Is there a future for mesh-based live video streaming?

Fabio Picconi and Laurent Massoulié
Thomson Technology Paris Laboratory
Boulogne-Billancourt, France
{fabio.picconi, laurent.massoulie}@thomson.net

Abstract

Peer-to-peer live streaming systems allow a bandwidth-constrained source to broadcast a video feed to a large number of users. In addition, a design with high link utilization can achieve high stream rates, supporting high-quality video. Until now, only tree-based designs have been shown to achieve close-to-optimal rates in real-life conditions, leaving the question open as to the attainable efficiency of completely unstructured mesh-based approaches.

In this paper we answer that question by showing that a carefully-designed mesh-based system can achieve close-to-optimal stream rates. Specifically, we implement and evaluate a design based on a mesh-based algorithm called DP/LU. Contrary to tree-based designs, DP/LU uses an unstructured overlay, which is easier to construct and is highly resistant to churn. In addition, we introduce mechanisms for overlay rewiring and source scheduling that lead to significant performance improvements.

Our experimental evaluation shows that our design achieves 95% of the maximum achievable stream rate in a static environment, and 90% under high churn. This demonstrates that mesh-based designs are an excellent choice for scalable and robust high-quality peer-to-peer live streaming.

1. Introduction

Live video streaming services are spreading quickly over the Internet. Video sharing websites such as YouTube attract millions of users per day, and a large number of TV channels are already available on the Internet or through IPTV services provided by ISPs. However, centralized systems require costly high-bandwidth links at the server. For instance, YouTube spends $1 million a day for the huge bandwidth needed by its servers [19].

Peer-to-peer live video streaming, or P2PTV [1, 2, 3, 4, 5, 6], greatly reduces the bandwidth requirements of the source by making users serve part of the stream to other peers downloading the same content. However, many existing P2PTV systems provide limited video quality (poor resolution, image freezes, etc.), while others require high-bandwidth proxies to achieve a high QoS. This poor quality is due to the limited capacity of most peers, but also to the use of simple streaming protocols which underutilize the clients' uplinks, wasting available bandwidth. This problem will become more serious as users will soon demand high definition video, increasing the required stream bitrate.

Motivated by these issues, recent research has focused on designing systems that achieve close-to-optimal rates [13, 15, 9]. However, most efforts have been directed towards tree-based approaches, while mesh-based ones have received less attention. Indeed, trees can result in highly efficient designs. Their main advantage is that multiple consecutive packets are pushed down the tree along the same paths, resulting in predictable traffic flows and low control traffic. Conversely, in a mesh-based system peers typically make different scheduling decisions for each packet, e.g., based on the number of packets already downloaded by the peer's overlay neighbors. As a result, per-packet paths and delivery times are highly variable. Designing a mesh algorithm that achieves high rates and meets playback deadlines despite such variability is a challenging task.

It is therefore not surprising that tree-based designs are becoming increasingly widespread. GridMedia [13], a widely deployed system, achieves near-optimal rates using a scheme that constructs a set of trees out of an unstructured overlay. CoolStreaming [1], another popular system, has recently switched from a mesh to a tree-based design similar to that of GridMedia. Furthermore, a recent Planetlab-based study [10] has even concluded that a mesh-based algorithm called DP/RU, which is analytically shown to be rate-optimal, cannot achieve high rates in real-life conditions. All this leads to the following question: will tree-based designs replace mesh-based ones for high-quality live video streaming?

We believe that the answer is no. To support this, we perform a study which shows that a carefully-designed mesh-based system can achieve near-optimal rates and low dif-

978-0-7695-3318-6/08 $25.00 © 2008 IEEE
DOI 10.1109/P2P.2008.18

fusion delays. Specifically, we implement and evaluate a design based on the DP/LU algorithm, which was recently shown, along with other algorithms, to reach close-to-optimal streaming rates in a simulation-based study [9]. In the process, we identify some optimizations that increase the performance of our prototype, and which are applicable to other mesh-based designs. We perform an extensive experimental evaluation in an emulated environment of up to 640 nodes. Our measurements show that our prototype reaches 95% of the maximum achievable rate, that is, a performance equal to or higher than that achieved by other non-mesh near-optimal designs [13, 15]. Furthermore, we show that our design is highly scalable and churn tolerant.

To summarize, the contributions of this paper are: 1) we provide experimental evidence that a mesh-based design can achieve near-optimal rates while tolerating high churn, concluding that mesh architectures are strong alternatives to tree-based ones for high-quality video streaming; 2) we identify some optimizations which help increase the rate performance of mesh-based designs; 3) we derive a general lower bound for the diffusion delay in heterogeneous capacity networks, and provide insights on how to optimize such delay in real live streaming designs.

The remainder of this paper is as follows: Section 2 presents related work. Section 3 describes bounds on delay and rate performance of any scheme. It then presents two efficient mesh-based algorithms, DP/RU and DP/LU. Section 4 discusses our design optimizations, and Section 5 shows our evaluation results. Section 6 concludes the paper and discusses future work.

2. Related work

Early work on peer-to-peer streaming [24, 27, 28] produced several designs based on single-tree, self-organizing overlays, showing that application-layer multicast was a valid alternative to native IP multicast. The main drawback of these systems was their vulnerability to churn, as well as a suboptimal link utilization. Thus, subsequent efforts focused on producing more robust and efficient architectures, typically dividing the stream into multiple substreams, and pushing each of them through a different tree. These multi-tree designs differ mainly in the way they construct the overlay. SplitStream [22] relies on Pastry [30], a structured routing algorithm used in Distributed Hash Tables. Conversely, Bullet [29] and Chunkyspread [23] construct trees out of an unstructured overlay.

