array. This array may contain several DISs and different DISs have different source indices. The first-element in each DIS is referred as *data-head*. In Figure 2, the data-head is represented by shaded rectangles. The child nodes of a data head are stored in the deep-first order. For each element in the DIS-Array, there are four fields: planeID, NodeId, FunctionProperty (regular/transcoding capability), and source index. For a child node, the source index of that node is the same as that of its parent or is assigned by the source node if its parent is the source node. Only nodes in the same interchangeable set can participate the rotation as to be discussed in section 4.3.

3.1.2 MIS and TMatrix

On the metadata overlay, similarly, based on the transcoding input and output versions, we have different transcoding planes. For example, in Figure 1, transcoding nodes A', B', and C' perform identical transcoding and thus are on the same transcoding plane. With a total of n different quality versions, there are at most $\frac{1}{2}n \times (n-1)$ different transcoding planes without counting the impractical cases (transcoding from a lower quality stream to a higher quality stream). We use a upper triangle matrix of size $n \times n$, denoted by *TMatrix*, to represent the transcoding planes. Figure 3(a) shows an example of TMatrix. All transcoding nodes belonging to the same transcoding plane form an MIS if their transcoding processes have the same input and output. Figure 3(b) shows an example of column j's MIS lists in TMatrix. Similar to DIS-Array, the first `non-null` element is referred as *meta-head*. In a MIS, the meta-head performs full transcoding while other nodes perform the same meta-transcoding.

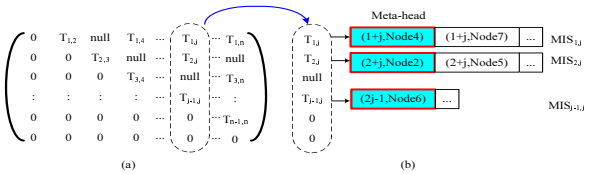

Figure 3. A TMatrix example on the Metadata overlay. The fields of an element are planeID, and NodeID

3.2 Bi-Overlay Construction

The overlay is constructed and adjusted upon node arrivals and departures. We present the node arrival and departure processes in section 3.2.1 and section 3.2.2, respectively.

3.2.1 Node Arrival

When a node arrives, it first contacts the source node or a bootstrap node with its quality requirement. In DOOR, the quality requirements of node i are denoted as a tuple of $< fr_i, br_i, cd_i >$, where fr_i, br_i, and cd_i represent the frame rate, bit rate, and color depth requirements, respectively. In the following context, we always map the quality tuple requirement to a quality index and refer to as a quality version. We assume that if there are n quality versions, the best quality version (original version) is 1 and the lowest quality version is n.

- **Case 1 – no transcoding required and quality plane:** First, the joining node contacts the source node or a bootstrap node to obtain a candidate parent list containing the nodes that can provide the exact desired object quality. The joining node selects one of them with the smallest height (from the source node) in the tree. In this case, only the data overlay is updated. At the same time, this node needs to join a particular quality plane. Suppose the joining node requests streaming quality i and its candidate parent's source index is j,
  - The node contacts the data-head of DIS_{ij}. If the data-head can accept the joining node, it adds the joining node to its list. Otherwise, the node contacts another candidate parent with a different source index.
  - If all the candidate parents have been tried without any success and the source node has enough bandwidth, the source node will assign a source index j' and the joining node becomes the data-head of $DIS_{ij'}$. The NodeID and source index are recorded in $DIS_{ij'}$.
- **Case 2 – transcoding required and transcoding plane:** If the desired object version does not exist, the joining node must find a parent node that is able to do transcoding for it. In this case, both the metadata overlay and the data overlay need to be updated. At the same time, the parent node needs to join a particular transcoding plane. Based on whether there is available metadata for transcoding,
  - **metadata available and meta-transcoding**: The joining node checks TMatrix from the source node by examining the j^{th} column (note that quality version 1 is the highest quality version). The possible parent candidates are from $T_{1,j}$ to $T_{j-1,j}$ (a total of $j-1$ candidates). The joining node starts to probe nodes that are receiving version t ($0 < t < j$), if $T_{t,j}$ is not `null` in TMatirx. If it is for bit rate transcoding, t is selected where $j-t$ is the largest, otherwise, the smallest.[1]

