Layered Peer-to-Peer Streaming [*]

Yi Cui, Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
{*yicui, klara*}@cs.uiuc.edu

ABSTRACT

In this paper, we propose a peer-to-peer streaming solution to address the on-demand media distribution problem. We identify two issues, namely the asynchrony of user requests and heterogeneity of peer network bandwidth. Our key techniques to address these two issues are cache-and-relay and layer-encoded streaming. A unique challenge of layered peer-to-peer streaming is that the bandwidth and data availability (number of layers received) of each receiving peer are constrained and heterogeneous, which further limits the bandwidth and data availability of its downstream node when it acts as the supplying peer. This challenge distinguishes our work from existing studies on layered multicast. Our experiments show that our solution is efficient at utilizing bandwidth resource of supplying peers, scalable at saving server bandwidth consumption, and optimal at maximizing streaming qualities of all peers.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Applications; C.2.5 [**Local and Wide-Area Networks**]: Internet; D.4.4 [**Communications Management**]: Network Communication; D.4.8 [**Performance**]: Simulation

General Terms

Algorithms, Design, Performance

Keywords

Peer-to-Peer, Layered Streaming, Overlay

[*]This work was supported by the National Science Foundation under contract number 9870736, the Air Force Grant under contract number F30602-97-2-0121, National Science Foundation Career Grant under contract number NSF CCR 96-23867, NSF CISE Infrastructure grant under contract number NSF EIA 99-72884, and NASA grant under contract number NASA NAG 2-1250.

1. INTRODUCTION

Large-scale on-demand multimedia distribution has been shown as one of the killer applications in current and next-generation Internet. In existing solutions, multicast has been extensively employed since it can effectively deliver media data to multiple receivers, while minimizing server and network overhead. However, the nature of multicast is intrinsically in conflict with some important features of media distribution, namely the *asynchrony* of user requests and *heterogeneity* of client resource capabilities. For asynchrony, a user may request media data at any time, which is against the synchronous transmission manner of multicast. For heterogeneity, clients may request stream of different qualities due to their own resource constraints, especially network bandwidth. Therefore, no single multicast stream can meet everyone's requirement. These issues are illustrated in Fig. 1.

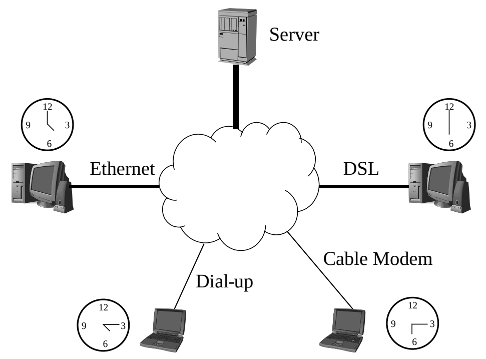


Figure 1: Asynchrony and Heterogeneity in On-Demand Media Distribution

Despite the above conflicts and numerous solutions to resolve them[2][9][11], the most serious problem faced by multicast today is the deficiency of its deployment in the wide-area network infrastructure. Moreover, this problem is likely to persist in the foreseeable future. As an alternative, application-layer multicast [14] is proposed. In this approach, end systems, instead of routers, are organized into an overlay to relay data to each other in a peer-to-peer fashion. Besides its initial success in addressing the absence of IP multicast, application-layer overlay is now believed to offer us more, because end system is flexible enough to provide more functionalities than simply forwarding packets, such as caching[16], data/service indirection[6], resilient routing[1], peer-to-peer streaming[5][4][13], etc. Bearing this in mind, we now revisit the aforementioned issues of asynchrony and heterogeneity in peer-to-peer streaming.

First, recent studies on cache-and-relay[10][16] show promising solutions to resolve the conflict between asynchronous requests and synchronous multicast transmission in peer-to-peer streaming. By delaying the received stream through caching, an end host could relay it to another peer, who requests the same data at a later time. Our previous study[16] further showed that this approach introduces less server/network overhead than IP-multicast-based solutions.

Second, the layer-encoded streaming approach[11][9][12] has the potential to address the issue of heterogeneity. For example, in layered multicast[11][12], a stream is encoded into multiple layers, then fed into different IP multicast sessions. A receiver with constrained bandwidth only needs to receive a subset of all layers to decode the stream with certain degraded quality.

However, existing layered streaming solutions cannot be directly applied to peer-to-peer system. The fundamental reason roots at the dual role of the end host as both supplying peer and receiving peer. First, as a receiving peer, an end host may receive only a subset of all layers, due to its limited inbound bandwidth or processing capabilities. In peer-to-peer system, this means that its data availability as a supplying peer is also constrained, which further limits the data availability of its downstream peers. Second, the outbound bandwidth of relaying nodes are limited and heterogeneous. This means that as a supplying peer, it has constrained bandwidth supply to the downstream nodes. These problems never arise in layered multicast, whereby the end host is always the sink of data path. To summarize, these unique challenges distinguish our study from previous ones in layered streaming.

The rest of this paper is organized as follows. In Sec. 2, we formulate the problem of layerd peer-to-peer streaming and show its NP-complete complexity. In Sec. 3, we present our solution. Sec. 4 evaluates its performance. Sec. 5 reviews the related work. Finally, Sec. 6 concludes.

2. PROBLEM FORMULATION

We consider the distribution of a layer-encoded stream across a set of peers with heterogeneous inbound and outbound bandwidths. As shown in Fig. 2, a peer can retrieve the stream at any time, requiring arbitrary quality, i.e., arbitrary number of layers. Each peer caches the received stream in a local circular buffer for a certain amount of time. In other words, a buffer window is kept along the playback of the stream, say 5 minutes. In this way, any later peer (within 5 minutes) requesting the same stream can receive the buffered data from the current host. Furthermore, it can retrieve different layers from multiple peers in parallel. For example, in Fig. 2, H_3 is 3 minutes later than H_1 and 2 minutes later than H_2. Thus, H_3 can stream from H_1 and H_2. On the other hand, H_4 can only stream from H_3 since it is already outside the buffer windows of H_1 and H_2. We consider the media server (H_0) as a special peer, which stays online permanently and has all layers of the stream.