More recently, a large number of tree-less (also known as mesh-based) systems have been widely deployed over the Internet [17, 4, 2, 3, 5] and studied in academic research [26, 25]. These designs are based on unstructured overlays in which no parent-child relationships exist between nodes. As a result, the overlay is easier to maintain and is highly resistant to churn. In fact, the unstructured overlay resembles that of file-sharing systems such as BitTorrent [12] and Gnutella [7], known to tolerate high levels of churn. However, unlike file-sharing applications, these systems employ algorithms which are designed for real-time delivery.

Despite the widespread deployment of mesh designs, recent efforts to optimize link utilization have focused on tree-based overlays. GridMedia [13] and CoolStreaming [1] both employ push-pull protocols that seek highly efficient, stable parent-child relationships. Another recent algorithm [15] has been shown to achieve high rates by spreading chunks over a set of $n + 1$ spanning trees of limited depth. Given the cost of all-to-all communications, scalability is ensured by organizing the overlay into a hierarchy of small clusters, each acting as a source for lower-level clusters [16].

While these studies have shown that tree-based designs can achieve close-to-optimal rates in real-life conditions, the rate efficiency of mesh-based designs has only been established through analysis or simulations [18, 9]. One of the contributions of this paper is to experimentally show that mesh-based systems can also achieve near-optimal streaming rates.

3. Optimal mesh-based algorithms

The main goal of live streaming is to deliver stream data to all clients before their playback deadline. Design efficiency is essentially characterized by the streaming rate that can be sustained, and by the delay it takes for data to reach clients from the source.

When the only bandwidth limitations are the uplink capacities, the maximum achievable streaming rate reads

$$r_{max} = \min\{u_s, \frac{u_s + \sum_{i=1}^{n} u_i}{n}\}, \qquad (1)$$

where u_s is the server's uplink capacity, u_i the capacity of client i, and n the number of clients (see Kumar et al. [14]).

In the case of homogeneous bandwidths (i.e., $u_s = u_i = u$), it has been shown in [9] that for any algorithm, decentralized or not, the delay D verifies $D \geq u^{-1}(\log_2(n) - O(1))$. In this expression, u is expressed in blocks per second, where blocks (i.e., chunks) are the atomic data unit exchanged between peers.

For heterogeneous environments, no simple characterization of the optimal delay is available to date. Therefore, it is impossible to say with our current knowledge whether any give scheme is delay-optimal or not.

We can however establish a general lower bound on the maximal delay D of any scheme. This takes the following form. Given arbitrary uplink bandwidths $u_s, u_i, i =$

$1, \ldots, n$, let $G(t)$ denote the corresponding maximal number of block copies that can be spread in the system, given that at time 0 only the source has one block. Then for any algorithm, and any streaming rate r, the maximal delay D verifies

$$D \geq \min(d \geq 0 : \ G(d) - G(d - 1/r) \geq n\}. \quad (2)$$

The detailed argument, with a formal definition of function G, are provided in the appendix. One insight we learn from the argument is that, to spread blocks at optimal speed G, one must target fast peers preferentially to slower ones.

We now present two mesh-based algorithms which can achieve optimal rates. More details can be found in references [18, 9]. Note that there exist other optimal algorithms which are not mesh-based [15, 14].

3.1. Most deprived peer, random useful chunk

Massoulié et al. [18] have proposed an epidemic live streaming algorithm, proven to be rate-optimal for fully connected overlays and arbitrary node bandwidths. The algorithm, called DP/RU (Deprived Peer / Random Useful), spreads chunks through the overlay in an epidemic, random-like fashion. Although the optimality proof assumes a complete graph, simulations show that DP/RU also achieves near-optimal rates when using a small-degree unstructured overlay [9].

The algorithm can be described as follows: let $P(u)$ denote the set of chunks that a node u has received at a given time. A node u periodically selects its *most deprived* neighbor, i.e., the neighbor which maximizes $|P(u) \setminus P(v)|$ for all $v \in S$, where S is the set of neighbors of u. Then, it uploads a *random useful* chunk to that node. The term *useful* indicates that the receiving node does not already possess the chunk.

DP/RU is push-based: peers actively push chunks to their neighbors whenever they can, i.e., when they have a free upload slot and possess at least one useful chunk. This is to be contrasted with pull-based systems, in which clients send download requests to their neighbors. Examples of pull-based systems are BitTorrent [12] and the original CoolStreaming algorithm [17].

In practice, $P(u)$ is computed over a small set of chunks which the peer is currently interested in downloading. If seq_{play} indicates the sequence number of the last chunk passed to the video player, the peer attempts to download chunks in the range $[seq_{play}+1, seq_{play}+win]$. This range is called the *download window*, and is a configurable system parameter. The download window length is typically set to 10-30 seconds worth of chunks.

Besides its proven rate optimality, DP/RU has the following desirable properties. Its use with small degrees allows to scale to large groups; since it does not rely on trees or structured overlays, it is resilient to churn; scheduling decisions are essentially done locally.

In contrast, the Adaptive Queue-based Chunk Scheduling algorithm (AQCS [15]), also proven rate-optimal, requires a fully connected overlay, limiting scalability of the system. GridMedia and CoolStreaming construct a set of trees, making the system more complex and more sensitive to churn. Finally, the Hierarchically Clustered P2P Streaming System [16] requires a centralized management node that maintains a global view of the network in order to keep clusters balanced.

