# On the Minimum Delay Peer-to-Peer Video Streaming: how Realtime can it be?

Yong Liu
Electrical and Computer Engineering
Polytechnic University
Brooklyn, NY, 11201, USA
yongliu@poly.edu

## ABSTRACT

P2P systems exploit the uploading bandwidth of individual peers to distribute content at low server cost. While the P2P bandwidth sharing design is very efficient for bandwidth sensitive applications, it imposes a fundamental performance constraint for delay sensitive applications: *the uploading bandwidth of a peer cannot be utilized to upload a piece of content until it completes the download of that content.* This constraint sets up a limit on how fast a piece of content can be disseminated to all peers in a P2P system. In this paper, we theoretically study the impact of this inherent delay constraint and derive the minimum delay bounds for realtime P2P streaming systems. We show that the bandwidth heterogeneity among peers can be exploited to significantly improve the delay performance of all peers. We further propose a simple *snow-ball streaming* algorithm to approach the minimum delay bound in realtime P2P video streaming. Our analysis suggests that the proposed algorithm has better delay performance and more robust than existing tree-based streaming solutions. Insights brought forth by our study can be used to guide the design of new P2P systems with shorter startup delays.

## Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Applications; C.2.4 [**Distributed Systems**]: Distributed Applications; C.4 [**Performance of Systems**]: Design Studies

## General Terms

Algorithm, Design, Performance

## Keywords

Peer-to-Peer Streaming, Realtime, Delay Bound

## 1. INTRODUCTION

Video-over-IP applications have recently attracted a large number of users on the Internet. Youtube [13] alone hosted some 45 terabytes of videos and attracted 1.73 billion views by the end of August 2006. While Youtube employs content delivery networks to stream video to end users, Peer-to-Peer (P2P) video streaming solutions utilize the uploading bandwidth of end users to distribute video content at low infrastructure cost. Several P2P streaming systems have been deployed to provide on-demand or realtime video streaming over the Internet [2, 15, 9, 10]. While the initial successes of P2P streaming are impressive, compared with the traditional TV services provided by cable companies, all current P2P streaming systems suffer from longer video startup delays, namely, the lag between a video object is chosen by a user and the actual playback starts on his/her screen. Our recent measurement study [3] on a popular P2P streaming system showed that the playback delays range from tens of seconds to a couple of minutes.

In traditional server-client video streaming systems, the video startup delay perceived by a client is determined by the delay and bandwidth variations on the its connection with the server. P2P systems exploit the uploading bandwidth of individual peers to distribute content at low server cost. While the P2P bandwidth sharing design is very efficient for bandwidth sensitive applications, such as file sharing, it imposes a fundamental performance constraint for delay sensitive applications: *the uploading bandwidth of a peer cannot be utilized to upload a piece of content until it completes the download of that content.* This constraint sets up a limit on how fast a piece of content can be disseminated to all peers.

In this paper, we theoretically study the impact of this inherent delay constraint and derive the minimum delay bounds for realtime P2P streaming systems. Our analytical results unveil the impact of the bandwidth distribution among peers on their delay performance. We show that the bandwidth heterogeneity among peers can be exploited to improve the peers' delay performance. Even a very small percentage of super peers can significantly reduce the video start-up delays for all peers. We further propose a simple *snow-ball streaming* algorithm to approach the minimum delay bound in realtime P2P video streaming. The delay performance of the proposed algorithm is compared with existing tree-based streaming solutions. Our analysis indicates that the proposed snow-ball streaming algorithm not only can achieve a close-to-optimum delay performance, but also has the potential to do so in face of realistic network impairments, such as long propagation delays and random bandwidth variations. The delay bounds derived in our analysis can serve as delay performance benchmarks for various proposed/deployed P2P streaming systems. Insights brought forth by our study can be used to guide the design of new P2P streaming systems with shorter start-up delays.

The paper is organized as follows. In Section 2, we provide a short overview on the existing P2P streaming solutions. The bounds on the delay for a single chunk dissemination is established in Section 3 for both homogeneous and heterogeneous P2P net-


*MM'07*, September 23–28, 2007, Ausburg, Bavaria, Germany.

work environments. A snow-ball chunk dissemination algorithm is introduced to achieve the delay bound for a single chunk dissemination. In Section 4, we show that the snow-ball chunk dissemination algorithm can be extended to a snow-ball streaming algorithm to achieve the delay bounds in continuous video streaming. The performance of the snow-ball algorithms under realistic network environment is analyzed in Section 5. The paper is concluded with future works in Section 6.

## 2. BACKGROUND AND RELATED WORK

Existing P2P streaming solutions can be classified into the following categories.

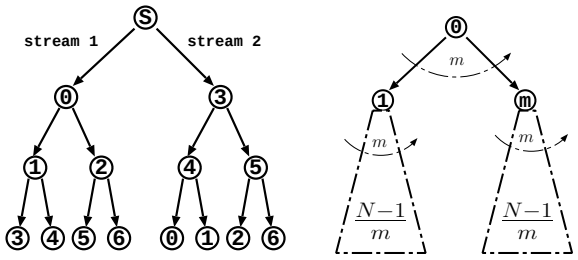
### 2.1 Single-Tree Streaming

In a single-tree based approach, peers are organized into a tree rooted at the server. Each peer receives the stream from its parent peer and forward to its children peers. The fan-out degree of a peer is limited by its uploading bandwidth. An early example is Overcast [4]. One major drawback of single-tree approach is that all the leaf nodes don't contribute their uploading bandwidth. They account for a large ratio of peers in the system. This largely degrades the peer bandwidth utilization efficiency.

### 2.2 Multi-Tree Streaming

To solve the leaf nodes problem, Multi-Tree based approaches have been proposed [1, 5]. In multi-tree streaming, the server divides the stream into $m$ sub-streams. Instead of one streaming tree, $m$ sub-trees are formed, one for each sub-stream. In a fully balanced multi-tree streaming, the node degree of each sub-tree is $m$. Each peer joins all sub-trees to retrieve sub-streams. Any peer is positioned on an internal node in only one sub-tree and only uploads one sub-stream to its $m$ children peers in that tree. In each of the remaining $m-1$ sub-trees, the peer is positioned on a leaf node and downloads a sub-stream from its parent peer. Figure 1(a) shows an example of two-tree streaming for 7 peers.

For any tree-based streaming approach, a chunk is disseminate in a hierarchical way. As illustrated in Figure 1(b), for a $m-$degree tree of $N$ peers, peer 0 sends a chunk to its $m$ children peers at level 1, each of which then is responsible for disseminating the chunk in its own subtree with $(N-1)/m$ peers (including themselves). In terms of time, after $m$ transmissions by peer 0, the task of disseminating a chunk to $N$ peers becomes $m$ sub-tasks of disseminating the chunk to $\frac{N-1}{m}$ peers.




(a) 7-nodes example (b) hierarchical view

**Figure 1: Multi-Tree Based Streaming**

### 2.3 Mesh-based Streaming

The management of streaming trees is challenging in face of frequent peer churns. Mesh-based streaming systems are more robust against peer dynamics. Many recent P2P streaming systems adopt mesh-based streaming approach [15, 8, 12, 6, 14]. In a mesh-based system, there is no static streaming topology. Peers establish and terminate peering relationship dynamically. A peer may download/upload video from/to multiple peers simultaneously. A recent simulation study [7] suggests that mesh-based systems have superior performance than tree-based solution. However, in practice, the delay performance of mesh-based streaming is still not satisfactory. Our analytical results indicate mesh-based systems have a lower delay bound than that can be achieved by the optimal tree-based systems. One important motivation of the study presented in this paper is to provide some guidelines for the design of peering strategies and chunk scheduling schemes in mesh-based streaming systems.

Despite of P2P streaming systems' popularity, few studies have addressed their delay performance analytically. One related work was presented in [11]. Authors of [11] studied the trade-off between the server bandwidth cost, the maximum number of peers that can be supported, and the maximum number of streaming hops experienced by a peer. We study the optimal streaming strategy when the server only plays a minimum role in video uploading. The delay bounds obtained through our analysis is much tighter than that predicted in [11], and can be achieved by the proposed snow-ball streaming algorithm.

## 3. BOUND ON SINGLE CHUNK DISSEMINATION

In P2P video streaming systems, video content is disseminated in units of *chunks*. One chunk is normally transmitted in multiple packets [3]. Given a P2P system with a server and $N$ peers, one can ask the question: *If the server generates a chunk of content at time $t=0$, how does one disseminate that chunk to all $N$ peers in the shortest time possible?* The answer depends on the size of the chunk, available bandwidth and the propagation delays among all nodes in the system, including the server and all peers.