    If the joining node can find a parent (denoted as P), P needs to join the metadata overlay by contacting meta-head of $T_{t,j}$. If P can find a parent node to join in the metadata subtree, P will receive the shared metadata $m_{t,j}$ within the metadata subtree and can share it with other incoming nodes. If the meta-head cannot accept P into the metadata subtree, the joining node need to probe another `non-null` element in the j^{th} column of TMatrixand repeat above joining process. If all `non-null` nodes have been tried, go to next step.
  - **metadata unavailable and full transcoding**: If a parent is not found in the previous step, the joining node must look for a parent that can perform full transcoding for it. To minimize the total transcoding overhead, the joining node starts to probe nodes that are receiving version t ($0 < t < j$), if $T_{t,j}$ is `null` in TMatrix. The joining node probes nodes where `t-j` is the smallest or the largest, depending on the type of transcoding as discussed above. If a node P accepts the joining node as a child, P starts a full transcoding process from scratch. Accordingly, P joins the metadata overlay and updates the element $T_{t,j}$ in TMatrix if node P needs to do transcoding from quality t to quality j and node P becomes the meta-head of $T_{t,j}$. The element $T_{t,j}$ of TMatrix is changed from `null` to (t, P). At the same time, the arriving node joins the data overlay with source index i. The data-head of element DIS_{ji} accepts the joining node as its member and updates the DIS_{ji} list.

  If a parent node still can not be found and the version j is not the lowest quality version, the node starts to request version $j+1$ following the above procedure. Otherwise, the joining request is rejected.

3.2.2 Node Departure

In P2P/overlay streaming, participating nodes may leave the system at any time. When a node departs, DIS-Array and TMatrix must be updated to reflect the change on the data overlay and metadata overlay. The procedures are as follows.

- **Case 2.1:** If the departing node is a leaf node and is receiving version j, assuming its parent is P,

  (1) **DATA-DEPART**: if P is not a transcoding node, remove the departing node from P's children list and the departing node leaves the data tree as follows:

  - If the departing node is the data-head of DIS_{jk} (k is the source index of departing node) and it is the only node in the list of DIS_{jk}, remove the departing node from the list and set DIS_{jk} to $null$.
  - If the departing node is the data-head and is not the only node in the list of DIS_{jk}, the data-head selects the node with the largest bandwidth as the new data-head. In addition, the corresponding data-head element of DIS_{jk} in DIS-Array is updated.

  (2) **META-DEPART**: if P is a transcoding node conducting transcoding from quality i to quality j, P leaves the metadata subtree $T_{i,j}$ as follows:

  - If P is the meta head and the only node in the subtree, the element $T_{i,j}$ of TMatrix is set to $null$.
  - Otherwise, P selects the node with the largest available bandwidth to be the new meta-head in the transcoding plane and the meta-head element of $T_{i,j}$ in TMatrix is set to the newly selected node.

  In addition, the departing node leaves the data tree following the `DATA-DEPART` algorithm.

- **Case 2.2:** If the departing node is neither a leaf node nor a transcoding node, the children of the departing node need to find a new parent. The parent of the departing node is a good candidate if the parent is not a transcoding node and has enough bandwidth. If the children of the departing node cannot find an appropriate parent, the children of the departing node need to perform a rejoin process and the rejoin-process may change the transcoding plane but not the quality plane because the children do not change their quality version requirement. If the parent (denoted as P) of the departing node is a transcoding node, P leaves the metadata subtree $T_{i,j}$ following the **META-DEPART** algorithm and the corresponding transcoding plane is updated. If the departing node is on the DIS list of quality plane j, the departing node is removed from the system following the **DATA-DEPART** algorithm.

- **Case 2.3:** If the departing node receiving version i is not a leaf node and is a transcoding node, the adjustment must be done on both the corresponding quality plane and the transcoding plane. Assume the departing node D is in the metadata tree $T_{i,j}$ and the children of node D need quality version j. The departing node leaves the metadata subtree $T_{i,j}$ and the transcoding plane is updated. Following the **META-DEPART** algorithm, the departing node leaves the metadata subtree. If the departing node is on the DIS list of quality plane j, node D is removed from the system according to the **DATA-DEPART** algorithm. In addition, the children of node D on the data overlay need to find another transcoding parent having the same metadata in the metadata subtree that departing node is originally in. If they cannot find transcoding parents through metadata subtree $T_{i,j}$, they need to rejoin the system.

4. DYNAMIC BI-OVERLAY ROTATION

Although metadata sharing through the metadata overlay can significantly reduce the total amount of CPU cycles demanded for online content adaptation, the overhead would be charged on fixed nodes, which not only is unfair, but also may exhaust the resources on these nodes quickly and leads to the collapse of the system. In addition, on the data overlay, different nodes may contribute different amounts of bandwidth due to the different numbers of children they serve. In order to deal with these heterogeneity and fairness issues, we propose dynamic bi-overlay rotation. We present rotation criteria for selecting candidate nodes, when to rotate and how to rotate in Section 4.1, Section 4.2, and Section 4.3, respectively.

4.1 Rotation Criteria

The rotation on the metadata overlay aims to reconcile CPU and bandwidth consumption for online content adaptation, while the rotation on the data overlay is to balance the bandwidth contribution of different nodes. Thus, to select nodes for rotation, it must be based on nodes' contribution on different overlays.