That said, mesh-based systems such as DP/RU also have their disadvantages. Overhead is typically higher compared to other architectures due to frequent signaling traffic. In fact, each client must periodically inform its neighbors of the set of chunks that it possesses (this is known as the *buffer map*), so that useful chunks can be selected for transmission. Another problem, common to all push-based algorithms, is due to the presence of *collisions*. A collision occurs when two clients concurrently push the same chunk to the same node. This happens because the algorithm does not coordinate pushes among clients. Unfortunately, DP/RU is even more susceptible to collisions than completely randomized algorithms, due to its strategy of selecting the most deprived peer. Clearly, collisions should be avoided as they result in wasted bandwidth from redundant transmissions. In the next section we will see how to avoid them completely by switching to a pull-token mechanism.

3.2. Most deprived peer, latest useful chunk

Although DP/RU is known to be rate-optimal, simulations show that its delay performance is not satisfactory [9]. Furthermore, a Planetlab-based study [10] showed that a practical implementation of DP/RU does not achieve high streaming rates. The poor performance of DP/RU can be explained by its random chunk selection, which does not consider the chunk's position with respect to the playback deadline. This results in high delays, which in turn produce high chunk misses when the download window is small.

However, simulations show that a slight modification to the algorithm can significantly lower the propagation delay [9]. In this new algorithm, peers select the most deprived neighbor as in DP/RU, but push the *latest useful* chunk instead of a random useful one. Here latest means the chunk with the highest sequence number, i.e., the one most recently generated by the source. Following the same name convention, the algorithm is called DP/LU. Given its good rate and delay performance in simulations, we base our design on the DP/LU mechanism, to which we bring several improvements.

4. Optimizations

During the implementation and evaluation of our DP/LU prototype we identified a number of optimizations which were necessary to achieve high streaming rates. In this section we discuss some of them which are not specific to our design, and which may be useful to other mesh-based architectures.

4.1. Overlay management

In an overlay with limited degree, distributing high-capacity nodes uniformly across the system is crucial to deliver high streaming rates to all peers. Otherwise, the overlay could become unbalanced, leading to lower streaming rates and deadline misses in low-capacity regions.

We avoid this problem using an overlay management algorithm in which nodes select a set of neighbors according to their uplink capacities. The scheme works as follows: for simplicity, all peers have the same maximum degree d. Let us assume that each peer knows the uplink capacity u_i of its neighbor i, with $1 \leq i \leq d$. Each peer computes the average value U_N of all u_i. We call U_N the *average neighborhood upload capacity* of a peer. Intuitively, for the overlay to be bandwidth-balanced, all peers should have a similar value of U_N. If the source generates a constant bit-rate stream, we can use the stream rate r as a reference value for U_N, since r is known by all peers. Thus, each node periodically compares its U_N to r. If $U_B < r$, the peer picks the slowest neighbor, and replaces it with another randomly chosen node. Conversely, if $U_N > r$, the fastest neighbor is replaced. Thus, this scheme attemps to bring U_N close to r for all peers.

Note that this scheme balances the overlay even if the mean capacity of the system is different than r, as all nodes compare their U_N to the same reference value. However, it can also produce unnecessary disconnections by constantly rewiring the overlay. Two optimizations can reduce the number of disconnections: first, estimating the mean capacity c of the entire network, and comparing U_N to c instead of r; second, replacing neighbors only if $|U_N - c| > \alpha c$, with $0 < \alpha < 1$ (i.e., using a threshold).

In our current implementation, rewiring is based on nominal uplink bandwidths, that peers declare to their neighbors when establishing a connection. The nominal bandwidth of a peer represents the uplink limit specified by the user (most existing peer-to-peer applications allows users to specify this). An alternative could consist in making rewiring decisions on the basis of measured download rates. We leave the evaluation of such alternative schemes for future work.

	Uplink (kbps)	Fraction of nodes
class 1	4000	0.15
class 2	1000	0.25
class 3	384	0.4
class 4	128	0.2

Table 1. Uplink capacity distribution.

4.2. Source scheduling algorithm

During our initial experiments, we found that modifying the source strategy to give preference to faster peers increased the system performance. Moreover, our analysis of the minimum diffusion delay (see Section 3 and Appendix) suggests that spreading chunks preferentially to fast nodes produces delays close to the optimum. Based on these observations, we have implemented a source push strategy in which the number of chunks that a peer receives from the source is proportional to its upload capacity.

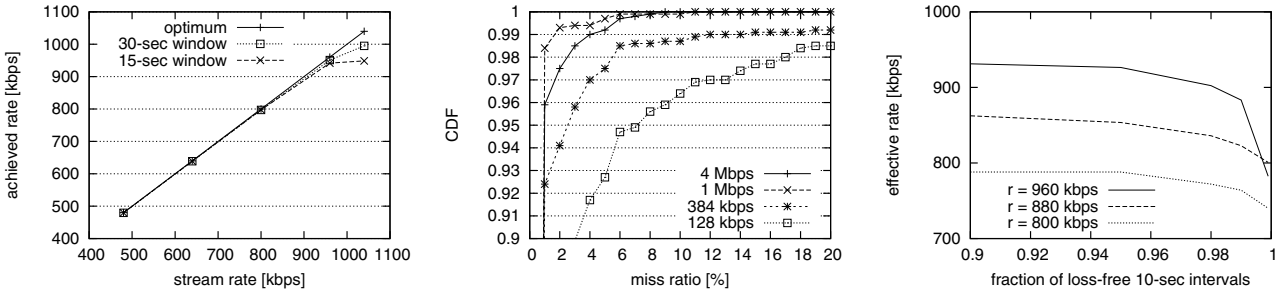
Similarly to the rewiring mechanism presented in the previous section, peers declare their nominal bandwidth to the source, which uses this information to make scheduling decisions. Note that this assumes that peers are honest, i.e., they do not try to free-ride the system by declaring a false capacity to the source. Other designs also make similar assumptions regarding peer-source exchanges [15]. Relaxing this assumptions is left for future work.