Without loss of generality, we can normalize the chunk size to be one, and choose the video streaming rate as the bandwidth unit. Consequently, the chunk transmission time on a unit bandwidth link is 1 time unit, which equals to the chunk playback time. For now, let's assume the propagation delay between any two nodes is dominated by the chunk transmission delay and thus can be ignored. We will take propagation delays into account in Section 5.1 when the transmission delay becomes small. Without using P2P dissemination, if the server has a bandwidth of $N$, the server can upload that chunk to all peers by $t=1$. However, this is not a scalable solution when $N$ is large. We are interested in the delay performance when peers upload chunks among themselves.

### 3.1 Homogeneous case

We start with a homogeneous case where the server and all peers have bandwidth of 1. This corresponds to the *Tit-for-Tat* case in Bittorrent where peers upload roughly the same amount of data as they download. We further assume that the server will upload only one copy of the chunk to one peer and won't participate the chunk dissemination afterward.

#### *3.1.1 Single-Tree Chunk Dissemination*

Given the unit bandwidth on all peers, a peer can only have one child. The only possible single-tree based streaming solution is a chain: the server uploads the chunk to peer 0, then peer 0 uploads it to peer 1, and so on until peer $N-2$ uploads it to peer $N-1$. The chunk propagates along the chain from the server to all peers in time $N$. The average delay is $(N+1)/2$.

### 3.1.2 Multi-Tree Chunk Dissemination

If the multi-tree approach with degree $m$ is employed, a chunk propagates from the server to all peers along a sub-tree with node degree of $m$. If there are $N$ peers, the number of levels of each subtree is $K = \lceil \log_m(N(m-1)+1) \rceil$. The only peer at level 0 downloads the chunk directly from the server, a peer at level $i$ then uploads a video chunk to $m$ children peers at level $i+1$. Let $N_i$ be the number of peers at level $i$. Then $N_i = m^i$, $0 \le i < K-1$, and $N_{K-1} = N - m^{K-2}$. Since each peer/server only has uploading bandwidth of 1, if the uploading is done in parallel, all children peers of one peer will receive the chunk $m$ time slots after their common parent receives the chunk. The peer at the top level can always receive the chunk from the server after one time slot. For parallel uploading, the peers at the very bottom level will receive the chunk in $(K-1)m+1$ time slots. The average delay among all peers is

$$\bar{D}_p(m) = \frac{1}{N}\sum_{i=0}^{K-1}(im+1)N_i \qquad (1)$$

When $N$ is large, the average delay and the worst-case delay are both of the form

$$D_p(m) = m\log_m N + o(1) = \frac{m}{\log_2 m}\log_2 N + o(1). \qquad (2)$$

The shortest delay is achieved at the degree

$$m^* = \underset{m}{\mathbf{argmin}} \quad D_p(m) = 3,$$

i.e., the server divides the stream into 3 sub-streams, and feeds each stream into one sub-tree with node degree of 3. The minimum delay, in both average and worst-case sense, is $1.89\log_2 N + o(1)$.

If the uploading is done sequentially, the first child peer will receive the chunk from its parent within 1 time slot, and the last child of a peer will receive the chunk after $m$ time slots. The longest delay at level $i$ is still $im+1$. Therefore the worst-case delay is still $(K-1)m+1$. A degree of 3 can achieve the minimum worst-case delay of $1.89\log_2 N + o(1)$. The average delay at level $i$ is $(m+1)/2$ time slots more than the average delay at level $i-1$. The peer at the top level can always receive the chunk from the server after one time slot. We can calculate the average delay among all peers as

$$\bar{D}_s(m) = \frac{1}{N}\sum_{i=0}^{K-1}(i(m+1)/2+1)N_i.$$

Again, when $N$ is large, the average delay is

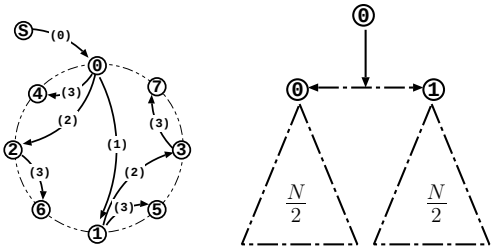
$$\bar{D}_s(m) = \frac{m+1}{2}\log_m(N) + o(1) = \frac{m+1}{2\log_2 m}\log_2 N + o(1).$$

To minimize the average delay, the optimal degree is 4, and the minimum average delay is $1.25\log_2 N + o(1)$, which is less than $2/3$ of the average delay of parallel uploading.

### 3.1.3 Snow-Ball Chunk Dissemination

For single chunk dissemination, peers only need to disseminate one chunk, instead of a continuous stream of chunks. After downloading the chunk, a peer can keep uploading that chunk to other peers until all peers receive it. This will largely reduce the chunk dissemination time. The accumulation of the aggregate uploading bandwidth for the chunk mimics the formation of a snow-ball. We refer it as the *snow-ball chunk dissemination* approach. Figure 2(a) illustrates the progress of snow-ball chunk dissemination for eight peers. An arc from node $i$ to node $j$ with a label $k$ represents peer $i$ (or the server) uploads the chunk to peer $j$ in time slot $k$. The server uploads the chunk to peer 0 in time slot 0. In time slot 1, peer 0 uploads the received chunk to peer 1. In time slot 2, both peer 0 and peer 1 will upload the chunk to peer 2 and 3 respectively. Peer $0, 1, 2, 3$ will upload the chunk to peer $4, 5, 6, 7$ in time slot 3. It takes 4 time slots for all peers to receive the chunk.

For general cases, the snow-ball approach disseminates a chunk in a recursive way. As illustrated in Figure 2(b), after peer 0 sends a chunk to peer 1, the task of disseminating a chunk to $N$ peers becomes two sub-tasks of disseminating the chunk to $\frac{N}{2}$ peers. Peer 0 continues to lead one sub-task, and peer 1 becomes the leader for the other sub-task. Even though the task splitting degree is only 2, compared with degree $m$ in Figure 1(b), it happens after only 1 chunk transmission, instead of $m$ transmissions in Figure 1(b). We will show that the snow-ball branching is actually the fastest branching process.




(a) 8-nodes (b) recursive view

**Figure 2: Snow-ball Chunk Dissemination**

Let $x(i)$ denote the number of peers that have the chunk at the beginning of time slot $i$. In time slot 0, the server uploads the chunk to one peer, therefore, $x(1) = 1$. Afterward, every peer with the chunk will upload it to another peer in one time slot, we have $x(i) = 2 * x(i-1) = 2^{i-1}$. Therefore it takes $K^* = 1 + \lceil \log_2 N \rceil$ time slots for all $N$ peers receive the chunk. One peer receives the chunk after 1 time slot, $2^{i-2}$ peers receive the chunk after $i$ time slots $\forall 1 < i < K^*$, and $N - 2^{K^*-2}$ peers receive the chunk after $K^*$ time slots. The average delay performance is

$$\bar{D} = \frac{1}{N}\left\{1 + \sum_{i=2}^{K^*-1} i2^{i-2} + K^*(N - 2^{K^*-2})\right\}.$$

If $N = 2^{K^*-1}$, the average delay is: $\bar{D} = \log_2 N + \frac{1}{N}$.

THEOREM 1. *In a homogeneous P2P streaming system, the snow-ball chunk dissemination approach simultaneously achieves the minimum average peer delay and the minimum worst-case peer delay.*

*Proof:* For arbitrary chunk dissemination approach, let $x(i)$ denote the number of peers that have the chunk at the beginning of time slot $i$. Since the server will upload the chunk to the first peer at time slot 0, we always have $x(1) = 1$. $x(i)$ is necessarily a non-decreasing function of $i$. We define a peer delay function $T(k)$ as the delay for the $k$-th peer to receive the chunk. Then the worst-case peer delay is $T(N)$ and the average delay is $\sum_{k=1}^{N} T(k)/N$. Given $\{x(i), i \ge 1\}$, $T(k)$ can be calculated as

$$T(k) = \min\{i : x(i) \ge k\}, \qquad 1 \le k \le N. \qquad (3)$$

Due to the homogeneous unit uploading bandwidth among peers, we always have $x(i+1) \le 2x(i)$, i.e., a peer can at most upload the chunk to another peer within one time slot. By induction, $x(i) \le$

$2^{i-1}$. For the snow-ball approach, $x^*(i) = 2^{i-1}$. Therefore, for any other chunk dissemination approach, $x(i) \leq x^*(i), i \geq 1$. Let $T^*(k)$ be the peer delay function for the snow-ball approach. Since $x(i) \leq x^*(i), i \geq 1$, due to (3), we have $T^*(k) \leq T(k), k \geq 1$. Therefore, the snow-ball chunk approach simultaneously achieves the shortest average and worst-case chunk dissemination delay. ■

Table 1 compares the delay performance of snow-ball chunk dissemination with tree-based and the optimal multiple-tree approach: For a system of 1024 peers, if the transmission delay of a chunk is 0.2 second, it takes only 2 seconds for the snow-ball approach to complete chunk dissemination to all peers, while the delay achieved by the optimal multi-tree approach is 3.8 seconds. Since the single-tree approach degrades to a chain, peers' average delay is around 100 seconds.