A good peer-to-peer streaming solution should consider two factors. The first factor is the overall streaming quality of all peers, regarded as the *system benefit*. The second factor is the server bandwidth consumption, regarded as the *system cost*. Therefore, our goal is to maximize the *net benefit* (system benefit excluding system cost), under each

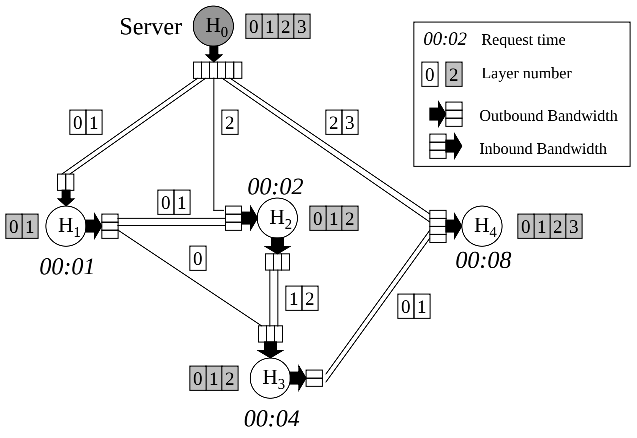


Figure 2: Layered Peer-to-peer Streaming

peer's bandwidth constraint. We introduce the following definitions.

- **Layer-encoded Stream** $\{l_0, l_1, l_2, \ldots, l_L\}$, with l_0 as the base layer and others as enhancement layers. These layers are accumulative, i.e., l_i can be only decoded if layers l_0 through l_{i-1} are available. For now, we assume that all layers have identical streaming rate. In Sec. 3.4, we will relax this assumption.

- **Peers** $\{H_0, H_1, H_2, \ldots, H_N\}$, with H_0 as the server. H_1 through H_N are sorted based on their requesting times. Furthermore, we use I_k and O_k to denote the inbound and outbound bandwidths of H_k. For the purpose of simplicity, I_k and O_k are measured as the number of layers H_k can receive and send.

- **Streaming Quality** Q_k, the streaming quality received at H_k, measured as number of received layers. In particular, Q_k^m is used to denote the number of layers received from the supplying peer H_m. Consequently, Q_k^0 is the number of layers received from server H_0.

- **Layer Availability** A_k, number of layers available at the cache of H_k. In order to save cache space, H_k is allowed to discard some received layers (always the highest ones, i.e., the most enhanced ones) after playback. Therefore, $A_k \leq Q_k$.

- **Buffer Length** B_k, measured as the time length of H_k's buffer. Note that with the same cache space, B_k may vary depending on how many layers H_k caches.

- **Request Time** t_k, the time at which H_k starts to playback the stream. For two peers H_k and H_m, if $t_k < t_m$ and $t_m - t_k \leq B_k$, then H_k is qualified as H_m's supplying peer. In other words, H_m falls within the buffer window of H_k, denoted as $H_k \rightarrow H_m$. Given that the server H_0 can stream to all peers at any given time, we have $H_0 \rightarrow H_k$ $(k = 1, \ldots, N)$.

- **Supplying Peer Constraint** C_k, which sets the maximum number of supplying peers H_k can stream from in parallel. We do so to lower the synchronization complexity of parallel streaming. If the streaming quality

Q_k still cannot be met by all the supplying peers due to their limited outbound bandwidth or layer availability, we allow the server H_0 to send the missing layers to H_k.

Given above definitions, we can now formalize our goal as

$$\textbf{maximize} \quad \sum_{k=1}^{N} (Q_k - Q_k^0)$$

$$\textbf{subject to} \quad \textbf{(1)}\ Q_k \le I_k (1 \le k \le N)$$
$$\textbf{(2)}\ \sum_{H_k \to H_m} Q_m^k \le O_k (1 \le k \le N)$$

The first constraint states that the streaming quality (number of layers received) of each peer H_k should not exceed its inbound bandwidth. The second constraint is that as a supplying peer, H_k cannot output more layers than its outbound bandwidth allows to send.

Theorem 1: Layered Peer-to-peer Streaming problem is NP-complete.

Proof is provided in Appendix A.

3. LAYERED PEER-TO-PEER STREAMING SOLUTION

We now present our solution. We have following design objectives.

First, each peer H_k should configure its own streaming session (what layers to stream from which peer) *locally*. As Theorem 1 showed that having global view and control does not help to reduce the problem complexity, this approach is more practical and cost-effective[1].

Second, as the client requests come at different times, our solution should be *incremental*, i.e., the streaming quality of existing peers must be preserved when admitting new peers. As such, a new client H_k can only utilize the residual outbound bandwidth of its supplying peers.

3.1 Basic Algorithm

We first present the basic algorithm, which assumes that H_k is allowed to receive data from unlimited number of senders. The algorithm is a greedy approach, in that it always maximally utilizes the outbound bandwidth of the peer with the smallest number of layers.

Executed by H_k, the algorithm takes the following inputs: (1) I_k, the inbound bandwidth of H_k; (2) $S = \{H_1, \ldots, H_M\}$, a set of hosts qualified as the supplying peers of H_k. This means that H_k falls within the buffer windows of H_1 through H_M. These peers are sorted such that $A_1 \le A_2 \le \ldots \le A_M$. The algorithm returns the streaming quality Q_k, and collects the selected supplying peers into set P.

Fig. 3 illustrates our algorithm. H_k's inbound bandwidth allows to receive 11 layers, as represented by the white bar. It requests individual layers from supplying peers H_1 through H_4. H_1 has 3 layers in its cache, as represented by its bar. Likewise, H_2 has 4 layers and so on. The shadowed squares of H_1 through H_4 represent the number of layers their outbound bandwidths allow to send. Note that these shadowed squares only indicates how many layers one can send. For example, H_3 has six layers available in its cache, and is able to output any five of them. Black squares represent layers which are already allocated.

```
UnconstrainedSenders($I_k, H_1, \ldots, H_M$)
    /* Calculation of $Q_k$ */
1       $Q_k \leftarrow 0$
2       $P \leftarrow \phi$
3       $m = 1$
4       repeat
5           $Q_k^m \leftarrow min(O_m, A_m - Q_k, I_k - Q_k)$
6           $enque(P, H_m)$
7           $Q_k \leftarrow Q_k + Q_k^m$
8           $m \leftarrow m + 1$
9       until $Q_k = I_k$ or $m > M$

    /* Layer Allocation */
10      $index \leftarrow 0$
11      repeat
12          $H_m \leftarrow deque(P)$
13          allocate layers $\sum_{i=index}^{index+Q_k^m-1} l_i$ to $H_m$
14          $O_m \leftarrow O_m - Q_k^m$
15          $index \leftarrow index + Q_k^m$
16      until $P = \phi$
```