4.3. Pull mechanism

In their experimental study of DP/RU, Liang et al. [10] have pointed out the problem of collisions in push-based systems (cf. Section 3.1), and provided a solution based on a *pull token* mechanism. We implement a similar technique, which works as follows : let A be a peer which wishes to push data to one of its neighbors. First, A selects a neighbor B according to the *most deprived* policy. Then, it sends a *pull token* message to B, indicating that it is willing to transmit up to k chunks, where k depends on the A's available uplink bandwidth. Upon receiving the token, B identifies the *latest useful* chunks that it can download from A, and sends back a *pull request* message containing up to k chunk sequence numbers. Finally, A sends the chunks that B requested.

In some cases, a node B receiving a pull token from A may determine that A has less than k useful chunks. In this case, B replies with c chunk requests, where $c < k$. We say that the remaining $k - c$ chunk transfers have been *waived* by B. Upon receiving the reply, node A immediately issues a new pull token offering the $k - c$ chunk transfers waived by B to another peer.

The main difference between our scheme and that of



(a) delivery rate (b) 10-sec interval miss ratios (c) effective rate with erasure codes

Figure 1. (a) Average delivery rate reaches 95% of maximum. (b) Cumulative Distribution Function (CDF) of miss ratios measured within 10-sec intervals. (c) For a 99.9% robustness level with erasure codes, a raw bitrate of 880 kbps yields better performance than 960 kbps.

Liang et al. lies in the choice of k. In their mechanism k is fixed, and depends on the node's uplink capacity. Thus, a node only issues a pull token when it has enough available bandwidth to transmit k chunks. Conversely, in our design the same sender may issue tokens with different values of k. For instance, if a node waives $w = k - c$ chunks, the sender will immediately reissue a new token with $k' = k - c$ to another node. The rationale is that quickly reissuing tokens may reduce the delay of the pull token mechanism, decreasing the average propagation delay.

Our current prototype only implements our scheme with variable k. Thus, we have not compared the efficiency of the two pull token variants. Nevertheless, our measurements show that our mechanism achieves almost full link utilization even in the presence of a high number of waived chunks.

5. Evaluation

We have developed a DP/LU implementation, including the optimizations described in the previous section. The prototype is written in C++ and has roughly 6000 lines of code. We evaluate its performance by deploying it on a local cluster of 10 PCs plus an emulator machine. In order to increase the network size in our experiments we run up to 8 clients on each physical machine. Our machines are 2.0 GHz dual-Opterons, with a total of four cores per machine. Each client takes less than 5% of CPU time, so the impact of running several clients per core is negligible. We also perform a 640-client experiment on a bigger cluster with similar hardware, this time using 30 machines.

We emulate uplink capacities and wide-area latencies using Modelnet [8]. We set the uplink capacities according to the distribution of Table 1, which is based on a recent measurement study [11], and was also used by other authors to evaluate various live streaming systems [15]. The average uplink capacity of this distribution is around 1 Mbps. The downlink bandwidth is set to 10 Mbps for all nodes.

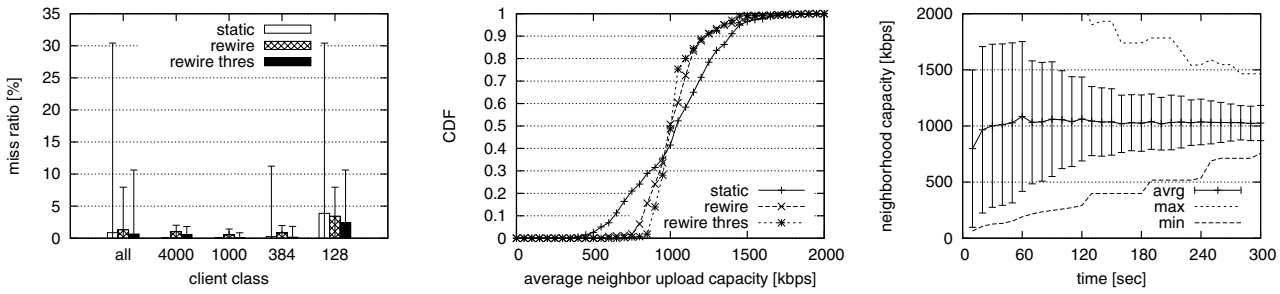
Unless otherwise noted, we use a base configuration of 80 clients and 1 source, which also works as tracker. The maximum node degree is set to 20, except for the source which is connected to all clients. The source's uplink is set to 1.1 Mbps, which yields a maximum sustainable stream rate of 1040 kbps according to Equation (1). The stream rate is set to 960 kbps, i.e., 90% of the maximum achievable rate. The download window is set to 30 seconds, and the chunk size to 10 KB. All nodes join simultaneously at the beginning of each experiment, and we let the overlay stabilize for 60 seconds. We then log for 5 minutes the number of chunks received and missed by each client.

5.1. Delivery rate

In this first experiment we measure the average chunk delivery rate for various stream rates. The delivery rate of a client is computed as the total amount of stream data received by the client, divided by the duration of the experiment. We then compute the average rate for all clients.

Figure 1(a) shows that the delivery rate is almost perfect for stream rates up to 960 kbps. At 1040 kbps, the maximum sustainable rate of the system, our prototype delivers around 90% of the packets with a 15-second download window, and 95% with a 30-second window. The 10% and 5% difference is due to chunk misses, i.e, chunks which are delivered to the clients after the playback deadline, and therefore are not useful to the video player. Clearly, increasing the download window increases the time available for peers to download a chunk, thus lowering the number of misses (see also Section 5.3).