In the snow-ball approach, peers who receive the chunk in the $k$-th time slot upload the chunk for $K_{min} - k$ times, the peers who receive the chunk in the last time slot (about half of the peers) don't get a chance to upload the chunk to other peers. Their uploading bandwidth can be utilized to upload other chunks in continuous video streaming when multiple chunks are in transition simultaneously. We will further show in Section 4 that the snow-ball chunk dissemination can be extended to *snow-ball continuous streaming* to continuously disseminate a stream of chunks and the worst-case delay for each chunk is still $1+\lceil \log_2 N \rceil$. The snow-ball streaming in Section 4 is designed in an optimal way such that the uploading bandwidth of all peers are fully utilized to achieve the minimum delay bound for each chunk.

### 3.1.4 Effect of Increasing Server Bandwidth

If the server bandwidth is increased from 1 to $C$, we can divide $N$ peers into $C$ clusters, and let the server upload the chunk to one peer in each cluster within one time slot. Then, within each cluster, we can employ tree, multi-tree or snow-ball approach to disseminate the chunk. For the chain approach, the delay can be reduced by a factor of $C$. However, for both multi-tree and snow-ball approach, the improvement is only a constant proportional to $\log_2 C$. If the server participates in the snow-ball dissemination, at each round, the server can upload the chunk to $C$ peers. Let $z(k)$ be the number of nodes (including the server) with the chunk at the beginning of time slot $k$,

$$z(k) = 2 * z(k-1) + C - 1 = (2^k - 1)C + 1.$$

The finish time is $\lceil \log_2(\frac{N}{C} + 1) \rceil$. The delay improvement is still bounded by $\lceil \log_2 C \rceil$.

### 3.1.5 Effect of Increasing Peer Bandwidth

Secondly, if we also increase the bandwidth of each peer from 1 to $C$, in the tree based approach, the server can simultaneously upload to $C$ peers within one timeslot, and each peer can also upload to $C$ peers within one timeslot. Therefore, we can construct a streaming tree rooted at the server with node degree of $C$. The delay performance can be calculated in a similar way of the Multi-Tree case. If parallel uploading is employed, both the average and worst-case delay is $\log_C(N) + o(1)$. If sequential uploading is employed, the worst case delay is still $\log_C(N) + o(1)$, the average delay can be reduced to $\frac{C+1}{2C} \log_C(N) + o(1)$.

Multi-Tree approach can still be utilized. Now all the uploading from a peer to its children can be accelerated by a factor of $C$. The delays can be reduced to $1/C$ of the unit bandwidth case. Then the optimal degrees for parallel uploading and for sequential uploading remain to be 3 and 4 respectively.

For the snow-ball algorithm, at each time slot, each peer can upload to $C$ peers simultaneously, therefore,

$$x(k) = (C+1) * x(k-1) = (C+1)^{(k-1)}.$$

The finish time is $K_{min} = \lceil \log_{C+1} N \rceil + 1$.

In the previous calculation, we assume a peer uploads the chunk simultaneously to $C$ children peers. All $C$ children peers will receive the chunk at the end of the time slot. From the study of multi-tree approaches, we learned that sequential uploading can achieve better average delay performance than parallel uploading. We can adopt sequential uploading in snow-ball approach. A peer uploads the chunk to $C$ other peers *sequentially*, so that the peer receiving the chunk first can immediately upload to other peers without wait for the next time-slot. The delay performance in this case is actually $\frac{\lceil \log_2 N \rceil + 1}{C}$. This is because, with bandwidth $C$ and sequential upload, each peer can finish the upload of one chunk within $1/C$ time slot. If we change the time unit to be $1/C$ of the original time unit, the server and peer bandwidth becomes 1, we go back to the homogeneous case in Section 3.1 , all peers can receive the chunk within $\lceil \log_2 N \rceil + 1$ small time slots, which is $\frac{\lceil \log_2 N \rceil + 1}{C}$ original time slots.

## 3.2 Heterogeneous Cases

In real network environment, different peers have different types of network access, therefore, different uploading bandwidth. From the study so far, the chunk dissemination delay is determined by how quickly peers' bandwidth can be utilized to upload the chunk. We define the system-wide *usable uploading bandwidth* $\mathcal{U}(t)$ for the chunk as the aggregate uploading bandwidth that can be utilized to upload the chunk at any time $t$. In the homogeneous case, every peer has the same uploading bandwidth. $\mathcal{U}(t)$ is proportional to the number of peers with the chunk $x(t)$. The order at which peers receive the chunk has no impact on how $\mathcal{U}(t)$ grows over time. However, in a heterogeneous environment, the order at which peers receive the chunk determines the growth speed of $\mathcal{U}(t)$, and consequently the chunk dissemination delay. For the quick growth of $\mathcal{U}(t)$, the intuition is to upload the chunk to peers with large uploading capacities first.

In this section, we study the impact of uploading bandwidth heterogeneity among peers on the chunk dissemination delay by studying several typical cases. It will become clear that the peer uploading bandwidth heterogeneity enables the snow-ball approach to achieve a shorter chunk dissemination delay than the homogeneous case.

### 3.2.1 Case 1: Super-peers and Free-riders

Suppose there are $N/C$ super peers that can upload at rate $C > 1$. All the remaining peers are free-riders and don't participate in the uploading. The chunk can be disseminated by the snow-ball approach to all $N/C$ super peers within $1 + \frac{1}{C}\lceil \log_2(N/C) \rceil$ time slots. Since all super peers can aggregately upload $N$ copies of the chunk within one time slot, they can upload the chunk to the remaining $(1-1/C)N$ free-riders in $1-1/C$ additional time slot. The total delay is $\frac{\lceil \log_2(N/C) \rceil - 1}{C} + 2$. In this case, the average uploading bandwidth of peers are $\bar{u} = 1$. If all peers have the average uploading bandwidth 1, the shortest delay is $\lceil \log_2 N \rceil + 1$, which is around $C$ times of the heterogeneous case. This shows that heterogeneity of peer uploading bandwidth helps reduce the chunk dissemination delay.

### 3.2.2 Case 2: Multi-level Bandwidth Hierarchy

In the previous case, peers form a two-level hierarchy according to their uploading contribution. A fraction of $1/C$ super peers with uploading bandwidth $C$ stay at the top level and feed video chunk

**Table 1: Minimum Delay Achieved by Different Streaming Strategies for Homogeneous Case**

| Peer Delay | Single-Tree | Multi-Tree, Parallel | Multi-Tree, Sequential | Snow-ball Chunk |
|---|---|---|---|---|
| average | $\frac{N+1}{2}$ | $1.89\log_2 N + o(1)$, m=3 | $1.25\log_2 N + o(1)$, m=4 | $\log_2 N + \frac{1}{N}$ |
| worst-case | $N$ | $1.89\log_2 N + o(1)$, m=3 | $1.89\log_2 N + o(1)$, m=3 | $\log_2 N + 1$ |

to the free-riders at the bottom level. In real network environment, peers can be clustered based on the types of their network access. In this case, we extend the two-level hierarchy to accommodate multiple levels and show that even a very small percentage of super peers can bootstrap the chunk dissemination.

Suppose there are $N_1$ super peers with bandwidth $C_1$, $N_1N_2$ medium peers with bandwidth $C_2$ and $N_1N_2N_3$ slow peers with bandwidth $C_3$. To quickly disseminate the chunk to all peers, the following chunk scheduling algorithm can be employed:

1. use the snow-ball algorithm to upload to $N_1$ super peers within time $1 + \frac{1}{C_1}\lceil \log_2 N_1 \rceil$;
2. each of those $N_1$ super peers acts as a server with bandwidth $C_1$ and uploads to $N_2$ other medium peers. As studied in Section 3.1.4, the uploading can finish within time $1+\frac{\lceil \log_2(N_2/C_1)\rceil}{C_2}$, now $N_1N_2$ medium peers have the chunk;
3. each of those $N_1N_2$ medium peers acts as a server with bandwidth $C_2$ and uploads to $N_3$ other slow peers within time $1+\frac{\lceil \log_2(N_3/C2)\rceil}{C_3}$, now $N_1N_2N_3$ slow peers have the chunk.