4.1.1 Node contribution on the data overlay

On the data overlay, different nodes make different contributions due to the different numbers of children they serve. To characterize the bandwidth contribution of a node, we define Contribution Index (CI) to reflect how much bandwidth the node contributes to the system. The CI of node i on the data overlay is defined as

$$CI_{bi} = \sum_{1}^{K} ub_{ik} \times t_{ik}. \tag{1}$$

where t_{ik} is node i's uploading service time for child k, K is the total number of children that node i serves on the data subtree, and ub_{ik} is the bandwidth consumption of node i for child k.

4.1.2 Node contribution on the metadata overlay

Different with data overlay, the Contribution Index of a node in metadata overlay includes two portions: CPU and bandwidth. The amount of the CPU cycles is dependent on full transcoding or meta-transcoding the node performs, while the bandwidth consumption on the metadata overlay is dependent on the size of metadata and the number of children the node serves.

Accordingly, without considering the scarcity of different resources, the contribution index of node i is $CI_i = a \times CI_{ci} + b \times CI_{mbi}$, where a and b represent the normalization factors for the CPU and bandwidth consumption. If there are k nodes in a metadata subtree, $a = 1/\sum_1^k CI_{ci}$ and $b = 1/\sum_1^k CI_{mbi}$. The CI_{mbi} is calculated by

$$CI_{mbi} = \sum_{1}^{K} umb_{ik} \times t_{ik}. \tag{2}$$

where t_{ik} is node i's uploading service time for child k, K is the total number of children that node i serves in the metadata subtree, and umb_{ik} is the metadata bandwidth load of node i for child k. In addition, CI_{ci} is calculated by: $CI_{ci} = c_{iv1,iv2} \times t_i$, where $c_{iv1,iv2}$ represents the node i's transcoding quality distance from version $v1$ to version $v2$ and t_i is the time during which node i is an active transcoder.

4.1.3 Fragile edge and rotation pair selection

Based on the definitions of CI in section 4.1.1 and 4.1.2, CI of each node on these overlays can be calculated periodically. Note that the contribution of each node is accumulative from the beginning. Based on their contributions, the protocol selects pairs of nodes as candidates to exchange their positions in the tree. The selection works as follows. First, with the contribution of each node, each link is tagged with the absolute contribution difference value of the two neighboring nodes in each DIS and MIS. Second, the data head or meta head selects the link with the largest difference as a *fragile edge*. The two nodes connected by a fragile edge form a rotation pair. Thus, each rotation would only affect a small number of nodes. In addition, a node having contributed a lot can exchange its position with a node having contributed little in a recursive fashion to balance their resource consumption.

4.2 Rotation Schedule

Having identified the rotation pairs, we need to determine when to perform rotation. The frequency of rotation should be considered carefully. If the subtree is rotated too frequently, it keeps the overlays busy with adjusting new neighborhood relationships. If it is rotated too slowly, the resource contributions of different nodes could be very skew. Thus, an appropriate rotation interval should be determined and the interval selection can be contribution-driven or time-driven. A time-driven approach is straightforward. That is to rotate the sub-tree with a fixed time interval I_r. However, the contribution is not only affected by the time period, it is also affected

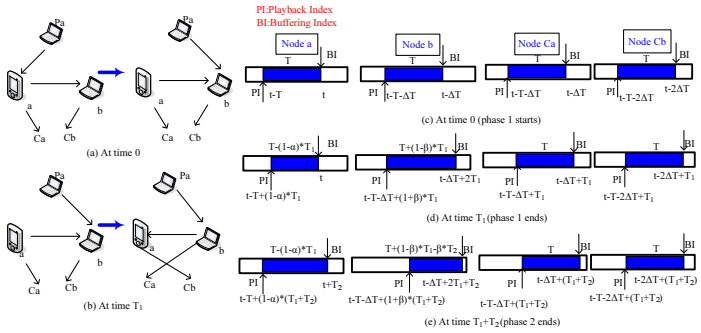

Figure 4. The rotation process on the data overlay

by the node arrivals and departures since these dynamics would affect the number of children each node serves. Therefore, our interval is determined by considering both aspects. That is, by monitoring the node arrivals and departures to the sub-tree, the data-head on the data overlay and the meta-head on the metadata overlay can determine the rotation interval dynamically as follows. Each arrival or departure is taken as an event. If the event rate, denoted as E_r, is less than E_α, a system threshold, the rotation interval is I_r. Otherwise, the rotation interval is $I_r \times E_r$. The purpose of such a design is to give extra time for nodes in a metadata subtree to adapt to the dynamics.

4.3 Rotation Protocol

Although rotation can balance the resource consumption among participating nodes, it is important to guarantee that the rotation would not disrupt the streaming delivery and would not degrade user perceived experience.