Besides from the average delivery rate, it is also important to know how chunk misses are distributed among clients, and over time. For instance, if chunks misses are



(a) rewiring protocol (b) neighborhood capacity CDF (c) rewire protocol speed

Figure 2. Effects of the rewiring protocol: (a) worst miss ratio is significantly reduced thanks to a more bandwidth-balanced overlay. (b) The distribution of neighborhood capacities is less skewed. (c) A completely unbalanced overlay converges to a balanced configuration in around 300 seconds (10 rewiring intervals).

spread over time, then techniques such as erasure codes [21] or multiple description coding [22] can be used to limit the impact of chunk misses on video quality. Conversely, encoding techniques may not be effective if a client misses a large number of consecutive chunks.

To determine the distribution of missed chunks, we measure the clients' miss ratio within 10-second intervals. Figure 1(b) shows that faster clients do better than slower ones. For instance, clients with 1 Mbps uplink miss less than 2% of the chunks in 99% of the intervals, whereas 384 kbps clients may miss up to 12%. There are two reasons behind this effect: first, fast nodes receive more chunks from the source than slow nodes, so the average propagation delay from the source is lower; second, signaling packets (i.e., pull requests) experience higher delays in lower bandwidth clients, increasing the chunk propagation delay. Notice that 4 Mbps clients do slightly worse than 1 Mbps clients. This is due to the overlay rewiring algorithm, which disconnects either the fastest or the slowest neighbor of a client, which are usually 4 Mbps and 128 kbps nodes.

Figure 1(c) shows the effective bitrate that would result if erasure codes were applied to reconstruct missing chunks, using a download window of 15 seconds[1]. The x-axis represents the percentage of fully-recoverable 10-second intervals, i.e., a measure of the desired robustness of the system. Clearly, higher resilience requires more redundancy, reducing the effective bitrate. Note that for a resilience of 99.9%, the effective bitrate is higher when using a raw bitrate of 880 kbps instead of 960 kbps. The reason is that at 960 kbps, the worst 0.1% 10-second intervals exhibit a very high loss-rate, due to the system working very close to its maximal capacity. This suggests that in some cases where a high level of robustness is required it may be better to use a lower bitstream with a less redundant code.

[1] We do not actually implement erasure codes. Instead, we assume coding is applied within 10-second intervals, and we calculate the necessary redundancy needed to correct a given percentage of the stream.

5.2. Overlay rewiring

In this section we evalute the effectiveness of the overlay rewiring scheme. We compare three configurations: a static overlay, a dynamic overlay with the standard rewiring protocol, and the rewiring protocol with a 10% threshold (i.e., $\alpha = 0.1$). In all three cases, nodes initially connect to 20 neighbors chosen at random. The rewiring interval is set to 30 seconds, and peers use the streaming rate r as the reference value for the neighborhood capacity U_N. We perform three runs for each case and compute the average and maximum values of the miss ratio.

The bars in Figure 2(a) show the average miss ratio, whereas the error bars indicate the *worst* miss ratio among all clients. While all three cases achieve similar average miss ratios, the worst miss ratio drops from 30% in the static overlay to 10% in the dynamic one. With a static overlay, a client may end up connected to a majority of low-capacity neighbors, and thus be unable to download enough chunks from them. The rewiring protocol avoids this situation, by continuously adapting the overlay such that clients connect to both fast and slow neighbors. Notice also that using a 10% threshold decreases the average miss ratio, thanks to a lower number of disconnections.

The effect of the rewiring protocol can also be seen in Figure 2(b), which compares the distribution of average neighborhood capacities for the three cases at the end of the experiment. In the static case, some nodes end up with an average neighborhood capacity as low as 500 kbps. With rewiring enabled, nodes get neighborhoods of at least 700-800 kbps.

Finally, we evaluate the convergence speed of our rewiring protocol in the worst case scenario, that is, by starting with a completely unbalanced overlay where fast nodes

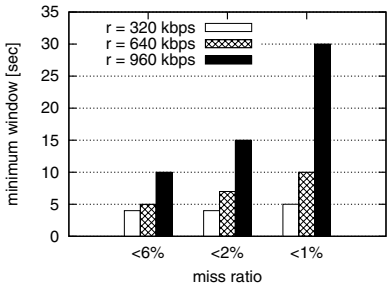


Figure 3. The minimum window for a given miss ratio decreases when more excess bandwidth is available.

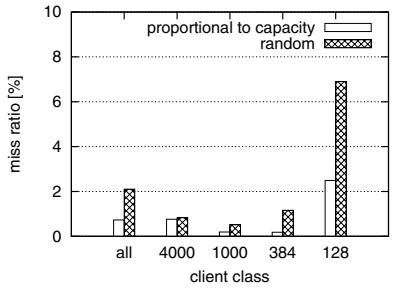


Figure 4. Source scheduling with preference to fast nodes increases performance.

are connected to other fast nodes only. Figure 2(c) shows the maximum, minimum and average values of the neighborhood capacities across the system over time. The system converges to a steady state in around 300 seconds, that is, 10 rewiring intervals.

5.3. Impact of window length and source strategy

First, we evaluate the effect of the download window size. Figure 3 shows the minimum download window needed to keep the miss ratio below 1%, 2% and 6%, for three different stream rates. As we saw in Figure 1(a), the miss ratio can be reduced by increasing the length of the window. However, we also observe that lower stream rates require smaller windows. In fact, lowering the stream rate means that the clients' uplink become underutilized. This, in turn, reduces the transmission delay of a chunk, as more bandwidth is available on each neighbor connection. Smaller delays reduce the propagation delay, allowing for a shorter download window.

Second, Figure 4 shows the effect of varying the source push strategy. We run two experiments, one where the source pushes chunks to randomly chosen peers, and the other where faster nodes receive more chunks from the source (cf. Section 4.2). The curve shows that favoring faster nodes improves performance significantly.