Table 1: Basic Algorithm

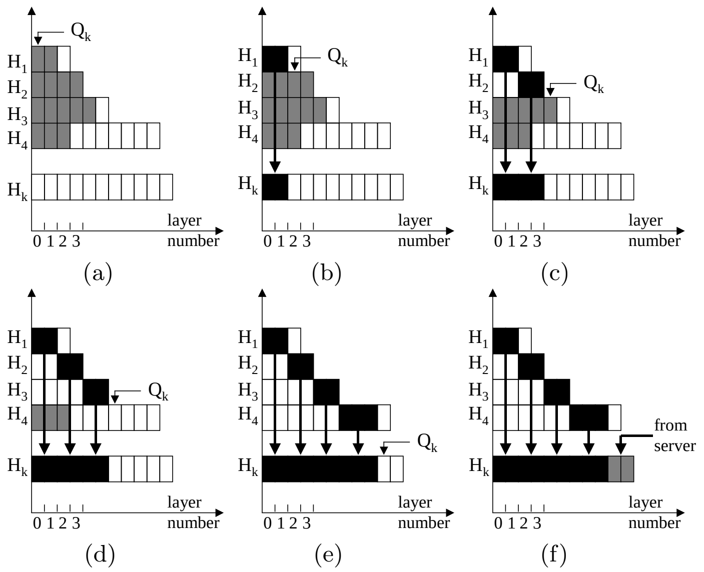


Figure 3: Sample Illustrating the Basic Algorithm

Initially, Q_k at H_k is 0 (Fig. 3 (a)). H_k first uses up the outbound bandwidth of H_1 and increases Q_k to 2 (Fig. 3 (b)). Then, although H_2 is able to output four layers, H_k only needs two layers from it, since the lower two are already allocated to H_1 (Fig. 3 (c)). For the same reason, H_3 only needs to output two layers. Note that its allocated layers shift up, since H_3 is free to output any layers available in its cache (Fig. 3 (d)). After H_4 is done, S becomes empty and Q_k is still two layers lower than the expected quality of H_k

[1]In order to do so, H_k needs to know who are qualified as its supplying peers and their layer availability and current outbound bandwidth. The dissemination and acquisition of such information can be done through any publish/subscribe service, which is considered orthogonal and not covered in this paper. Interested readers are also refereed to our previous study[15], which proposed a time-based publish/subscribe solution to address this problem.

(Fig. 3 (e)). Finally, we ask the server to send the missing layers to H_k (Fig. 3 (f)).

Theorem 2: The basic algorithm can allocate maximum number of layers for H_k.

We leave the proof in Appendix B.

3.2 Enhanced Algorithm under Supplying Peer Constraint

We now turn to a more constrained scenario, where H_k can only stream from finite number of supplying peers. Assuming the existence of $S = \{H_1, \ldots, H_M\}$ defined in Sec. 3.1, let us denote $Q_k^*(M, 1)$ as the optimal solution if H_k can only choose one supplying peer from H_1 through H_M. This solution is straightforward: we only need to find the peer which can send the largest number of layers. Then for $Q_k^*(M, 2)$, assuming $Q_k^*(m, 1)$ is already known, we can find the one from H_{m+1} to H_M that is able to further contribute the most layers, denoted as $Q_{max}(H_{m+1}, \ldots, H_M)$. Then $Q_k^*(m, 1) + Q_{max}(H_{m+1}, \ldots, H_M)$ is a candidate answer to $Q_k^*(M, 2)$. Repeating this procedure from $m = 1$ to $M - 1$, the best solution can be found as

$$Q_k^*(M, 2) = \max_{1 \le m < M}[Q_k^*(m, 1) + Q_{max}(H_{m+1} \ldots H_M)]$$

In general, we have

$$Q_k^*(M, C_k) = \max_{C_k - 1 \le m < M}[Q_k^*(m, C_k - 1) + Q_{max}(H_{m+1} \ldots H_M)]$$

This is a typical dynamic programming problem. We show our algorithm in Tab. 2.

Theorem 3: Under constraint C_k, the enhanced algorithm can allocate maximum number of layers for H_k.

We leave the proof in Appendix C. The algorithm complexity is $O(C_k M^2)$.

3.3 Node Departure

Node departure can happen frequently, due to user logout, machine crash or network failure. Upon losing a supplying peer H_m, the receiving peer H_k should reconfigure its session by rerunning the layer allocation algorithm. However, during this *transient period*, its streaming quality may degrade. Here we discuss how our solution can adapt to this situation.

First, if H_m departs normally, it will notify H_k to reconfigure its session. Meanwhile, H_m continues to stream data remained in its buffer to H_k as normal. Therefore, as long as the reconfiguration of H_k's session finishes before H_m's buffer is drained, H_k will stay unaffected. Otherwise, H_m can be regarded as failed, which will be addressed below.

If H_m fails, upon detecting it, H_k has two options. First, It can temporally request from the server the layers which were allocated to H_m, until its session reconfiguration is finished. In other words, during this time, the server acts as H_m. Second, if the server bandwidth is already fully occupied, then the streaming quality of H_k has to be degraded gracefully. As an example, in Fig. 4, H_k initially received data from supplying peers H_1 through H_4. When H_2 fails, H_k asks other peers to stream as usual. However, H_3's layers are shifted down to meet the gap left by H_2. H_4's layers

```
ConstrainedSenders($I_k, C_k, H_1, \ldots, H_M$)
    /* Initialization */
1     for $c \leftarrow 0$ to $C_k$
2         for $m \leftarrow 0$ to $M$
3             $Q_k^*(m, c) \leftarrow 0$
4             $P^*(m, c) \leftarrow \phi$

    /* Calculation of $Q^*(M, C_k)$ */
6     for $c \leftarrow 1$ to $C_k$
7       for $m \leftarrow c$ to $M - C_k + c$
8         for $t \leftarrow m$ to $M - C_k + c$
9           $Q' \leftarrow min(O_t, A_t - Q_k^*(m-1, c-1),$
                    $I_k - Q_k^*(m-1, c-1))$
10          if $Q_k^*(t-1, c) > Q_k^*(t, c)$ or
11              $Q_k^*(m-1, c-1) + Q' > Q_k^*(t, c)$
12          begin
13            if $Q_k^*(t-1, c) < Q_k^*(m-1, c-1) + Q'$
14            then
15              for $i \leftarrow 1$ to $m-1$
16                  $Q_k^{i*}(t, c) \leftarrow Q_k^{i*}(m-1, c-1)$
17              $Q_k^{t*}(t, c) \leftarrow Q'$
18              $P^*(t, c) \leftarrow P^*(m-1, c-1) \cup \{H_t\}$
19              $Q_k^*(t, c) \leftarrow Q_k^*(m-1, c-1) + Q_k^{t*}(t, c)$
20            else
21              for $i \leftarrow 1$ to $t-1$
22                  $Q_k^{i*}(t, c) \leftarrow Q_k^{i*}(t-1, c)$
23              $P^*(t, c) \leftarrow P^*(t-1, c)$
24              $Q_k^*(t, c) \leftarrow Q_k^*(t-1, c)$
25          end

    /* Layer Allocation */
26    $index \leftarrow 0$
27    repeat
28        $H_m \leftarrow deque(P^*(M, C_k))$
29        $Q_k^m \leftarrow Q_k^{m*}(M, C_k)$
30        allocate layers $\sum_{i=index}^{index+Q_k^m-1} l_i$ to $H_m$
31        $O_m \leftarrow O_m - Q_k^m$
32        $index \leftarrow index + Q_k^{m*}(M, C_k)$
33    until $P^*(M, C_k) = \phi$
```