The total delay is

$$3 + \frac{\log_2 N_1}{C_1} + \frac{\log_2 (N_2/C_1)}{C_2} + \frac{\log_2 (N_3/C_2)}{C_3}.$$

Without those super and medium peers, the fastest chunk dissemination to $N_1N_2N_3$ slow peers takes time $1+\frac{1}{C_3}(\log_2 N_1+\log_2 N_2+\log_2 N_3)$.

This suggests that the existence of super peers (even if a very small percentage) can dramatically reduce the chunk dissemination delay. For example, to disseminate a chunk to $32k = 2^{15}$ peers with bandwidth 1 need at least 15 time slots. Meanwhile, if $N_1 = N_2 = N_3 = 32$, and $C_1 = 10$, $C_2 = 5$, $C_3 = 1$, in other words, 32 (only 0.1%) of them have bandwidth of 10 and 1024 (only 3%, $N_2$=32) of them have bandwidth of 5, the time to disseminate a chunk to all $33k$ peers is less than 5.2 time slots. The example can be easily extended to incorporate more than 3 levels. Another insight obtained from this example is that: peers should be organized into tiers according to their uploading bandwidth, peers within each tier should help each other to obtain the chunk in the shortest possible time, then pass it down to the neighboring lower tier. This way, the delay of dissemination to the whole network can be reduced.

### *3.2.3 General Heterogeneous Case*

For general heterogeneous cases, one can index peers according to the decreasing order of their uploading capacities. Suppose the sorted uploading capacities of peers are: $u_1, u_2, ...u_N$. To derive a lower bound on the shortest chunk dissemination time, let's allow chunk stripping, namely, multiple peers can upload different portions of a chunk to the same peer simultaneously. If the first $k$ peers have the chunk at time $t$, the uploading to peer $k+1$ can finish by $\frac{1}{\sum_{j=1}^{k} u_j}$, therefore the lower delay bound can be calculated as

$$\hat{D} = 1 + \sum_{i=1}^{N-1} \frac{1}{\sum_{j=1}^{i} u_j}.$$

However, this is a loose bound. For example, for the homogeneous case, the bound is $\hat{D} = 1 + \sum_{i=1}^{N-1} \frac{1}{i} \leq 2 + \ln(N-2)$. We know the shortest delay without chunk stripping is instead $1 + \log_2 N$. In this section, we study several variations of the snow-ball algorithm to accelerate the chunk dissemination in general heterogeneous cases.

*Heterogeneous Parallel Snow-ball Approach:*

Assume $\{u_i\}$ are all integers. let $x(k)$ be the number of peers with the chunk at the beginning of time slot $k$.

1. In time slot 0, the server uploads the chunk to peer 1. x(1)=1;
2. In time slot $k \geq 1$, any peer with ID $j$, $1 \leq j \leq x(k)$, uploads the chunk in parallel to peers with IDs from $x(k) + \sum_{i=1}^{j-1} u_i$ to $x(k) + \sum_{i=1}^{j} u_i$;
3. $x(k+1) = x(k) + \sum_{i=1}^{x(k)} u_i$. If $x(k+1) < N$, $k = k+1$, go back to step 2; otherwise finishes.

This way, peers with larger uploading bandwidth will receive the chunk first and continuously upload the chunk to other peers until all peers receive the chunk. Let $\bar{u} = (\sum_{i=1}^{N} u_i/N)$ be the average uploading bandwidth among peers. Since peers are sorted according to the decreasing order of their uploading capacities, we have

$$x(k+1) = x(k) + \sum_{i=1}^{x(k)} u_i \geq x(k) + x(k)\bar{u} = (\bar{u}+1)x(k).$$

By induction, we will have $x(k) \geq (\bar{u}+1)^{k-1}$. Therefore the finish time is less than $\lceil \log_{\bar{u}+1} N \rceil + 1$, which is the delay of the parallel snow-ball approach in a system with homogeneous peer uploading bandwidth of $\bar{u}$ as studied in Section 3.1.5. This again demonstrates that snow-ball chunk dissemination approach has even better delay performance when peers have heterogeneous uploading bandwidth.

In this approach, due to parallel uploading, peers receive the chunk at the end of some time slot. Since we know sequential uploading has superior delay performance than parallel uploading, we can also develop a sequential snow-ball approach for heterogeneous systems. After receive the chunk, a peer will continuously upload it to other peers one after another. Since peers have different uploading bandwidth, the finish time of chunk uploading by different peers are no longer aligned. This makes it difficult to coordinate the uploading scheduling among peers. Here we develop a greedy snow-ball scheduling algorithm to achieve short delays in heterogeneous uploading.

*Heterogeneous Sequential Snow-ball Approach:* Again, index peers in the decreasing order of their uploading capacities. At any time instant $t$, let $E(t)$ be the ordered set of peers without the chunk, and $U(t)$ the ordered set of uploading peers. At any time, the status of a peer in $U(t)$ can be in either *busy*, meaning it is uploading the chunk to some peer, or *ready*, meaning it is available for next uploading.

1. Initialization: $U(1) = \{1\}$, set peer 1's status to *ready*; $E(1) = \{2, \cdots, N\}$;
2. Choose the first peer $i$ in the ordered set $U$ with status *ready*, pick the first peer $j$ from the ordered set $E$, let peer $i$ upload

the chunk to peer $j$ using its uploading bandwidth $u_i$, set peer $i$'s status to *busy*, and remove peer $j$ from set $E$. Repeat this step until either no peers are *ready* in $U$ or $E$ is empty;

3. After peer $i$ completes the uploading to peers $j$, add $j$ to $U$, set $j$'s status to *ready*, also set peer $i$'s own status to *ready*. If $E$ is not empty yet, go back to step 2.

However, due to the misalignment of the finish time of uploading events, this algorithm cannot guarantee to achieve the minimum delay. For example, for a system with 5 peers, if peer 1's uploading bandwidth is 10, other peer uploading capacities are 1, 1, 1, 1. When peer 1 finishes the upload to peer 2, peer 1 will upload the chunk to peer 3, and peer 2 will upload the chunk to peer 4. Then peer 4 will receive the chunk after 1.1 time slots. However, if we just let peer 1 upload the chunk to all other peers, every peer can get the chunk by 0.4 time slots. It is possible to develop an optimal uploading schedule for peers by carefully calculating the finish time instants for all possible upload combinations for all peers. We skip the discussion here.

# 4. SNOW-BALL STREAMING

In single chunk dissemination, any peer can be utilized to upload the chunk after it has downloaded the chunk. In continuous streaming, one new chunk is generated every time slot. When the server capacity is less than $N$, one chunk cannot be disseminated to all pees within one time slot. Therefore, there will be more than one chunk in transition at any given time. If $K^*$ is the minimum transmission delay for a single chunk, there will be at least $K^*$ chunks in transition at any given time. If the chunk scheduling is not set up appropriately, some chunks cannot be disseminated to all peers within $K^*$ time slots.

## 4.1 Homogeneous Environment

In this section, we show that, for the homogeneous case, it is possible to set up a chunk schedule such that all chunks can be disseminated to all peers within the minimum delay time. In the snow-ball chunk dissemination approach, the server uploads the chunk to the first peer at time slot 0. Before the beginning of time slot $K^* = \lceil \log_2(N) \rceil + 1$, all $N$ peers will receive the chunk. Let $\phi(j)$ be the number of peers that have the chunk at the beginning of time slot $j$ and will upload that chunk in time slot $j$. We have

$$\phi(j) = \begin{cases} 2^{j-1} & 1 \le j \le K^* - 2 \\ N - 2^{\lceil \log_2(N) \rceil - 1} & j = K^* - 1 \\ 0 & j \ge K^* \end{cases}$$

We call $\Phi^* \triangleq \{\phi(j) : j = 1 \cdots K^* - 1\}$ the *snow-ball chunk dissemination profile*.