For these objectives, we leverage a technique that has been commonly used in practice.[20] That is, for a playback, if the playback speed is slightly changed (increased or decreased) in a certain range, users would not perceive the difference. This technique has been used by TV channels to insert additional advertisements during the normal broadcasting. The practice indicates that this range should be controlled under 20%. With the support of this technique, we carefully manage the buffers of the involved nodes for rotation to guarantee the unchanged size of the buffers before and after rotation. In the mean time, we aim to minimize the total time spent on rotation.

4.3.1 Data overlay rotation and buffer management

Rotation is performed simultaneously for the node pairs connected by fragile edges. For a pair of nodes (a, b) to be rotated, assume node a is the parent of node b on the subtree. Without losing the generality, we assume that the child set of node a (except b) is clustered and denoted as C_a, and the child set of node b is denoted as C_b and the parent of node a is denoted as P_a.

We assume that by default, each active node maintains a playback buffer that can sustain T second playback at the normal playback speed v. We also assume there is a playback delay of ΔT between node a and node b. This delay applies to any parent and child pair. Our rotation procedure consists of two phases and we assume the duration for them is T_1 and T_2, respectively. Phase 1 starts at time 0 and ends at time T_1 and phase 2 starts at time T_1 and ends at time $T_1 + T_2$. Based on these assumptions, the initial buffer size of node a and b is $T \times v$ and the stream content in related nodes before phase 1 is shown in Figure 4(c). Our goal is thus to guarantee that after the rotation, the buffer size of the involved nodes remains the same based on the following procedure:

1. Figure 4(a) shows the original situation when time is 0 (phase1 starts). When rotation starts, node P_a stops streaming to node a and starts to stream to node b. Simultaneously, node a 1) reduces its playback rate by α percentage and thus its playback rate is $(1-\alpha) \times v$; 2) continues to stream to node b. At the same time, node b increases its playback speed by β percentage and thus its playback rate is $(1+\beta) \times v$. Hence, the buffering speed of node a is 0. The buffering speed of node b is $2 \times v$ because it receives streaming from node a and node P_a simultaneously. This phase continues till T_1. When it is T_1, the buffer size of node a

is decreased from $T \times v$ to $(T - (1-\alpha) \times T_1) \times v$ and the buffer size of node b is increased from $T \times v$ to $(T + (1-\beta) \times T_1) \times v$. While for node C_a and C_b, as long as T_1 is less than ΔT, their buffer size is not changed. The index of playback and buffering and the buffers of different nodes after phase 1 are shown in Figure 4(d).

2. Figure 4(b) shows the rotation situation when time is T_1 (the time when phase 2 starts). At time T_1, node b starts to stream to node a and node C_a. Node a starts to stream to node C_b. At the same time, node a stops streaming to node C_a and node b stops streaming to node C_b. From time T_1 to T_2, the buffer size of node a is increased to $(T - (1-\alpha) \times T_1 + \alpha \times T_2) \times v$ and the buffer size of node b is decreased to $(T + (1-\beta) \times T_1 - \beta \times T_2) \times v$.

Through the entire rotation procedure, the buffering and playback speed of node C_a and C_b are not changed. Thus, their buffer sizes are not affected by the rotation.

Assume for node a and node b, they resume the normal playback after T_{2a} and T_{2b} in Phase 2. For node a, at time $T_1 + T_{2a}$, its buffer size should be equal to or larger than $T \times v$, the buffer size before the rotation. Thus,

$$(T - (1-\alpha) \times T_1 + \alpha \times T_{2a}) \times v \geq T \times v => T_{2a} \geq \frac{1-\alpha}{\alpha} \times T_1 \tag{3}$$

For node b, we also expect that when it is $T_1 + T_{2b}$, its buffer size should be equal to or larger than $T \times v$ and node b can resume normal playback speed v. That is

$$(T + (1-\beta) \times T_1 - \beta \times T_{2b}) \times v \geq T \times v => T_{2b} \leq \frac{1-\beta}{\beta} \times T_1. \tag{4}$$

To minimize $T_1 + T_2$ (the duration of the entire rotation), where $T_2 = max\{min\{T_{2a}\}, max\{T_{2b}\}\}$, we can get

$$T_1 + T_2 = \begin{cases} \frac{1}{\alpha} \times \Delta T & \text{if } \alpha \leq \beta, \\ \frac{1}{\beta} \times \Delta T & \text{if } \alpha > \beta. \end{cases} \tag{5}$$

As we have mentioned, α, β should be controlled under 20%. The rotation time thus depends on ΔT, which is one-hop delay on the overlay. It varies depending on the underlying physical topology. However, with an average estimate of 100 ms of ΔT, the rotation can be finished in 0.5 seconds if the α and β are set as 20%.