Table 2: Enhanced Algorithm

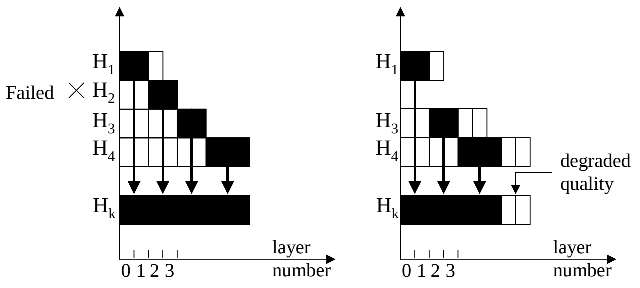


Figure 4: Graceful Degradation of Streaming Quality (when server bandwidth is fully occupied)

are shifted down likewise. Thus, H_k's quality only drops by H_2's share.

Another concern is that the quality degradation of H_k could further cause the quality degradation of its children. In this case, H_k can be regarded as normal departure. As explained earlier in this subsection, the streaming quality of H_k's children will not be affected until H_k's buffer is drained. Therefore, buffering can effectively absorb the propagation of quality degradation.

3.4 Layer Rate Heterogeneity

So far in this paper, we have assumed that all layers have identical streaming rate. In practice, however, this is often not the case[12]. To show the complexity of this problem, we first go through the following (re)definitions.

- r_i, the streaming rate of a layer l_i (Kbps).
- I_k and O_k, the inbound and outbound bandwidth of H_k, measured as raw data rate (Kbps).

We define the *Heterogeneous-Rate Layer Allocation Problem* as follows. Given a set of layers $\{l_0, \ldots, l_L\}$ with different streaming rates $\{r_0, \ldots, r_L\}$, the receiving peer H_k, and a set of supplying peers $S = \{H_1, \ldots, H_M\}$, find an optimal solution, which allocates maximum number of layers for H_k.

Theorem 4: Heterogeneous-Rate Layer Allocation Problem is NP-complete.

We leave the proof in Appendix D. We modify existing algorithms (Tab. 1 and Tab. 2) to accommodate the layer rate heterogeneity. They are shown in Tab. 3 and Tab. 4, respectively. To save space, we only show the modified part of each algorithm.

```
UnconstrainedSenders($I_k, H_1, \ldots, H_M$)
        ...
4       repeat
5           $Q_k^m \leftarrow max(n \mid \sum_{i=Q_k}^{Q_k+n} r_i \le O_m, \sum_{i=0}^{Q_k+n} r_i \le I_k$
                             $Q_k + n \le A_m)$
6           $enque(P, H_m)$
7           $Q_k \leftarrow Q_k + Q_k^m$
8           $m \leftarrow m + 1$
9       until $\sum_{i=0}^{Q_k} r_i > I_k$ or $m > M$
        ...
```

Table 3: Modified Basic Algorithm for Layer Rate Heterogeneity

```
ConstrainedSenders($I_k, C_k, H_1, \ldots, H_M$)
        ...
9           $Q' \leftarrow max(n \mid \sum_{i=Q_k^*(m-1,c-1)}^{Q_k^*(m-1,c-1)+n} r_i \le O_t,$
                        $\sum_{i=0}^{Q_k^*(m-1,c-1)+n} r_i \le I_k,$
                        $O_k^*(m-1, c-1) + n \le A_t)$
        ...
```

Table 4: Modified Enhanced Algorithm for Layer Rate Heterogeneity

4. PERFORMANCE EVALUATION

We simulate a peer-to-peer streaming system of total 40000 peers. We categorize peers into three classes: (1) Modem/ISDN peers, which take 50% of the population with maximum total bandwidth of $112Kbps$; (2) Cable Modem/DSL peers, which take 35% of the population with maximum total bandwidth of $1Mbps$; and (3) Ethernet peers, which take rest of the population with maximum total bandwidth of $10Mbps$. Each peer requests the 60-minute video at different times during the 24-hour run. The layer rate of the video is $20Kbps$. Its full-quality streaming rate is $1Mbps$, which consists of 50 layers.

4.1 Overall Streaming Quality and Scalability

We compare our solution with *versioned streaming*, another commonly used solution to address the end host heterogeneity. In our experiment, the video is encoded into 50 versions with different streaming rates. Each version is distributed using an independent application-layer multicast tree.

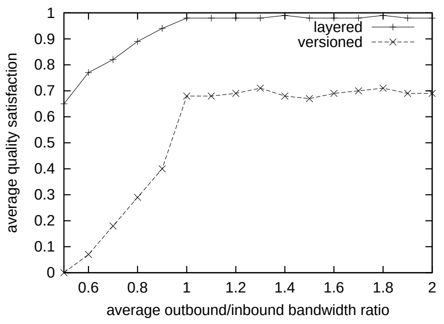


Figure 5: Overall Streaming Quality (Request Rate=$120req/hr$, Buffer Length=$5min$)

We first test these two solutions at utilizing the outbound bandwidth of supplying peers. Since each peer may expect different streaming qualities, we propose a new metric *Streaming Quality Satisfaction*, which is defined as the ratio of received quality and expected quality of a peer H_k, namely Q_k/I_k. The maximum value is 1. As shown in Fig. 5, when the average ratio of each peer's outbound/inbound bandwidth is greater or equal to 1, the average quality satisfaction of layered approach is almost 1, which means that the peers can virtually self-support. On the other hand, the curve of versioned approach never goes over 0.7. Moreover, when the outbound/inbound ratio is below 1, the performance of layered approach degrades linearly, which indicates that it is always able to fully utilize the marginal outbound bandwidth of supplying peers. In comparison, the curve of versioned approach drops suddenly since most supplying peers' outbound bandwidth cannot send out the entire video. This reveals that the layered approach is more adapted to the bandwidth asymmetricity (outbound bandwidth less than inbound bandwidth), which is often the case for Cable Modem and ADSL users.