THEOREM 2. *For a homogeneous P2P streaming system, there exists a continuous streaming schedule such that all chunks in the stream will be disseminated to all peers with the shortest delay $K^*$ achieved by the snow-ball algorithm for single chunk dissemination.*

*Proof:* Without loss of generality, the server uploads chunk $i \ge 0$ to some peer at time slot $i$. Let $y_i(k)$ be the number of peers that have chunk $i$ and will upload chunk $i$ to other peers at time slot $k$. For any feasible schedule, we should have $\sum_{i=0}^{\infty} y_i(k) \le N, \forall k$, i.e., at any time slot the aggregate uploading bandwidth for all chunks is at most $N$, and $y_i(k+1) \le 2 * y_i(k)$, i. e., each peer can upload to at most one peer within any time slot. A streaming schedule can achieve the optimal delay $K^*$ for each chunk if and only if each chunk can be uploaded according to the snow-ball chunk dissemination profile $\Phi^*$ after it is uploaded to some peer by the server, i. e.,

$$y_i(k) = \begin{cases} \phi(k-i) & (i+1) \le k < i + K^* \\ 0 & \text{otherwise} \end{cases}$$

It can be verified that such a schedule satisfies the feasibility constraints:

$$\sum_{i=0}^{\infty} y_i(k) = \sum_{i=k-K^*+1}^{k-1} y_i(k) = \sum_{j=1}^{K^*-1} \phi(j) = N - 1$$

and $y_i(k+1) \le 2 * y_i(k)$.

To complete the proof, for each time slot, we need to construct a uploading schedule for all active chunks. Let $\mathcal{S}$ be the set of all peers. Denote by $\mathcal{S}_i(k)$ the set of peers that have chunk $i$ at the beginning of time slot $k$ and will upload the chunk to $|\mathcal{S}_i(k)|$ other peers without chunk $i$ in the time slot. To follow the optimal dissemination profile $\Phi^*$, it is sufficient to have at each time slot $k$ $|\mathcal{S}_i(k)| = y_i(k)$ and $\{\mathcal{S}_i(k), i \ge 0\}$ are pairwise disjoint (since each peer can only upload one chunk in one time slot). We call the previous condition the sufficient condition $\Lambda$ to achieve the minimum delay streaming. We complete the proof of the theorem by constructing a chunk uploading schedule for each time slot through inductions:

**Initial condition:** *The server uploads chunk 0 to peer 0 in time slot 0. Therefore, at the beginning of time slot 1, $\mathcal{S}_0(1) = \{0\}$, and $\mathcal{S}_i(1) = \emptyset, i > 0$. It can be easily verified that the sufficient condition $\Lambda$ is satisfied at the beginning of time slot 1.*

**Induction:** *If at the beginning of time slot $k \ge 1$, the condition $\Lambda$ is satisfied, we can construct a schedule in time slot $k$, such that $\Lambda$ is still satisfied at the beginning of time slot $k+1$.*

At the beginning of time slot $k$, according to $\Lambda$, $k_o = \max(k - K^* + 1, 0)$ is the ID of the oldest chunk that needs to be uploaded in time slot $k$. Then $\mathcal{S}_i(k) = \emptyset, \forall i < k_o, \forall i \ge k$; and $\{\mathcal{S}_i(k), k_o \le i < k\}$ are pairwise disjoint, $|S_i(k)| = y_i(k)$. Define a set $\mathcal{F}(k) = \mathcal{S} - \cup_{i=k_o}^{k-1} \mathcal{S}_i(k)$, i.e., the set of peers that don't need to upload any chunk at the beginning of time slot $k$. The following scheduling will guarantee the $\Lambda$ condition is still satisfied at the beginning of time slot $k+1$.

**I.** If $k_1 = k - K^* + 1 \ge 0$, chunk $k_1$ will be uploaded for the last time in slot $k$. Since the chunk has been uploaded $1 + \sum_{i=1}^{K^*-2} \phi(i)$ times by the server and peers in the previous $K^* - 1$ time slots, only $\phi(K^* - 1)$ peers don't have it. Let all peers in set $\mathcal{S}_{k_1}(k)$ upload chunk $k_1$ to those peers and finish the upload of chunk $k_1$. Peers in $\mathcal{S}_{k_1}(k)$ can be used to upload other chunks in time slot $k+1$. We set $\mathcal{F}(k) = \mathcal{F}(k) \cup \mathcal{S}_{k_1}(k)$. Then $|\mathcal{F}(k)| \ge \phi(K^* - 1)$.

**II.** If $k_2 = k - K^* + 2 \ge 0$, chunk $k_2$ will be uploaded for the second-to-last time in slot $k$. According to $\Phi^*$, $\phi(K^* - 2)$ peers in set $\mathcal{S}_{k_2}(k)$ will upload chunk $k_2$ to other peers that don't have chunk $k_2$. In addition, the schedule should guarantee that there will be $\phi(K^* - 1)$ peers available in time slot $k+1$ to upload chunk $k_2$.

If $\phi(K^* - 1) \le \phi(K^* - 2)$, let each peer in $\mathcal{S}_{k_2}(k)$ upload chunk $k_2$ to any peer without chunk $k_2$, then pick $\phi(K^* - 1)$ peers out of $\mathcal{S}_{k_2}(k)$ to form the set of peers to upload chunk $k_2$ in next time slot, i.e., $\mathcal{S}_{k_2}(k+1)$. Other peers in $\mathcal{S}_{k_2}(k)$ can be used to upload other chunks in time slot $k+1$. We set $\mathcal{F}(k) = \mathcal{F}(k) \cup \mathcal{S}_{k_2}(k) - \mathcal{S}_{k_2}(k+1)$. We have $|\mathcal{F}(k)| \ge \phi(K^* - 2)$

If $\phi(K^* - 1) > \phi(K^* - 2)$, from step 1, $|\mathcal{F}(k)| \ge \phi(K^* - 1)$, we can take a subset $\mathcal{M}(k)$ of $\phi(K^* - 1) - \phi(K^* - 2)$ peers out of $\mathcal{F}(k)$, and let $\phi(K^* - 1) - \phi(K^* - 2)$ peers in $\mathcal{S}_{k_2}(k)$ upload chunk $k_2$ to peers in $\mathcal{M}(k)$. Remaining peers in $\mathcal{S}_{k_2}(k)$

then upload chunk $k_2$ to arbitrary peers without chunk $k_2$. Now peers in $\mathcal{M}(k)$ are ready to upload chunk $k_2$ in time slot $k+1$, therefore, we set $\mathcal{S}_{k_2}(k+1) = \mathcal{S}_{k_2}(k) \cup \mathcal{M}(k)$; $\mathcal{F}(k) = \mathcal{F}(k) - \mathcal{M}(k)$. We also have $|\mathcal{F}(k)| \geq \phi(K^* - 2)$.

**III.** Let $k_3 = \max(k - K^* + 3, 0)$. Any chunk $i$, $i \in [k_3, k-1]$, needs to be uploaded to $\phi(k-i)$ peers by peers in set $\mathcal{S}_i(k)$. We have

$$\sum_{i=k_3}^{k-1} |\mathcal{S}_i(k)| \leq \sum_{j=1}^{K^*-3} \phi(j) = \phi(K^* - 2) - 1 \leq |\mathcal{F}(k)| - 1. \quad (4)$$

Then $\forall i \in [k_3, k-1]$, take a subset $\mathcal{U}_i(k)$ of $|\mathcal{S}_i(k)|$ peers out of $\mathcal{F}(k)$, let all peers in $\mathcal{S}_i(k)$ upload chunk $i$ to peers in $\mathcal{U}_i(k)$, and set $\mathcal{S}_i(k+1) = \mathcal{S}_i(k) \cup \mathcal{U}_i(k)$, $\mathcal{F}(k) = \mathcal{F}(k) - \mathcal{U}_i(k)$. At the end, due to (4), we will have $|\mathcal{F}(k)| \geq 1$.

**IV.** The server uploads chunk $k$ to some peer $m_k$ in $\mathcal{F}(k)$, and set $\mathcal{S}_k(k+1) = \{m_k\}$.

Following the previous scheduling steps, the sufficient condition $\Lambda$ will be satisfied at the beginning of time slot $k+1$.

**Conclusion:** *There exists a schedule such that all chunks can be disseminated with snow-ball chunk dissemination profile $\Phi^*$ and achieve the optimal delay $K^*$.* ■

Figure 3 illustrates an example of snow-ball streaming in a system with 8 peers. We use a sequence of 6 subfigures to show the snow-ball chunk schedules among all peers within 6 consecutive time slots. Blocks represent chunks and circles represent peers. For time slot $k$, a white chunk beside a peer is the chunk that the peer has and will be uploaded to another peer within that time slot. An arc from peer $i$ to $j$ indicates peer $i$ uploads its chunk to peer $j$. A black chunk beside a peer indicates the server will inject that chunk to the peer in time slot $k$. Chunk 0 is uploaded to all peers by the end of time slot 3 and chunk 1 is uploaded to all peers by the end of time slot 4. The example shows that all chunks can be disseminated to all peers 3 time slots after it is injected by the server.