As discussed in Section 4.2, modifying the source strategy to favor fast peers brings our design closer to a delay-optimal diffusion scheme. This suggests that delay performance could be further increased by modifying the DP/LU strategy implemented by peers, as the *most deprived* peer selection does not consider uplink capacity. Thus, a possible optimization could consist in giving preference to deprived peers which also have high uplink bandwidths. We leave such optimizations for future work.

5.4. Scalability

We now evaluate the scalability of our design. First, we increase the overlay size up to 640 peers and measure the average miss ratio. Figure 5(a) shows that a near 10-fold increase has a very low impact on the average miss ratio, which increases from 0.6% to around 1.25%. This suggests that the propagation delay of our design grows slowly with the network size, a property which seems to be shared by several epidemic algorithms similar to DP/LU [18, 9].

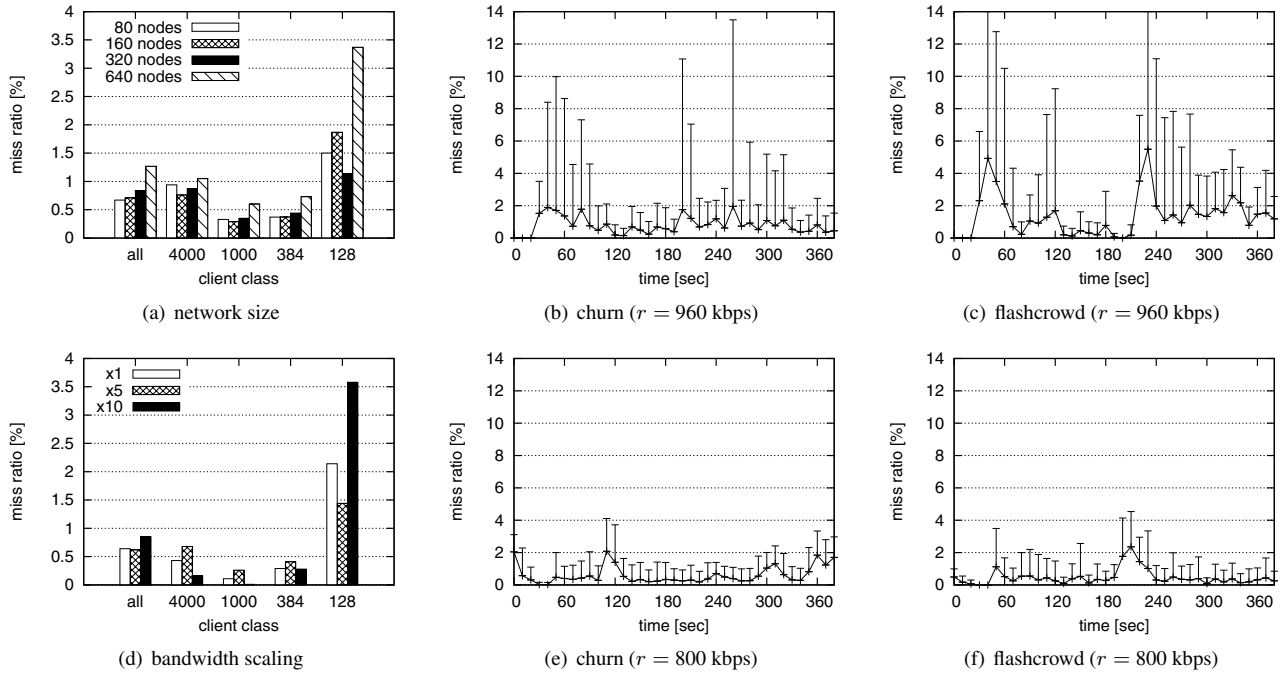
The figure also shows that there is a somewhat sharper increase in the miss ratio from 320 to 640 clients. This suggests that the average propagation delay for 640 nodes is getting close to the 30-second download window. Thus, for larger network sizes it may be necessary to increase the download window accordingly.

In a second experiment, we scale up all bandwidths in the system up to a factor of 10, i.e., uplink capacities as well as the source stream rate. This allows us to evaluate the performance of our design for streaming higher-definition content through faster clients links such as FTTH (fiber-to-the-tome). In this experiment we reduce the network size to 40 nodes to avoid congesting our Modelnet core (whose emulation precision degrades quickly beyond 600 Mbps of aggregate traffic). Figure 5(d) shows that the average miss ratio for a 4.8 Mbps stream (5x scaling) is comparable to that observed at 960 kbps. At 9.6 Mbps (10x scaling), the average miss ratio increases slightly but remains under 1%. This shows that our design also scales well with respect to network capacity.

5.5. Churn

Finally, we evaluate the performance of our system under churn. First, we simulate continuous, steady-state churn by killing and restarting a random node every 10 seconds.

Figure 5(b) shows that the average miss ratio always stays below 2%, confirming the robustness of our unstructured overlay architecture. The gap and high miss ratios



(a) network size (b) churn ($r = 960$ kbps) (c) flashcrowd ($r = 960$ kbps)

(d) bandwidth scaling (e) churn ($r = 800$ kbps) (f) flashcrowd ($r = 800$ kbps)

Figure 5. (a) Near 10-fold increase in size produces a minor performance drop. (b) (e) High robustness under continuous churn (1 join/leave every 10 seconds). (c) (f) Impact of flashcrowd at $t = 200$. (d) Bandwidths can be scaled with little or no performance drop (stream rate = 90% max).

observed in the first minute are due to initial node joins. Notice that some nodes temporarily experience high miss ratios throughout the experiment, as shown by the relatively high standard deviation values. These spikes are due to high bandwidth nodes disconnecting, which prevent some of their old neighbors from receiving enough data until they find a replacement neighbor.