We then test the scalability of these two solutions at saving server bandwidth. Fig. 6 shows that, when client request rate grows, the server bandwidth consumption of layered approach actually drops. The main reason is that, when a requesting peer joins, it also acts as a supplying peer to the

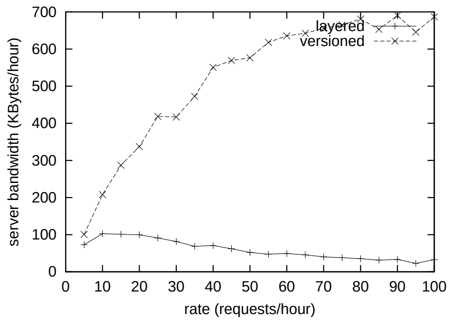


Figure 6: Server Bandwidth Consumption (Average Outbound/Inbound Bandwidth Ratio=1, Buffer Length=$5min$)

following requesting peer, which in turn forms a chain. This chain gets longer when the average interarrival time of different peers shortens. Such a chain effect also happens in the case of versioned streaming. However, since this approach always requires enough outbound bandwidth to output the entire video, only few supplying peers qualify, which causes the chain to be easily broken.

4.2 Impact of Design Parameters

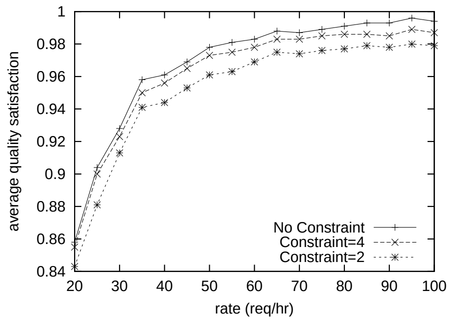


Figure 7: Impact of Supplying Peer Constraint (Average Outbound/Inbound Bandwidth Ratio=1, Buffer Length=$5min$)

For a receiving peer, limiting the number of supplying peers can help lowering the operation and synchronization complexity. On the other hand, it does not guarantee to maximally utilize the outbound bandwidth of all supplying peers, compared to the unconstrained case. Fig. 7 shows that constraining the number of senders to 4 already acquires nearly identical performance to the unconstrained case, in terms of average quality satisfaction.

Another important design parameter is the buffer length of each supplying peer. Apparently, longer buffer enables a supplying peer to help more later-coming peers, thus improving the overall streaming quality. As revealed in Fig. 8, this is true when request rate is low. This can help keep the peers chain (Sec. 4.1) from broken. Further increasing buffer size has very little improvement space, since it can help little at prolonging the chain. This finding suggests that with small-to-medium sized cache space (3 or 5 minutes out of an 1-hour video), the system can acquire great performance gain.

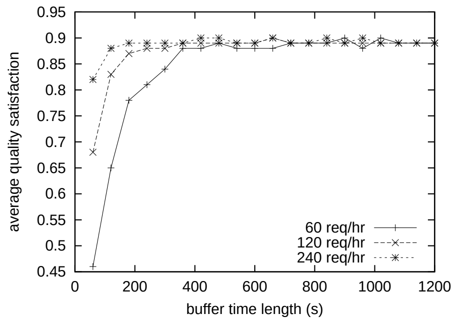
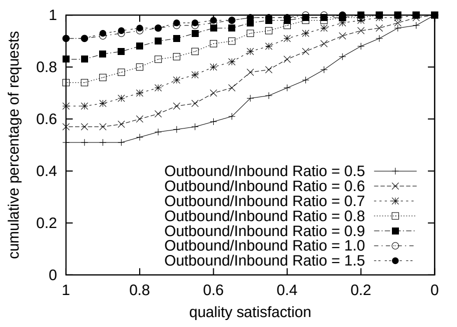


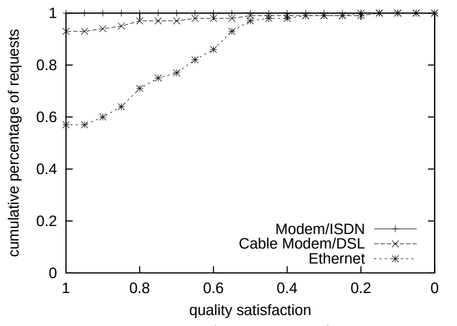
Figure 8: Impact of Buffer Length (Average Outbound/Inbound Bandwidth Ratio=0.8)

4.3 Fairness

As revealed in Fig. 5, when the average ratio of each peer's outbound/inbound bandwidth is below 1, the average quality satisfaction drops correspondingly, i.e., not every peer's streaming quality can be as good as expected. As such, a fair solution should ensure that such deviation does not greatly vary from one peer to another.



(a) Different Outbound/Inbound Bandwidth Ratios

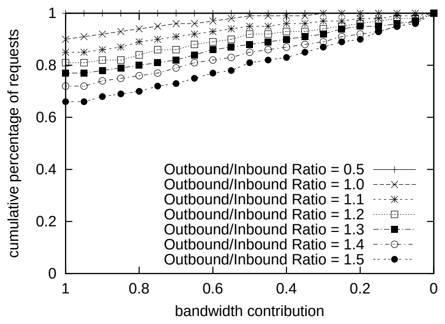


(b) Different Peer Classes (Outbound/Inbound Ratio=1)

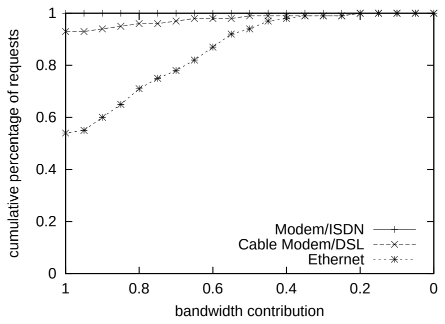
Figure 9: Cumulative Distribution of Quality Satisfaction (Request Rate=$120req/hr$, Buffer Length=$5min$)

We plot the cumulative distribution of peers with different quality satisfaction in Fig. 9 (a). When the average outbound/inbound ratio is 0.5, about 50% of the peers acquire the expected streaming quality. Then the quality satisfaction decreases almost linearly among the rest of the peers. We observe the similar trend when increasing the average outbound/inbound ratio. However, when the ratio becomes greater or equal than 1, only 90% of the population receive the full quality satisfaction. Furthermore, this percentage stays unimproved when we further enlarge the outbound bandwidth of supplying peers. We find the answer in Fig. 9 (b).