## 4.2 Heterogeneous Environment

For heterogeneous case, the delay bound for single chunk dissemination cannot always be achieved in streaming. For example, if the server's upload capacity is 1, and 7 peers' upload capacities are $2, 1, 1, 1, 1, 0$, a single chunk dissemination can be done in 3 time slots. However, no streaming algorithm can achieve this. If peer 0 is still uploading chunk 0 at timeslot 2, chunk 1 cannot be uploaded according to the greedy chunk profile $\Phi^*$. In this case, the first peer with bandwidth 2 becomes the scheduling bottle-neck for adjacent chunks. For the two special heterogeneous cases considered in Section 3.2, we are able to prove the existence of snow-ball streaming to achieve the minimum chunk dissemination delay for all chunks.

THEOREM 3. *For a P2P streaming system with $N/C$ super peers and $(1-1/C)N$ free-riders, there exists a continuous streaming schedule such that all chunks in the stream will be disseminated to all peers within a delay of $\frac{\lceil \log_2(N/C) \rceil}{C} + 2$ time slots.*

*Proof:* The idea is to first make sure all chunks can be streamed to all super peers within $1 + \frac{1}{C}\lceil \log_2(N/C) \rceil$ time slots. Then super peers will upload to free-riders whenever they have spare bandwidth. To achieve this, we change the time unit to $1/C$ of the original time slot. Measured in the new time slot, the server generates one new chunk every $C$ time slots. Suppose server only has uploading capacity of 1, and uploads chunk $i$ to some super peer by the end of time slot $C(i+1)$. For time slot $k$, let $y_i(k)$ be number of super peers uploading chunk $i$ to other super peers. To achieve the minimum streaming delay among all super peers, let $K^* = \lceil \log_2(N/C) \rceil$, we need

$$y_i(k) = \begin{cases} \phi(k - C(i+1)) & C(i+1)+1 \leq k \leq C(i+1) + K^* \\ 0 & \text{otherwise} \end{cases}$$

Let $i_1(k) = \lceil \frac{k-K^*}{C} \rceil - 1$ and $i_2(k) = \lfloor \frac{k-1}{C} \rfloor - 1$, $y_i(k) > 0$ iff $i_1(k) \leq i \leq i_2(k)$. Then

$$\sum_{i=0}^{\infty} y_i(k) = \sum_{i=i_1(k)}^{i_2(k)} \phi(k - C(i+1)) < N/C - 1$$

and $y_i(k+1) \leq 2 * y_i(k)$. According to Theorem 2, there exists a streaming schedule such that all super peers can receive the chunk within $1 + \frac{1}{C}\lceil \log_2(N/C) \rceil$ time slots. In addition, it can be shown that

$$\sum_{j=0}^{C-1} \sum_{i=0}^{\infty} y_i(k+j) = \sum_{j=0}^{C-1} \sum_{i=i_1(k+j)}^{i_2(k+j)} \phi(k+j-C(i+1)) = N/C - 1.$$

In other words, in any $C$ consecutive time slots, the aggregate number of uploading to super peers equals the number of super peers minus one. Since all super peers can upload $N$ times in $C$ time slots (one original time slot), therefore, we have $(1-1/C)N + 1$ spare super peer uploading available every $C$ time slots. After all super peers get chunk $i$ at time slot $C(i+1)+K^*$, in the following $C$ time slots, any super peer that is not responsible for uploading new chunks to other super peers can be utilized to upload chunk $i$ to a free-rider, and all free-riders can get the chunk by time slot $C(i+1) + K^* + C$. The achieved streaming delay is $2C + K^*$ sub-time slots, which is $\frac{\lceil \log_2(N/C) \rceil}{C} + 2$ original time slots. ■

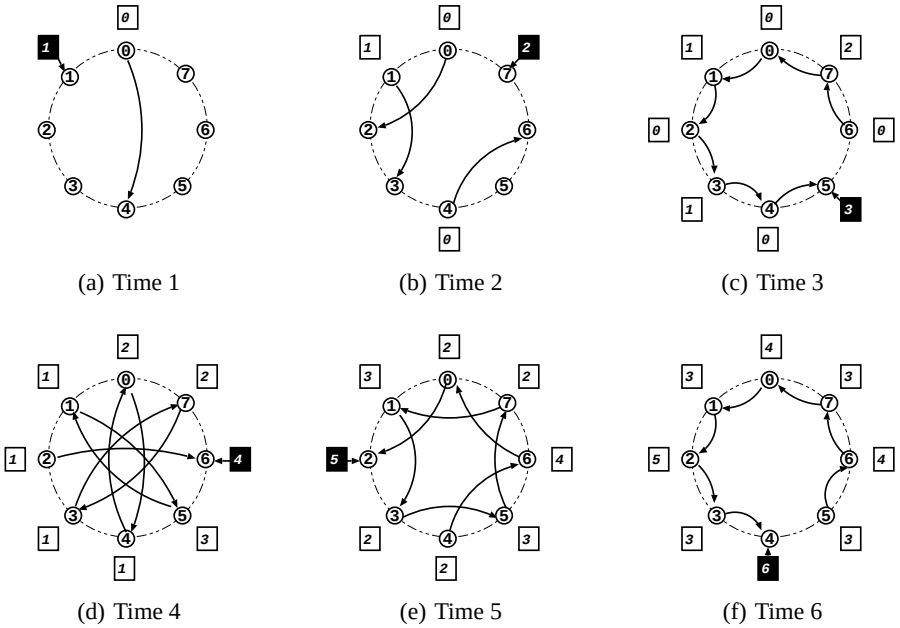
We list the chunk schedule for a system with 8 super-peers and 8 free-riders in Table 2. Super-peers are indexed from 0 to 7, each super peer has uploading capacity of 2, free-riders are labeled from $a$ to $h$. A tuple $(x, y)$ at row $i$ column $j$ means super peer $i$ will upload chunk $x$ to peer $y$ in time slot $j$. A chunk is uploaded to all super peers first, then it will be uploaded to all free-riders within one additional round. The overall chunk dissemination delay is 3.5 time slots.

**Table 2: Schedule between Super-peers and Free-riders**

| ID | 1 | 1.5 | 2 | 2.5 | 3 | 3.5 | 4 | 4.5 | 5 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0, 1 | 0, 2 | 0, 4 | 0, a | 2, 1 | 2, 2 | 2, 4 | 2, a | 4, 1 |
| 1 | | 0, 3 | 0, 5 | 0, b | | 2, 3 | 2, 5 | 2, b | |
| 2 | | | 0, 6 | 0, c | 0, g | 1, a | 2, 6 | 2, c | 2, g |
| 3 | | | 0, 7 | 0, d | 0, h | 1, b | 2, 7 | 2, d | 2, h |
| 4 | | | | 0, e | 1, 0 | 1, c | 1, g | 2, e | 3, 0 |
| 5 | | | | 0, f | 1, 1 | 1, d | 1, h | 2, f | 3, 1 |
| 6 | | | | 1, 4 | 1, 2 | 1, e | | 3, 4 | 3, 2 |
| 7 | | | 1, 6 | 1, 5 | 1, 3 | 1, f | 3, 6 | 3, 5 | 3, 3 |

COROLLARY 4. *If peers in a streaming system form a $M$-level hierarchy with $\prod_{k=1}^{i} N_k$ peers on level $i$ with uploading capacity of $C_i$, ($C_i > C_{i+1} \geq 1$), there exists a continuous streaming schedule such that chunks can be streamed to all peers with a delay of $M + \sum_{j=1}^{M} \frac{\lceil \log_2(N_i/(C_{i-1}-1)) \rceil}{C_i}$, where $C_0 = 2$.*

*Proof:* We can construct the chunk scheduling iteratively. Peers at level 1 pick $(C_1 - 1)N_1$ peers from level 2 as their free-riders. Construct a streaming schedule at level 1 according to theorem 3 such that $(C_1 - 1)N_1$ peers from level 2 will receive all chunks with delay $2 + \frac{\log_2(N_1)}{C_1}$. Then each of those peers can lead the



(a) Time 1 (b) Time 2 (c) Time 3

(d) Time 4 (e) Time 5 (f) Time 6

**Figure 3: Evolution of Chunk Scheduling of Snow-ball Streaming among $8$ Peers**

snow-ball streaming to $N_2/(C_1-1)$ peers at level 2 and $(C_2-1)N_2/(C_1-1)$ free-riders from level 3, by time $3+\frac{\log_2(N_1)}{C_1}+\frac{\log_2(N_2/(C_1-1))}{C_2}$, $(C_2-1)N_1N_2$ peers at level 3 will receive the chunk. They continue to do snow-ball streaming from level 3 to 4. The process can continue until all peers at the bottom level receive the chunk. ■

## 5. ANALYSIS IN REALISTIC SETTING

In this section, we analyze the performance of the proposed snow-ball algorithms in realistic P2P network settings.