Note that the system is working at 90% of its maximum rate, so there is very little extra bandwidth to compensate for node disconnections. Figure 5(e) shows the response to churn when the stream rate is lowered to 800 kbps. We observe that the standard deviation has dropped significantly. At this streaming rate nodes have some spare upload capacity, which is used to help clients that are no longer receiving chunks from disconnected nodes.

Finally, we evalute our system in a flashcrowd scenario. Flashcrowds occur when a massive number of users join the system concurrently. We set up an experiment in which 40 nodes simultaneously join an existing overlay of 40 nodes. Figures 5(c) and 5(f) show the impact of a flashcrowd occurring at $t = 200$. Although there is a noticeable increase in the average miss ratio during the flashcrowd, the average miss ratio stays below 6% for 960 kbps, and around 2% for 800 kbps.

5.6. Comparison to other designs

We now discuss our prototype's performance compared to that of other two recently evaluated systems, AQCS and GridMedia. The reader should keep in mind that we base our discussion on measurements from similar, but different studies. Therefore, the following is an indirect comparison and should only be considered as indicative.

First, we have shown that the average delivery rate of our prototype reaches 95% of the maximum achievable rate. This rate performance is comparable to that achieved by the AQCS algorithm in a recent experimental study [10], which used the same bandwidth distribution and stream rates as our evaluation. Compared to the GridMedia protocol, our designs seems to deliver higher rates: their system required overprovisioning bandwidth by 20% to keep the miss ratio under 1% in a 409-node experiment [13]. Our prototype achieves the same miss performance with only 10% overprovisioning (cf. Figure 5(a)).

Second, although we have shown that our prototype achieves reasonable diffusion delays, there seems to be a small delay penalty compared to tree and fully connected overlay designs. Using a 15-second window, AQCS showed a 0% miss ratio at 960 kbps, whereas we observed a loss of slightly over 1% for the same configuration. AQCS prob-

ably has a lower delay thanks to a maximum hop count of two. However, its fully connected overlay limits scalability to a few tenths of peers. The GridMedia study showed an average delay of only 2 seconds for their tree-based protocol. However, their evaluation was less stressful than ours: they overprovisioned total bandwidth by 20%, used churn traces with a longer average peer online time (1500-seconds), and employed a source over 6 times faster than the stream rate. In particular, a high source capacity allows multiple copies of a chunk to be pushed through several independent trees, significantly reducing the average delay. In fact, measurements not included in this paper show that increasing the source bandwidth from 1.1 to 6 Mbps allows us to reduce the download window from 30 to 6 seconds with no degradation of the delivery rate.

6. Conclusions and future work

Recent work on peer-to-peer streaming has focused on designing systems which maximize the achievable streaming rate. Moreover, recent experimental studies have shown that tree-based designs can reach close-to-optimal rates in real-life conditions. However, until now, efficient mesh-based designs have only been evaluated analytically or through simulations.

In this paper we have presented an experimental study which demonstrates that a pure mesh architecture can deliver near-optimal rates with low diffusion delays. In addition, we have identified some optimizations which help increase the efficiency of mesh-based designs. In particular, both analytical and experimental evidence suggests that giving preference to fast nodes can increase the performance of a system.

Given these encouraging results, we plan to extend our work and investigate a number of issues not covered in this paper. First, we plan to further evaluate the use of source coding (such as erasure codes or multiple description coding) to minimize the impact of chunk misses on image quality.

Second, we will look into mechanisms that determine whether the overlay has enough aggregate uplink capacity to deliver the stream with acceptable losses. An adaptive system could enable high-bandwidth helper nodes (independent from the source) when a capacity shortage is detected.

Finally, we will study how we can modify algorithms such as DP/LU to further increase their delay performance by favoring high capacity nodes not only at the source, but in all scheduling decisions.

References

[1] Susu Xie, Bo Li, Gabriel Y. Keung, and Xinyan Zhang. Coolstreaming: Design, Theory and Practice. In *IEEE Transactions on Multimedia*, Vol. 9, Issue 8, December 2007.

[2] PPLive, http://www.pplive.com

[3] PPStream, http://www.ppstream.com

[4] SopCast, http://www.sopcast.org

[5] TVants, http://tvants.en.softonic.com

[6] Joost, http://www.joost.com

[7] Gnutella. http://www.gnutella.com

[8] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proc. of OSDI*, 2002.

[9] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, A. Twigg. Epidemic Live Streaming: Optimal Performance Trade-Offs. In *Proc. of SIGMETRICS*, 2008.

[10] C. Liang, Y. Guo, and Y. Liu. Is Random Scheduling Sufficient in P2P Video Streaming? In *Proc. of ICDCS*, 2008.

[11] C. H. Ashwin R. Bharambe and V. N. Padmanabhan, Analyzing and Improving a BitTorrent Network Performance Mechanisms. In *Proc. of INFOCOM*, 2006.

[12] B. Cohen. Incentives Build Robustness in BitTorrent. In *Proc. of P2PECON*, 2003.

[13] M. Zhang, Q. Zhang, L. Sun, and S. Yang, Understanding the Power of Pull-based Streaming Protocol: Can We Do Better? In *IEEE JSAC, special issue on Advances in Peer-to-peer Streaming Systems*, 2007.

[14] R. Kumar, Y. Lin, and K. Ross. Stochastic fluid theory for P2P streaming systems. In *Proc. of INFOCOM*, 2007.

[15] Y. Guo, C. Liang, and Y. Liu. Adaptive Queue-based Chunk Scheduling for P2P Live Streaming. Technical Report, Polytechnic Univ., 2007.

[16] C. Liang, Y. Guo, and Y. Liu. Hierarchically Clustered P2P Streaming System. In *Proc. of IEEE Globecom*, Nov. 2007.