In Fig. 9 (b), we study the distribution of quality satisfaction over different peer classes when the average outbound/inbound bandwidth ratio is 1. Although all Modem/ISDN peers receive the full quality satisfaction, this is not the case for 5% of Cable Modem/DSL peers. For Ethernet peers, over 40% of them do not receive the stream quality as expected. The main reason is that when a peer of higher class (e.g., Ethernet) joins, it can happen that all its supplying peers belong to the lower classes (e.g., Cable Modem or ISDN). Therefore, even when these peers have available outbound bandwidth, they still do not have higher stream layers, which are requested by the peer of higher class.



(a) Different Outbound/Inbound Bandwidth Ratios



(b) Different Peer Classes (Outbound/Inbound Ratio=1)

Figure 10: Cumulative Distribution of Outbound Bandwidth Contribution(Request Rate=$120req/hr$, Buffer Length=$5min$)

We then evaluate whether our solution enables each peer to fairly contribute its outbound bandwidth. As shown in Fig. 10 (a), when the outbound/inbound ratio is 0.5, each peer contributes all of its bandwidth. When the ratio is 1, only 90% of all peers contribute all of its bandwidth. The contribution decreases linearly among the rest of the peers. Again, this can be explained when we plot the distribution of bandwidth contribution over different peer classes in Fig. 10 (b).

Fig. 10 (b) exhibits the similar pattern with Fig. 9 (b). All Modem/ISDN peers contribute all of their bandwidths. This is mainly due to the greedy nature of our layer allocation algorithm, which always first exploit the peers with smallest number of layers. Almost 40% of the Ethernet peers only partially contribute their bandwidth. This is mainly because that they mostly stream to the lower-class peers, who always first request layers from supplying peers of the same class, if any. To this end, we conclude that both *data availability constraint* and *bandwidth availability constraint* of supplying peers have impact on the issue of fairness.

4.4 Robustness

To test the robustness of our solution, we inject random node departures/failures into our simulation. We are mainly interested with the ability of our solution at absorbing the transient failure during stream session reconfiguration via buffering (Recall Sec. 3.3). In our experiment, 50% of the supplying peers depart early before the playback is finished. These peers are further categorized into *normal departure peers* and *failed peers*. A normal departure peer notifies its children upon leaving, but continues to stream until its buffer is drained. The children will stay unaffected if they can finish reconfiguring their sessions before the buffer is drained. Otherwise, they have to experience temporal quality degradation, as depicted in Fig. 4. On the other hand, if a peer fails, its children will be definitely affected. We use *Failure Ratio* to denote the percentage of failed ones among all departure peers.

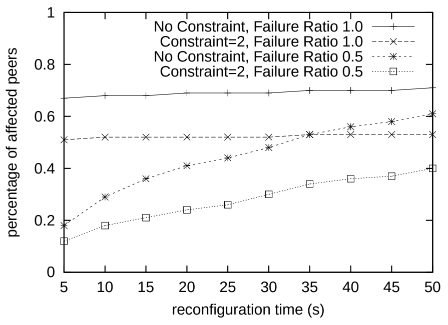


Figure 11: Percentage of Affected Peers (Request Rate=$120req/hr$, Buffer Length=$5min$, Average Outbound/Inbound Bandwidth Ratio=1)

As shown in Fig. 11, buffering can effectively "mask" more than half of the peer departures, when the average session reconfiguration time is small (5 seconds). The effect of buffering diminishes as the failure ratio grows. Eventually, it is rendered useless when all departure peers are failed ones. Also, one can impose supplying peer constraint to effectively lower the percentage of affected peers. However, as a side effect, the average quality degradation is higher than the unconstrained case (Fig. 12). The reason is that when the number of supplying peers is constrained, in order to maximize the streaming quality, the enhanced layer allocation algorithm (Sec. 3.2) always chooses supplying peers that can contribute most number of layers. Therefore, when one of them departs or fails, it is likely to incur more quality degradation.

4.5 Layer Rate Heterogeneity

Encoding a stream into more layers can help us better utilize the marginal inbound/outbound bandwidth of peers, therefore increases the average streaming quality and helps save server cost. However, the price is that we have to put redundant information into each layer. Such inefficiency adds up as we increase the number of layers.

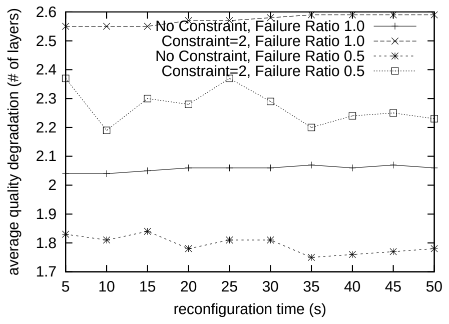


Figure 12: Average Quality Degradation (Request Rate=$120 req/hr$, Buffer Length=$5 min$, Average Outbound/Inbound Bandwidth Ratio=1)

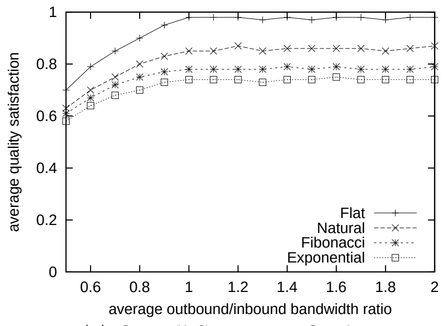
So far in this section, we have only adopted the flat rate scheme, i.e., the rate of all layers are identical. We now explore several other layer rate allocation schemes as follows.

- **Natural Number Scheme** Assuming that the rate of base layer l_0 is r_0, the rate of layer $l_k (k > 0)$ is $r_k = k \cdot r_0$. In this way, the original 50 layers in the flat rate scheme are collapsed into 10 layers.
- **Exponential Scheme** In this scheme, it follows that $r_k = r_0 \cdot 2^k$. The original 50 layers are collapsed into 6 layers.
- **Fibonacci Scheme** In this scheme, $r_1 = 2r_0$. The rates of other layers satisfy that $r_k = r_{k-1} + r_{k-2}$. The original 50 layers are collapsed into 7 layers.

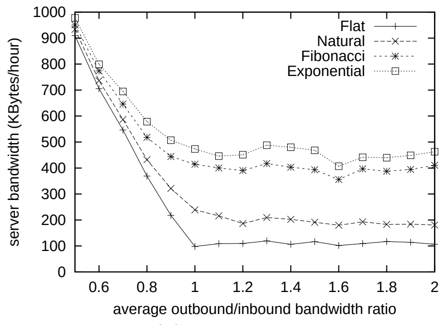
To save space, we only report the performance comparison results of these schemes in the case of unconstrained supplying peers. In terms of average quality satisfaction (Fig. 13 (a)), all three of them show the similar growing trend as the flat rate scheme, when increasing the outbound/inbound ratio. Among them, the natural number scheme performs the best, as it exhibits finer layer rate granularity than the other two. For the same reason, the exponential scheme ends up with the worst performance. Reflected in Fig. 13 (b), the server bandwidth consumption of natural number scheme remains the closest to the flat rate scheme. On the other hand, the server cost of exponential scheme is the highest.