### 5.1 Impact of Propagation Delays

From the analysis in the previous section, using smaller chunks in video streaming leads to smaller chunk transmission delay, consequently smaller dissemination latency. However, as the transmission delay getting smaller, the propagation delay will play a more important role. We still use the transmission time of a chunk as the time unit. Now suppose the propagation delay is $P=d-1$ time slots ($d\geq 2$). The time between a sender begins to upload the chunk and the receiving peer gets the whole chunk is $d$ time slots. For the multi-tree approach, if parallel uploading is employed, the chunk transmission delay from a peer to all its children increases from $m$ to $d+m-1$, the delay performance is $\frac{m+d-1}{\log_2 m}\log_2 N$; if sequential uploading is employed, the worst case delay is still $\frac{m+d-1}{\log_2 m}\log_2 N$, and the average delay is $\frac{2d+m-1}{2\log_2 m}\log_2 N$.

Again, denote by $x(k)$ the number of peers with the chunk at the beginning of time slot $k$. All the chunks received right before the beginning of time slot $k$ were sent out at the beginning of time slot $k-d$. Therefore we have

$$x(k)=x(k-1)+x(k-d).$$

$x(k)$ is a Fibonacci series with time lag $d$ ($d=2$ is the standard Fibonacci series). We can solve $x(k)$ by taking Z-Transform:

$$X(z)=\frac{Z^{-1}}{1-Z^{-1}-Z^{-d}},\rightarrow x(k)\sim\alpha_*^k,$$

where $\alpha_*$ is the largest root of $1-Z^{-1}-Z^{-d}$. The finish time is approximately $\log_{\alpha_*}N$, which is $\ln 2/\ln\alpha_*$ times of the snow ball delay without propagation delay. We plot the evolution of chunk dissemination at different propagation delays in Figure 4. Among them, $P=0$ corresponds to the case when the propagation delay is ignored as studied in Section 3. As predicted by the Z-transform

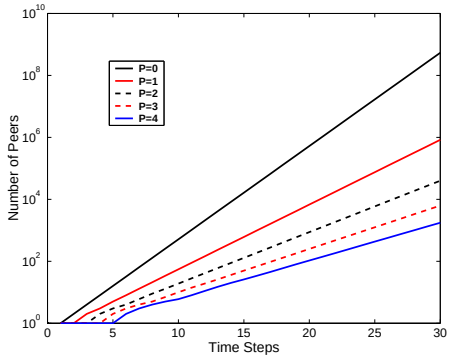



**Figure 4: Chunk dissemination speed at different propagation delays.**

analysis, the number of peers with the chunk grows exponentially after the first few time steps. For any propagation delay, the exponential growth rate, i.e., the slope of the curve in semi-log plot, is determined by the dominating root of $X(z)$. We compare the delay performance of multi-tree based strategies and the snow ball strategy in Table 3. The delay performance is measured in the unit of the average delay of snow-ball approach when there is no propaga-

tion delay. For parallel multi-tree strategy, we compute the optimal node degree that minimizes the average and worst-case delay at different propagation delays. For sequential multi-tree strategy, the node degree is optimized for the average delay and the associated worst-case delay is also calculated. As propagation delay increases, delay performance of all three strategies degrades. In addition, the optimal node degrees of multi-tree strategies also increase and a peer will spend more time to upload the same chunk to all its children. This makes it closer to the uploading philosophy of snow-ball streaming: *a peer should keep uploading the same chunk until all peers have it.*

**Table 3: Minimum Delay Achieved by Different Streaming Strategies with Propagation Delays**

| Prop. Delay | M-Tree, Para. | | M-Tree, Seq. | | | Snow Ball |
|---|---|---|---|---|---|---|
| | degree | delay | degree | average | worst | |
| 0 | 3 | 1.89 | 4 | 1.25 | 2.0 | 1.0 |
| 1 | 4 | 2.5 | 5 | 1.723 | 2.584 | 1.44 |
| 2 | 4 | 3.0 | 6 | 2.127 | 3.095 | 1.813 |
| 3 | 5 | 3.445 | 7 | 2.493 | 3.562 | 2.15 |
| 4 | 6 | 3.869 | 8 | 2.833 | 4.0 | 2.465 |

More generally, if the delays among peers are heterogeneous with distribution $(p_i, \Delta_i), 1 \leq i \leq M$, i.e., a chunk uploaded by a peer at the beginning of time slot $k$ will be received by a peer before the beginning of time slot $k + \Delta_i$ with probability $p_i$, we have, in average sense,

$$x(k) = x(k-1) + \sum_{i=1}^{M} p_i x(k - \Delta_i),$$

The average number of peers that receive the chunk in time slot $k$ can be calculated as:

$$X(z) = \frac{Z^{-1}}{1 - \sum_{i=1}^{M} p_i Z^{-\Delta_i}}.$$

Again, when $k$ is large, $x(k)$ grows exponentially.

## 5.2 Impact of Bandwidth Variations

In previous sections, we assume that peers have constant uploading bandwidth and a chunk transmission completes in constant time: in sequential transmission, a chunk can be transmitted from a peer to another peer in one time slot, in parallel transmission with degree $m$, a peer can transmit a chunk simultaneously to $m$ children in $m$ time slots. Due to network traffic variations, the available bandwidth on a connection between two peers varies over time. Consequently, the transmission time of a chunk is not constant. In this section, we investigate the robustness of different streaming strategies against the randomness in chunk transmissions.

For the clarity of presentation, we assume all transmission delays are independent and follow the same distribution. We introduce random variable $\tau^s$ for sequential transmission time, with $E[\tau^s] = 1$ and $Var[\tau^s] = \sigma^2$; the $m$-parallel transmission time is $\tau^p$, with $E[\tau^p] = m$ and $Var[\tau^p] = m\sigma^2$.

For the chain-based approach, if peer 0 receives the chunk from the server at time 0, the time for the $i$-th peer to receive the chunk is $\sum_{j=1}^{i} \tau_i$, where $\tau_i$ is the transmission delay from peer $i-1$ to $i$. And $\{\tau_i\}$ are i.i.d following the distribution of $\tau^s$. The worst-case delay $D_N^c$ is the time for peer $N$ to receive the chunk:

$$E[D_N^c] = N, \quad Var[D_N^c] = N\sigma^2.$$

The average delay among all peers is

$$\bar{D}_c = \frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{j}\tau_i = \frac{1}{N}\sum_{j=1}^{N}(N+1-j)\tau_i.$$

Then we have

$$E[\bar{D}_c] = \frac{N+1}{2}; \qquad Var[\bar{D}_c] = \frac{1}{N^2}\sum_{k=1}^{N} k^2\sigma^2 \sim \frac{N\sigma^2}{3}.$$

This suggests that, in a chain topology, the impact of the randomness of individual chunk transmission on the average and worst-case chunk delay performance of all peers is proportional to the number of peers $N$ in the chain.

For the parallel multi-tree approach, all peers at the bottom level will receive the chunk after $\log_m N$ independent parallel chunk transmissions. Then we have for worst-case delay:

$$E[D_N^p] = m\log_m N, \quad Var[D_N^p] = m\log_m N\sigma^2.$$

For the sequential multi-tree approach, there is one peer at the bottom level that will receive the chunk after $m\log_m N$ independent sequential chunk transmissions. We have for worst-case delay:

$$E[D_N^s] = m\log_m N, \quad Var[D_N^s] = m\log_m N\sigma^2.$$

Therefore the mean and variance of the worst-case delay for multi-tree based approaches are proportional to $m\log_m N$.