Through a similar analysis we have performed on the data overlay rotation, assume that there is a delay of ΔT_m between nodes on the metadata overlay, we can derive that the total rotation duration on the metadata overlay, is equal to $\frac{\Delta T_m}{\beta}$ or $\frac{\Delta T_m}{\alpha}$. Thus, the minimization of this rotation duration is similar as what we have performed for the data overlay.

5. PERFORMANCE EVALUATION

In this section, we first conduct bit rate reduction and frame rate reduction based on our implemented transcoder to get real parameters, which are then used in ns2 simulations to evaluate a large scale overlay streaming system.

5.1 Real Experiments and Simulation Setup

We have conducted both bit rate reduction and frame rate reduction on our transcoder with a 1000-second video clip, which has a data rate of 500 kb/s and a frame rate of 30 frame/second (denoted as f/s hereafter). The video clip we have used is "foreman" coded by H.263. Four other versions have been transcoded from the original version for each type of transcoding. In bit rate reduction, these versions have 400 kb/s to 100 kb/s with a difference of 100 kb/s in successive versions. If we assume CPU consumption for a full transcoding session is 1 unit, meta-transcoding only demands 0.4 units for transcoding to these four versions with a metadata size of 10% of the original video file size regardless of different versions. In frame rate reduction, the other four versions are 500 kb/s and 400 kb/s at 30 f/s, 300 kb/s and 200 kb/s at 15 f/s, and 100 kb/s at 5 f/s. If CPU load for a full transcoding session from 30 f/s to 15 f/s takes 1 unit, the CPU load for full transcoding from 30 f/s to

5 f/s and from 15 f/s to 5 f/s takes 0.44 units and 0.36 units, respectively. While for meta-transcoding, the corresponding CPU loads are 0.5, 0.27, and 0.19 units and the corresponding metadata size is 20%-25% of the original video file size for transcoding between different frame rates, depending on the source bit rate and the target frame rate.

With the setting of these parameters in ns2, we set to evaluate *DOOR* by comparing it with other three schemes. Among them, *No-Transcoding* represents system in which no node in the system can perform transcoding but the source node can serve precoded versions. A joining request is accepted if the requested or a lower quality version is available. *Full-Transcoding* represents the system in which all transcoders perform full transcoding without metadata. *Meta-Transcoding* indicates the system with the metadata overlay, but without dynamic bi-overlay rotation, which is the scheme we proposed in Ref. 1.

In our experiments, there are 2000 nodes dynamically joining and leaving the system. The source node has all the different versions and its uploading bandwidth is 5 Mb/s. The distribution of the uploading bandwidth for other nodes is as follows: 5Mb/s (11%), 2Mb/s (3%), 896kb/s (9%), 384kb/s (21%) and 256kb/s (56%), which is based on the experimental conditions used in Ref. 21. The CPU constraints of nodes are randomly distributed from 1 unit/second to 2 units/second. Nodes with 5Mb/s and 2Mb/s uploading bandwidth randomly request quality versions 1 to 3, and nodes with other bandwidth randomly request quality versions 1 to 5. The node arrivals follow a Poisson distribution with the mean arrival rate as 1 request per second. The maximum arrival interval is 10 seconds. A node stays in the system receiving streaming for a duration ranging from 250 seconds to 1000 seconds. For rotation, the default rotation interval is set as 20 seconds. The system threshold E_α is set as 0.25.

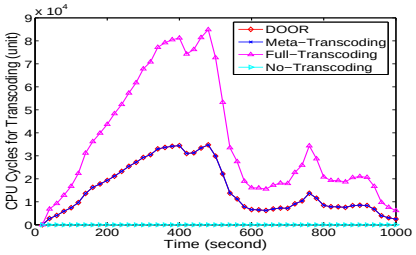

Figure 5. CPU cycles consumed (bit rate transcoding)

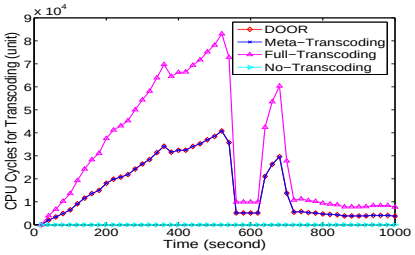

Figure 6. CPU cycles consumed (frame rate transcoding)

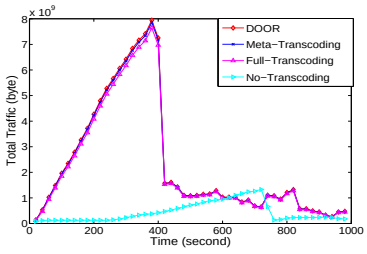

Figure 7. Total traffic (bit rate transcoding)

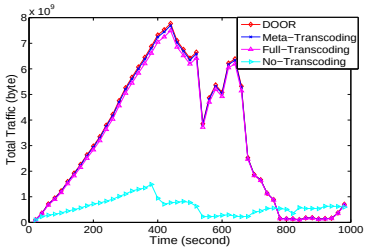

Figure 8. Total traffic (frame rate transcoding)