Despite their performance losses, we show that the new schemes can effectively save the operation and synchronization complexity of parallel downloading. In Fig. 13 (c), all of them constantly limit the average number of supplying peers within 1.8. In contrast, the average number of supplying peers in the flat rate scheme continuously increases. Plus, considering the fact that having fewer number of layers can reduce the aforementioned data inefficiency, the actual performance losses in Fig. 13 (a) and (b) can be even less.

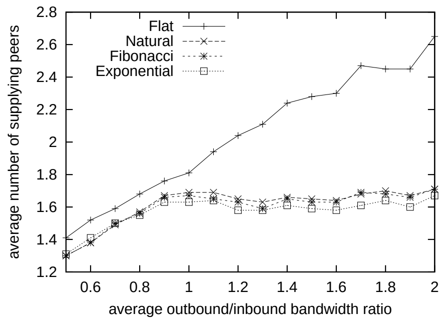
From Fig. 13, we may draw the general conclusion that finer layer rate granularity introduces more performance gain, as well as higher operation complexity. In practice, however, exploring the "sweetspot" of such tradeoff can be hard. Unlike livecast, where the source could modify layer rate on the fly according to network dynamics[12], the playback streaming requires the layer rate to be determined offline, or adjusted with limited flexibility. Therefore, an optimal layer rate allocation scheme is impossible without extensive measurement study on crucial factors, including the population of different peer classes, peers' joining sequence and their streaming access patterns, etc.



(a) Overall Streaming Quality



(b) Server Cost



(c) Average Number of Supplying Peers

Figure 13: Performance Comparison of Different Layer Rate Allocation Schemes (Average Request Rate=$120 req/hr$, Buffer Length=$5 min$)

5. RELATED WORK

The issues of asynchrony and heterogeneity in multimedia distribution have been extensively studied in the context of IP multicast. For asynchrony, various solutions (batching, patching, merging, periodic broadcasting, etc.) have been proposed to address the conflict between asynchronous user requests and synchronous nature of multicast communication. [2] provides a wrap-up review of these solutions and includes them into a unified analytical model. The basic idea of these techniques is to let a client joins single or multiple on-going multicast channels, which were assigned to previous clients. Although the missing initial part has to be unicast or multicast from the server, the cost is greatly reduced compared to the complete retransmission. This requires the client to have sufficient caching space, as well as enough bandwidth to receive multiple streams simultaneously.

For heterogeneity, layered multicast[11] was proposed, where a stream is separated into multiple layers, then transmitted through different multicast channels. A client only needs to subscribe a subset of all layers, based on its bandwidth and processing capabilities. Many follow-up studies work on layer rate allocation mechanisms to maximize the overall streaming quality[12], ensure max-min fairness[3], enforce congestion control[7], or any combination of the above goals. The layered streaming approach has also been used in the context of unicast congestion control[9].

Before our work, several studies have explored the peer-to-peer streaming from different aspects. [5] by Xu et al. is one of the pioneering works that proposed the concept of peer-to-peer streaming. It mainly focuses on the analysis of the capacity of a peer-to-peer streaming system. ZIGZAG[4] by Tran et al. tries to construct an application-layer multicast tree, such that it minimizes the end-to-end delay (bounded tree height) and maximizes the utilization of peers' bandwidth (bounded node degree). The CoopNet[13] proposes a hybrid architecture, which integrates the client-server and peer-to-peer models. This architecture is proved scalable and robust against the "flash crowd", upon which happens the peers will relay data to each other to protect the server from being overwhelmed.

6. CONCLUSION

We presented a layered peer-to-peer streaming solution to address the asynchrony and heterogeneity issues in on-demand media distribution. Given the experimental results, we are confident to claim that our solution is optimal at maximizing the streaming quality of heterogeneous peers, scalable at saving server bandwidth. and efficient at utilizing bandwidth resource of supplying peers. We also evaluated our solution to see: (1) whether it establishes fairness among peers, in terms of streaming quality satisfaction and bandwidth contribution, and (2) whether it is robust against unexpected peer departures/failures.

Our initial conclusions heavily depend on our experimental assumptions on peer class population, their network bandwidth characteristics, their join/access patterns, etc. These factors deserve extensive measurement and statistical studies, as well as the design of new evaluation methodologies, all of which constitute the possible directions of future work.

7. REFERENCES

[1] D. Andersen, H. Balakrishnan, M. Kaashoek and R. Morris. Resilient overlay networks. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2001.

[2] D. Eager, M. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *IEEE Transaction on Knowledge and Data Engineering*, 13(5), 2001.

[3] D. Rubenstein, J. Kurose, and D. Towsley. The impact of multicast layering on network fairness. *IEEE/ACM Transactions on Networking*, 10(2), 2002.

[4] D. Tran, K. Hua and S. Sheu. Zigzag: An efficient peer-to-peer scheme for media streaming. In *IEEE INFOCOM*, 2003.

[5] D. Xu, M. Hefeeda, S. Hambrusch and B. Bhargava. On peer-to-peer media streaming. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2001.

[6] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *ACM SIGCOMM*, 2002.

[7] J. Byers, M. Luby and M. Mitzenmacher. Fine-grained layered multicast. In *IEEE INFOCOM*, 2003.

[8] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[9] R. Rejaie, M. Handley, and D. Estrin. Layered quality adaptation for internet video streaming. *IEEE Journal on Selected Areas of Communications, Special issue on Internet QoS*, 2000.

[10] S. Jin and A. Bestavros. Cache-and-relay streaming media delivery for asynchronous clients. In *International Workshop on Networked Group Communication (NGC)*, 2002.

[11] S. McCanne, V. Jacobson and M. Vetterli. Receiver-driven layered multicast. In *ACM SIGCOMM*, 1996.

[12] T. Kim and M. Ammar. A comparison of layering and stream replication video multicast scheme. In *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2001.

[13] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2002.

[14] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2000.

[15] Y. Cui and K. Nahrstedt. Proxy-based asynchronous multicast for efficient on-demand media distribution. In *SPIE Conference on Multimedia Computing and Networking Multimedia (MMCN)*, 2003.