We can calculate the mean and variance of the average delay performance for multi-tree based approaches using recursions. As illustrated in Figure 1(b), a $m$-degree tree of $N$ peers consists of the single peer at level 0 and $m$ sub-trees, each of which is rooted at a level 1 peer and has $(N-1)/m$ peers. Denote by $\mathcal{W}(N)$ the aggregate chunk delay of all peers in a $m$-degree tree with $N$ peers after the root peer receives the chunk. Assume peer 0 received the chunk at time $t = 0$, let $t_j$ be the time when peer $j$ at level 1 receives the chunk. We have

$$\mathcal{W}(N) = \sum_{j=1}^{m}\left(\frac{N-1}{m}t_j + \mathcal{W}_j\left(\frac{N-1}{m}\right)\right), \qquad (5)$$

where the first term indicates the delay of peer $j$ contributes to the delays of all peers in its sub-tree, the second term is the aggregate delay to disseminate the chunk in the $j$-th subtree. For the parallel multi-tree approach, $\{t_j\}$ is just the parallel transmission time from peer 0 to $j$. They follow the distribution of $\tau^p$. For the sequential multi-tree approach, $t_j = \sum_{i=1}^{j}\tau_i$, where $\tau_i$ is the transmission delay from peer 0 to peer $i$, following the distribution of $\tau^s$.

The average delay of all peers is simply $\bar{D}(N) = \mathcal{W}(N)/N$. It can be verified that for both parallel and sequential multi-tree, $E[\bar{D}_p(N)]$ and $E[\bar{D}_s(N)]$ are the same as the deterministic case. Based on (5), we also calculate the variance $Var[\bar{D}_{p,s}(N)]$ recursively:

$$Var[\bar{D}_p(N)] \approx \frac{m}{m-1}\sigma^2, Var[\bar{D}_s(N)] \approx \frac{\sum_{j=1}^{m} j^2}{m(m-1)}\sigma^2 \quad (6)$$

In both cases, the impact of the variability of individual transmissions on the average delay performance is *independent* of the number of peers. And the average delay variance *won't diminishes* as $N$ grows. This is due to the variability at the first few transmission steps will affect almost all peers.

In the snow-ball approach, a peer will keep uploading a chunk until all peers have the chunk. Within one time period, a peer has more bandwidth will upload to more peers than a peer with less bandwidth. Over time, the workload of the same peer is naturally adaptive to its bandwidth: upload more if it has more bandwidth;

upload less if its bandwidth reduces. As for the recursive view in Figure 2(b), due to the workload self-adaptiveness, the number of peers in each subtree is no longer $\frac{N}{2}$. What remains to be true is that the uploading in both subtrees will finish around the same time.

To further illustrate, let's assume the chunk transmission time between two peers follows exponential distribution with mean 1. Denote by $\delta_k$ the time interval between the time instants when the $k$-th and the $k+1$-th peer receive the chunk. $\delta_1$ is the transmission time from peer 0 to peer 1, it is an exponential random variable with rate 1. For $k \geq 2$, due to the memoryless property of exponential distribution, $\delta_k$ follows an exponential distribution with rate $k$. Therefore the worst case delay is $D_N = \sum_{k=1}^{N} \delta_k$, which follows a hyper-exponential distribution. We have

$$E[D_N] = \sum_{k=1}^{N} \frac{1}{k} < 1 + \ln N, \quad Var[D_N] = \sum_{k=1}^{N} \frac{1}{k^2} < 2$$

The expected chunk dissemination finish time is only $\ln 2 = 69.3\%$ of the deterministic case. Due to the constant bounded delay variance, for large $N$, snow-ball approach has better delay performance in random case than in the deterministic case. Similarly, we can calculate the average delay performance as

$$E[\bar{D}] = \sum_{k=1}^{N} \frac{N-k+1}{Nk} < \frac{1+\ln N}{N} + \ln N \quad (7)$$

$$Var[\bar{D}] = \sum_{k=1}^{N} \left(\frac{N-k+1}{Nk}\right)^2 < \sum_{k=1}^{N} \frac{1}{k^2} < 2 \quad (8)$$

Again, the average delay performance is better than the deterministic case. This result is somehow counter-intuitive at the first sight. The study in Section 3.2 shows that *the bandwidth heterogeneity among peers will reduce the chunk dissemination delay.* The result obtained here can be considered as a *temporal* heterogeneity result, i.e., *the peer bandwidth variations over time will also reduce the chunk dissemination delay.* To bridge these two results, we can consider an artificial example: for a continuous streaming among $N$ peers over time period of $T$, divide $T$ into two halves, if in the first half peer 0 to peer $\frac{N}{2}-1$ have bandwidth of 2, peer $\frac{N}{2}$ to peer $N-1$ have no bandwidth; in the second half, peer 0 to peer $\frac{N}{2}-1$ have bandwidth of 0, peer $\frac{N}{2}$ to peer $N-1$ have bandwidth of 2. The average bandwidth of all peers are just 1. According to Theorem 3, the streaming delay of $0.5\log_2 N + 1.5$ can be achieved in both halves, while the minimum delay for the deterministic case when every peer always has uploading bandwidth of 1 is $\log_2 N + 1$.

# 6. CONCLUSION AND FUTURE WORK

In this paper, we analytically study the delay performance of P2P streaming systems. We derive delay bounds that can serve as delay benchmarks for proposed/deployed P2P streaming systems. Through our analysis, we quantify the impact of the bandwidth distribution among peers on their delay performance. Insights brought forth by our study can be used to guide the design of new P2P streaming systems with shorter start-up delays. A snow-ball streaming algorithm is proposed to achieve a close-to-optimum delay performance in various P2P network environment. Our preliminary analysis indicates that the snow-ball streaming algorithm is robust to network impairments, such as long propagation delays and random bandwidth variations. The next step is to implement the proposed snow-ball algorithm in a distributed mesh-based streaming system. We will test its performance in real network environment and compare it with the theoretical bounds predicted by our analysis here. Another direction for future work is to extend the delay performance analysis to take into consideration other factors, such as peer churns, geographic locality of peers and correlations among individual chunk transmissions, etc.

# Acknowledgments

The author thanks anonymous reviewers for their valuable comments. The author also thanks Chao Liang for his input. This work is partially supported by NSF grant CNS-0519998.

# 7. REFERENCES

[1] CASTRO, M., DRUSCHEL, P., KERMARREC, A.-M., NANDI, A., ROWSTRON, A., AND SINGH, A. SplitStream: High-bandwidth multicast in cooperative environments. In *Proceedings of ACM SOSP* (2003).

[2] CHU, Y.-H., G.RAO, S., AND ZHANG, H. A case for end system multicast. In *Proceedings of ACM SIGMETRICS* (2000).

[3] HEI, X., LIANG, C., LIANG, J., LIU, Y., AND ROSS, K. W. A measurement study of a large-scale p2p iptv system. *IEEE Transactions on Multimedia* (November 2007).

[4] JANNOTTI, J., GIFFORD, D. K., JOHNSON, K. L., KAASHOEK, M. F., AND O'TOOLE, JR., J. W. Overcast: Reliable multicasting with an overlay network. In *Proceedings of Operating Systems Design and Implementation* (2000), pp. 197–212.

[5] KOSTIŃC, D., RODRIGUEZ, A., ALBRECHT, J., AND VAHDAT, A. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proceedings of ACM Symposium on Operating Systems Principles* (2003).

[6] MAGHAREI, N., AND REJAIE, R. Prime: Peer-to-peer receiver-driven mesh-based streaming. In *Proceedings of IEEE INFOCOM* (2007).

[7] MAGHAREI, N., REJAIE, R., AND GUO, Y. Mesh or multiple-tree: A comparative study of live p2p streaming approaches. In *Proceedings of IEEE INFOCOM* (2007).

[8] PAI, V., KUMAR, K., TAMILMANI, K., SAMBAMURTHY, V., AND MOHR, A. Chainsaw: Eliminating trees from overlay multicast. In *The Fourth International Workshop on Peer-to-Peer Systems* (2005).

[9] PPLIVE. PPLive Homepage. `http://www.pplive.com`.

[10] PPSTREAM. PPStream Homepage. `http://www.ppstream.com`.

[11] SMALL, T., LIANG, B., AND LI, B. Scaling laws and tradeoffs in peer-to-peer live multimedia streaming. In *Proceedings of the 14th annual ACM international conference on Multimedia* (2006), pp. 539–548.

[12] VENKATARAMAN, J. C. V., AND FRANCIS, P. Multi-tree unstructured peer-to-peer multicast. In *Proceedings of 5th International Workshop on Peer-to-Peer Systems* (2006).

[13] YOUTUBE. Youtube Homepage. `http://www.youtube.com`.

[14] ZHANG, M., ZHAO, L., TANG, J. L. Y., AND YANG, S. A peer-to-peer network for streaming multicast through the internet. In *Proceedings of ACM Multimedia* (2005).

[15] ZHANG, X., LIU, J., LI, B., AND YUM, T.-S. P. DONet/CoolStreaming: A data-driven overlay network for live media streaming. In *Proceedings of IEEE INFOCOM* (2005).