[17] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. DONet/CoolStreaming: A data-driven overlay network for live media streaming. In *Proc. of INFOCOM*, 2005.

[18] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez. Randomized decentralized broadcasting algorithms. In *Proc. of INFOCOM*, 2007.

[19] http://techland.blogs.fortune.cnn.com/2008/03/25/youtube-looks-for-the-money-clip/

[20] N. Magharei, R. Rejaie, and Y. Guo. Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In *Proc. of INFOCOM*, 2007.

[21] R. Blahut. Theory and Practice of Error Control Codes. Addison Wesley, MA, 1994

[22] M. Castro, P. Druschel, A.-M. Kermarrec, A. R. Nandi, and A. Singh. SplitStream: High-bandwidth content distribution in a cooperative environment. In *ACM SOSP*, 2003.

[23] V. Venkararaman, K. Yoshida, and P. Francis. Chunkspread: Heterogeneous unstructured end system multicast. In *Proc. of ICNP*, 2006.

[24] Y.-H. Chu, S. G.Rao, and H. Zhang. A case for end system multicast. In *Proc. of SIGMETRICS*, 2000.

[25] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *IPTPS*, 2005.

[26] F. Pianese, D. Perino, J. Keller, and E. Biersack. Pulse: an adaptative, incentive-based, unstructured p2p live streaming system. In *IEEE Transaction on Multimedia*, 2007.

[27] P. Francis. Yoid: Extending the Internet Multicast Architecture. http://www.icir.org/yoid.

[28] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. OToole, Jr. Overcast: Reliable multicasting with an overlay network. In *Proc. of OSDI*, 2000.

[29] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination using an Overlay Mesh. In *Proc. of SOSP*, 2003.

[30] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of IFIP/ACM Middleware*, 2001.

[31] C. Gkantsidis, J. Miller, P. Rodriguez. Comprehensive view of a Live Network Coding P2P system. In *Proc. of IMC*, 2006.

Appendix

In this section we derive a general lower bound for the maximum diffusion delay D with arbitrary uplink bandwidths.

Let $G(t)$ represent the largest number of packets that nodes can send by time t, assuming that at time 0 only the source node has any packet. Assume without loss of generality that the uplink bandwidths $u_1, \ldots, u_n$ are sorted in decreasing order: $u_1 \geq \ldots \geq u_n$.

The function G can be determined in an iterative manner, introducing the functions G_i which correspond to the same uplink bandwidths $u_s, u_1, \ldots, u_i$, but for which the uplink bandwidths u_j for $j > i$ have been set to zero. One then has the following relations:

$$\begin{aligned} G_0(t) &= \lfloor u_s t \rfloor, \\ G_i(t) &= G_{i-1}(t) + \lfloor \max(0, u_i t - \tau_i), i \geq 1 \rfloor, \end{aligned}$$

where $\tau_i = \inf\{t > 0 : G_{i-1}(t) \geq i\}$. Then $G(t) = G_n(t)$.

The proof of the lower bound (2) goes as follows. Assume there is a scheme that achieves successful delivery of all packets to all users with a delay no larger than $D_0 = D - \epsilon$ for some positive ϵ. Consider successive packets $p = 0, 1, \ldots, k$ generated at times $0, 1/r, \ldots, k/r$. By time D_0, by definition of G, no more than $G(D_0)$ copies of such packets have been disseminated. By assumption, n copies of packet 0 have been disseminated. Thus, at most $G(D_0) - n$ packets with labels 1 or higher have been spread. By definition of D, $G(D_0) - n < G(D_0 - 1/r)$. Let D_1 be the smallest $d > 0$ such that $G(d - 1/r) \geq G(D_0) - n$. By the previous inequality, $D_1 < D_0$.

Consider now the number of packets with label 1 or more present in the system by time $1/r + D_0$. Then by the previous argument and the definition of G, this is no larger than $G(D_1)$. Again, there are at most $G(D_1) - n$ such packets with label 2 or larger, which is smaller than $G(D_1 - 1/r)$. Iterating the argument, we find a decreasing sequence of delays $D_0 \geq D_1 \geq D_2 \ldots$ such that the number of packets with label i or larger present in the system by time $i/r + D_0$ is at most $G(D_i)$, with the inductive definition:

$$D_i = \inf\{d > 0 : G(d - 1/r) \geq G(D_{i-1}) - n\}.$$

Let D_∞ be the limit $\lim_{i \to \infty} D_i$. Then necessarily,

$$G(D_\infty) = \inf\{d > 0 : G(d - 1/r) \geq G(D_\infty) - n\}.$$

Thus for all positive δ, $G(D_\infty + \delta - 1/r) < G(D_\infty + \delta) - n$. Recalling the definition of D, it therefore follows that $D_\infty \geq D$. This is however a contradiction, in view of the condition $D > D_0 > D_\infty$.

Here are a couple of remarks regarding this lower bound on worst case delay.

- In the homogeneous case $u_i \equiv u_s = u$, and assuming $n = 2^I$, one has

  $$G(t) = \begin{cases} 2^{\lfloor ut \rfloor} \text{ if } t \leq u^{-1} I, \\ n + (n+1)(ut - I) \text{ otherwise.} \end{cases}$$

  Thus under the rate feasibility condition $(n+1)u \geq nr$, it can be readily checked that the bound provided by the proposition lies in the interval $[\log_2(n), \log_2(n) + 1]$.

- At this stage we do not know if the bound is tight for arbitrary heterogeneous profiles.

- A more involved argument, relying on Little's formula from queueing theory, can be made to show that for any scheme, the *average* delay $\overline{D}$ is lower-bounded as follows:

  $$\overline{D} \geq D - \int_{D-1/r}^{D} G(s) ds.$$