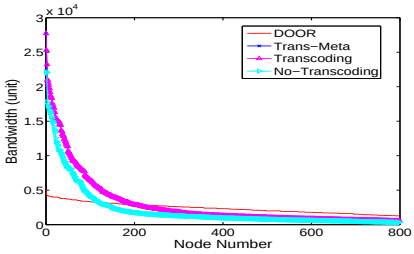

Figure 9. Data overlay rotation (bit rate transcoding)

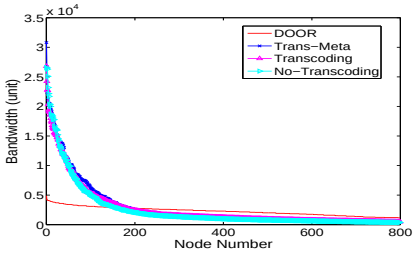

Figure 10. Data overlay rotation (frame rate transcoding)

5.2 Total CPU and Bandwidth Consumption

First, the total CPU and bandwidth consumptions of different schemes are studied. Figure 5 and Figure 6 show the normalized CPU consumption for bit rate and frame rate transcoding, respectively. The CPU load on the y-axis is computed every 20 seconds. As shown in both figures, *No-Transcoding* does not require any CPU cycles since no transcoding is performed. Both *Meta-Transcoding* and *DOOR* consume the same amount of CPU cycles since the difference between these two schemes is not on the total CPU consumption, but on CPU consumption on individual node, which will be evaluated later. Figure 5 and Figure 6 show that with the metadata overlay to share the metadata, both *DOOR* and *Meta-Transcoding* significantly outperform *Full-Transcoding*. On average,

metadata assisted schemes can save 58% and 51% CPU load comparing with *Full-Transcoding* for bit rate transcoding and frame rate transcoding, respectively.

Although the metadata overlay in *DOOR* and *Meta-Transcoding* can help reduce the total CPU demand during transcoding, there is traffic overhead, which is used for metadata sharing. Figure 7 shows the total traffic for bit rate transcoding, and Figure 8 shows the corresponding result for frame rate transcoding. The traffic amount on y-axis is summed every 20 seconds. Both figures indicate a significant traffic amount increase for three transcoding enabled schemes when compared with *No-Transcoding*. This is because with transcoding enabled, all three transcoding schemes can serve more clients with better streaming quality than *No-Transcoding*. Because of the traffic overhead due to the rotation process, *DOOR* consumes more bandwidth than *Meta-Transcoding*. Both Figure 7 and Figure 8 show that the rotation cost is only up to 1.3% of the total traffic amount in *Meta-Transcoding*.

This group of evaluation results shows that with relatively small overhead, the metadata overlay can effectively reduce CPU consumption for online content adaptation.

5.3 CPU and Bandwidth Consumption on Individual Nodes

After evaluating the total amount of resources consumed in different schemes, we study resource consumption on individual nodes to verify whether dynamic rotation can indeed balance their resource consumption.

First, the bandwidth contribution of each nodes is studied. As the major bandwidth consumption is on the data overlay, we focus on this portion to study whether the fairness is improved with our dynamic rotation scheme. Figure 9 and Figure 10 show the peer bandwidth contribution for bit rate transcoding and frame rate transcoding, respectively. Note that in both figures, only the first 800 largest bandwidth-contributors on the data overlay are shown and the contribution values of these nodes in these figures are sorted in descending order.

As shown in the Figure 9, compared with *No-Transcoding*, *Full-Transcoding*, and *Meta-Transcoding*, *DOOR* can significantly balance the bandwidth consumption among peers. Overall, nodes in other schemes have their contributions varying from 540 to 27700 with a standard deviation of 4056, while in *DOOR*, it is in the range from 1280 to 5100 with a standard deviation of 745.

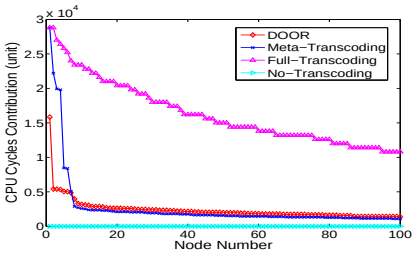

Figure 11. Metadata overlay rotation (bit rate transcoding)

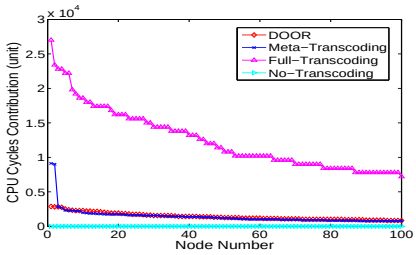

Figure 12. Metadata overlay rotation (frame rate transcoding)