[16] Y. Cui, B. Li and K. Nahrstedt. oStream: Asynchronous streaming multicast in application-layer overlay networks. *to appear in IEEE Journal on Selected Areas of Communications, Special Issue on Recent Advances in Service Overlay Networks*, 2003.

APPENDIX

A. PROOF OF THEOREM 1

We only need to prove that a special case of *Layered Peer-to-peer Streaming* is NP-complete. Let $Q_k = A_k = I_k = L + 1$ and $C_k = 1$ $(k = 1, \ldots, N)$. This means that every peer H_k is able, and willing to receive full-quality stream. Furthermore, H_k can only stream from one supplying peer. If not all layers can be streamed due to sender's outbound bandwidth constraint, the server H_0 will stream the missing layers to H_k. To ensure that Q_k is met for every client, we further assume that H_0 has enough outbound bandwidth to help each of them, i.e., O_0 is ∞. We refer to this problem as *Single-Sender Full-Quality Streaming.*

We construct the following graph as shown in Fig. 14.

The graph consists of three types of nodes. *Receiving Nodes* $\{R_2, \ldots, R_N\}$ represent receiving peers. *Supplying Nodes* $\{S_1, \ldots, S_{N-1}\}$ represent supplying peers[2]. An edge is directed from a sending node S_k to a receiving node R_m, if H_k could stream to H_m from its own cache ($H_k \rightarrow H_m$). The edge capacity is I_m, the inbound bandwidth of H_m. Finally, a *Virtual Node* V directs an edge to each sending node S_k. The edge capacity is O_k, the outbound bandwidth of S_k.

Now we can restate the *Single-Sender Full-Quality Streaming* problem as allocating a flow f_k from V to each receiving node R_k, such that the flow sum $\sum_{k=2}^{N} f_k$ is maximized. This is known as the *Single-Source Unsplittable Flow* problem, which is NP-complete[8]. It is also obvious that any particular flow allocation solution can be verified in linear time. Therefore, the *Single-Sender Full-Quality Streaming* problem is NP-complete and so is *Layered Peer-to-peer Streaming.*■

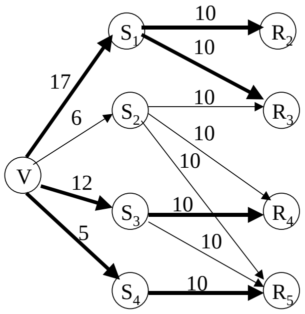


Figure 14: Single-Sender Full-Quality Streaming

B. PROOF OF THEOREM 2

We use induction to prove that the basic algorithm always allocates maximum number of layers from the peers dequeued from S so far. We use Q_k^m to denote the number of layers allocated, if m peers has been dequeued from S.

Basis. The first peer dequeued from S is H_1. Q_k^1 is the maximum of H_1's outbound bandwidth (O_1) or layer availability (A_1). Obviously, this is the maximum number of layers H_1 can contribute.

Induction Steps. Assume H_m is dequeued, and Q_k is the maximum number of layers that the previous peers H_1 through H_{m-1} can contribute. We show that we cannot increase the number of layers from H_1 through H_{m-1} to be more than Q_k^{m-1} by rearranging their existing layer allocation.

There are two cases as shown in Fig. 15. In the first case (Fig. 15 (a)), $Q_k^{m-1} = A_{m-1}$. This means that the peers H_1 through H_{m-1} already contribute the maximum number of layers allowed by their layer availability.

In the second case (Fig. 15 (b)), $Q_k^{m-1} < A_{m-1}$. This means that the outbound bandwidth of H_{m-1} has already been used up. We further go back to Q_k^{m-2}. If $Q_k^{m-2} = A_{m-2}$, then as we have shown in the first case, peers H_1 through H_{m-2} cannot further contribute any layers. Therefore, Q_k^{m-1} is the maximum number of layers H_1 through H_{m-1} can contribute. On the other hand, if $Q_k^{m-2} < A_{m-2}$, it means that the outbound bandwidth of H_{m-2} has also been used up. As such, we can go back continuously for at most m steps, until at some point n, where $Q_k^{m-n} = A_{m-n}$. In this case, peers H_1 through H_{m-n} cannot further contribute any layers; H_{m-n+1} through H_{m-1} have already used up their own outbound bandwidths. Therefore, Q_k^{m-1} is still the maximum number of layers H_1 through H_{m-1} can contribute.

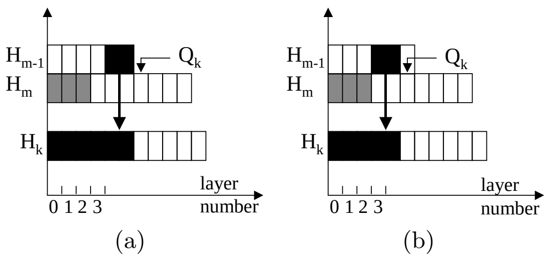


Figure 15: Proof of Theorem 2

Now we can conclude that Q_k^{m-1} cannot be increased by rearranging the existing layer allocation on H_1 through H_{m-1}. Therefore, H_m only needs to calculate Q_k^m based on the existing layer allocation of H_1 through H_{m-1}.■

C. PROOF OF THEOREM 3

The enhanced algorithm still employs the greedy approach of the basic algorithm. As shown in Theorem 2, each step of the algorithm can produce optimal result out of the supplying peers processed so far. Therefore, Q_k^m (defined in Appendix B) is qualified as the value function in dynamic programming, which traverses all combinations of C_k out of M peers. Therefore, the optimal solution can be acquired.■

D. PROOF OF THEOREM 4

We only need to prove that a special case of this problem is NP-complete. For $S = \{H_1, \ldots, H_M\}$, let $A_1 = \ldots = A_M = L + 1$ and $O_1 = \ldots = O_M = O$. This means that all supplying peers have $L + 1$ layers available in their cache. They also have the same outbound bandwidth O. We restate the problem as follows.

We regard $\{l_0, \ldots, l_n\} (n \leq L)$ as a finite set with $n + 1$ items. Each item l_i is associated with a size r_i. Our target is to find a partition, which separates this set into disjoint sets U_1 through U_M, such that the sum of the items in each set $U_i (1 \leq i \leq M)$ is less than O. We repeat the above procedure while increasing n, until $n = L$ or a valid solution cannot be found. This is known as the *Minimum Bin Packing* problem, which is NP-complete[8]. It is also obvious that any particular set partition solution can be verified in linear time. Therefore, the *Heterogeneous-Rate Layer Allocation* problem is NP-complete.■

[2] R_1 is missing because as the first receiver, H_1 has to receive the stream from the server. Similarly, S_N is missing since no other peers come after H_N.