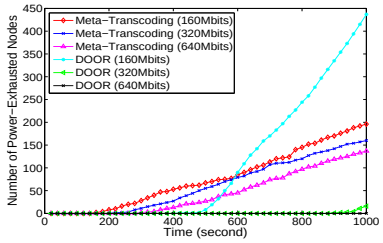

Figure 13. Power-exhausted nodes on the data overlay (bit rate transcoding)

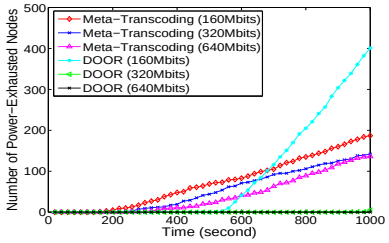

Figure 14. Power-exhausted nodes on the data overlay (frame rate transcoding)

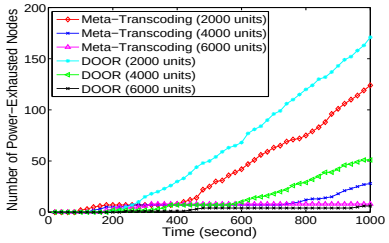

Figure 15. Power-exhausted nodes on the metadata overlay (bit rate transcoding)

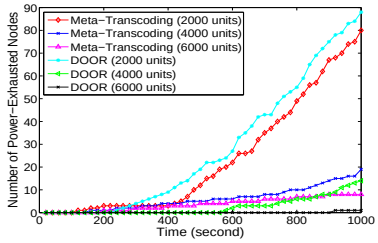

Figure 16. Power-exhausted nodes on the metadata overlay (frame rate transcoding)

Second, we evaluate CPU contribution of each peer to the metadata overlay. Figure 11 and Figure 12 show the CPU cycles contributed to the system for bit rate transcoding and frame rate transcoding. In these two figures, only the first 100 largest CPU-contributors are shown. In addition, the values of the these nodes in these figures are sorted in descending order. Clearly, among all the schemes, peers in *DOOR* contribute more evenly than in *Full-Transcoding* and *Meta-Transcoding*. For example, in Figure 12, for *Full-Transcoding*, the peer contribution ranges from 7200 to 27000 with a standard deviation of 4415, while in *Meta-Transcoding*, it is ranged from 780 to 9120 with a standard deviation of 1188. Neither of these two are comparable to *DOOR*, in which the range is from 840 to 2880 with a standard deviation of 517.

These results show that dynamic bi-overlay rotation, although comes with some cost, can effectively improve the fairness of peer contributions in both overlays.

5.4 Number of Nodes with Exhausted Power

DOOR is designed to enable transcoding to better serve heterogeneous devices. Normally mobile devices come with limited battery power. In this section, we evaluate how many nodes are forced to leave the streaming system once their power is exhausted. If one power-exhausted node leaves the system, the children of this node find their new parents as if the power-exhausted node leaves the system using our aforementioned peer departure algorithm. The condition is based on either the total amount of uploading bandwidth consumed or the total amount of CPU cycles consumed. That is, in the simulation, if a predetermined amount of bandwidth or CPU cycles is used up, the device is assumed dead and has to leave the system.

Figure 13 and Figure 14 show the number of nodes with exhausted power for bit rate transcoding and frame rate transcoding on the data overlay, respectively. The y-axis, the number of nodes with exhausted power, is summed every 20 seconds. We only compare *DOOR* and *Meta-Transcoding* with different thresholds due to the limited power for uploading to others. As shown in the figures, when we set the uploading capacity threshold as 160 Mbtis, 320 Mbits, and 640 Mbits, the trends are similar for bit rate reduction and frame rate reduction. In particular, compared to *Meta-Transcoding*, *DOOR* significantly reduces the number of power-exhausted peers when the thresholds are 320 Mbits and 640 Mbits by balancing the bandwidth usage among the peers. When the threshold is 640 Mbits, at time 1000 second, the number of power-exhausted nodes in *DOOR* is zero, while it is 140 in *Meta-Transcoding* (recall a total of 2000 nodes). However, when the threshold is 160 Mbits, the number of exhausted peers in *DOOR* is larger than that in *Meta-Transcoding*. The reason is that the available resources are evenly consumed and cannot meet the bandwidth requirement for most of peers. On the other hand, with skewed peer contributions in *Meta-Transcoding*, a smaller number of power-exhausted peers saved some other peers for a longer time. Although *DOOR* has this disadvantage, peers in *DOOR* begin to leave at 500 second while peers in *Meta-Transcoding* start to leave as early as at 150 second after their power is exhausted. Thus, there is a trade-off between whether the dynamic rotation is applied or not when the total available resource is very limited.

Similar experiments have been conducted on the metadata overlay for the CPU resources. Figure 15 and Figure 16 show the number of power-exhausted nodes for bit rate transcoding and frame rate transcoding on metadata overlay, respectively. Again, the value on the y-axis is summed every 20 seconds. We evaluate three CPU contribution thresholds: 2000, 4000, and 6000 CPU units. In Figure 16, the number of power-exhausted peers in *DOOR* is less than that in *Meta-Transcoding* when CPU contribution threshold is set as 4000 and 6000 units, and the gap increases with time. However, when CPU threshold is 2000 units, the number of power-exhausted peers in *DOOR* is more than that in *Meta-Transcoding* because the available CPU cycles of most of the peers cannot meet the requirement of average CPU resource requirement. Compared with Figure 16, in Figure 15, only when the CPU contribution threshold is 6000 units, the number of power-exhausted peers in *DOOR* is less than that in *Meta-Transcoding*. The reason is that bit rate reduction demands more CPU cycles than the frame rate reduction.

These results show that *DOOR* performs very well with moderate supply of resources, while a trade-off could be manipulated when severe resource shortage is present, which we leave for future work.

6. CONCLUSION

The increasing number of heterogeneous clients using all kinds of mobile devices to participate P2P/overlay streaming has brought heterogeneity and fairness problems. In this paper, in order to serve heterogeneous clients and improve the robustness and fairness of these systems, we propose a dynamic bi-overlay rotation scheme, which features efficient metadata sharing to reduce the total demanded resources for online content adaptation and dynamic peer rotation in both metadata and data overlays to balance the peer contributions of CPU and bandwidth resources. Performing real transcoding, we have obtained practical parameters and evaluated our proposed scheme in a large system. The results show that our proposed scheme is effective in balancing CPU and bandwidth consumption on individual nodes.

7. ACKNOWLEDGMENTS

We would like to thank our shepherd Kang Li and anonymous reviewers for their helpful comments on this paper. The work is supported by NSF grants CNS-0509061 and CNS-0621631.

REFERENCES

1. D. Liu, E. Setton, B. Shen, and S. Chen, "Pat: Peer-assisted transcoding for overlay streaming to heterogeneous devices," in *Proceedings of ACM NOSSDAV*, (Urbana-Champaign, IL), Jun. 2007.
2. "PPlive." http://www.pplive.com.
3. "PPStream." http://www.ppstream.com/.
4. "TVants." http://www.tvants-ppstream.com/.
5. Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," in *Proceedings of ACM SIGMETRICS*, (Santa Clara, CA), June 2000.
6. X. Liu, H. Jin, Y. Liu, L. Ni, and D. Deng, "Anysee: Peer-to-peer live streaming," in *Proceedings of IEEE INFOCOM*, (Barcelona, Spain), Apr. 2006.
7. X. Zhang, J. Liu, B. Li, and T. Yum, "Coolstreaming/donet: A data-driven overlay network for efficient live media streaming," in *Proceedings of IEEE INFOCOM*, (Miami, FL), Mar. 2005.
8. "New wireless standard." http://www.ieee802.org/11/.
9. "CDMA2000." http://www.umtsworld.com/technology/cdma2000.htm.
10. "UMTS." http://www.umtsworld.com/technology/overview.htm.
11. "Telephia." http://www.telephia.com/.
12. "Mobile User." http://www.dmwmedia.com/news/2006/09/15/report-global-3g-subscribers-to-reach-1-billion-by-2010.
13. V. Goyal, "Multiple description coding: Compression meets the network," in *IEEE Signal Processing Magazine*, **18**, September 2001.
14. V. Padmanabhan, H. Wang, and P. Chou, "Resilient peer-to-peer streaming," in *Proceedings of IEEE ICNP*, (Atlanta, GA), Nov. 2003.
15. M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth content distribution in a cooperative environment," in *Proceedings of 2nd IPTPS*, (Berkeley, CA), Feb. 2003.
16. M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "Promise: Peer-to-peer media streaming using collectcast.," in *Proceedings of ACM Multimedia*, (Berkeley, CA), Nov. 2003.
17. W. Ooi, "Dagster: Contributor-aware end-host multicast for media streaming in heterogeneous environment," in *Proceedings of ACM/SPIE MMCN*, (San Jose, CA), Jan. 2005.
18. D. Tran, K. Hua, and T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," in *Proceedings of IEEE INFOCOM*, (San Francisco, CA), Mar. 2003.
19. D. Liu, S. Chen, and B. Shen, "Amtrac: Adaptive meta-caching for transcoding," in *Proceedings of ACM NOSSDAV*, (Newport, RI), May 2006.
20. H. Jiang and S. Jin, "Nsync: Network synchronization for peer-to-peer streaming overlay construction," in *Proceedings of ACM NOSSDAV*, (Newport, RI), May 2006.
21. E. Setton, J. Noh, and B. Girod, "Congestion-distortion optimized peer-to-peer video streaming," in *Proceedings of ACM ICIP*, (Portland, OR), Oct. 2006.