Contracts: Practical Contribution Incentives for P2P Live Streaming

Michael Piatek* Arvind Krishnamurthy* Arun Venkataramani†
Richard Yang◇ David Zhang× Alexander Jaffe*

Abstract

PPLive is a popular P2P video system used daily by millions of people worldwide. Achieving this level of scalability depends on users making contributions to the system, but currently, these contributions are neither verified nor rewarded. In this paper, we describe the design and implementation of *Contracts*, a new, practical approach to providing contribution incentives in P2P live streaming systems. Using measurements of tens of thousands of PPLive users, we show that widely-used bilateral incentive strategies cannot be effectively applied to the live streaming environment. *Contracts* adopts a different approach: rewarding globally effective contribution with improved robustness. Using a modified PPLive client, we show that *Contracts* both improves performance and strengthens contribution incentives. For example, in our experiments, the fraction of PPLive clients using *Contracts* experiencing loss-free playback is more than 4 times that of native PPLive.

1 Introduction

System collapse due to large-scale reductions in user contributions is a major concern for PPLive, which is one of the most widely deployed live streaming services on the Internet today, serving more than 20 million active users spread across the globe. Using peer-to-peer (P2P) as the core technique, PPLive achieves cost-effective live video distribution by providing a small amount of seed bandwidth to a few participants, with the rest of the distribution being performed by users relaying data. Thus, the availability and scalability of PPLive depends crucially on the contributions of its users.

The current PPLive design neither verifies nor rewards contributions, creating the potential for strategic users to restrict their contribution, degrading robustness. This is particularly true in environments where capacity is limited or priced by usage. Furthermore, when developing an open live video streaming standard, relying on closed systems with proprietary protocols is not feasible.

In this paper, we explore how to provide practical contribution incentives for P2P live streaming, using PPLive as a concrete example. Although incentives have been studied extensively in the case of widely deployed file-sharing systems (e.g., [1, 16, 22]), live streaming presents unique challenges. For instance, clients cannot be rewarded with faster downloads once they are receiving data at the broadcast rate (since additional data has not yet been produced). While some recent proposals have considered contribution incentives in a live streaming setting (e.g., [17]), they do not take into account several practical considerations of deployed systems, such as client heterogeneity and operation under bandwidth constraints. We provide an examination of live streaming incentives grounded in experience with a deployed and widely used live streaming system.

We proceed in two steps. First, we use measurements of tens of thousands of PPLive clients to demonstrate quantitatively the challenges in adapting existing incentive strategies to the live streaming environment. We find that in practice, the majority of system capacity is contributed by a minority of high capacity users. As a result, incentive mechanisms that require balance between consumption and contribution will either exclude many users from participation or underutilize capacity substantially.

More broadly, bilateral exchange mechanisms widely used in bulk data distribution, such as tit-for-tat in BitTorrent [5], are ineffective in the live streaming environment, in part because sequential block availability sharply limits trading opportunities between peers. Imposing topologies that increase bilateral trading opportunities (e.g., [17]) increases the variance in block delivery delay, causing either increased playback deadline misses or increased startup delay. Tit-for-tat, for example, substantially reduces performance when applied to PPLive.

The second part of the paper describes *Contracts*, a new design for providing robust contribution incentives in live streaming P2P services. *Contracts* differs from existing techniques in two principal ways. First, *Contracts* is designed for the live streaming environment. Rather than relying on increased download rates, *Contracts* rewards contributions with increased quality of service when the system is constrained. Second, *Contracts* departs from traditional incentive mechanisms that rigidly constrain client behavior. Instead, we define a default *contract* specifying an agreement between an individual client and the overall system (i.e., PPLive and other clients) as to how its contributions will be evaluated by others. To enable this, we introduce a lightweight protocol that provides verifiable accounting of each client's contributions. But, the contract does not mandate fine-grained behavior, leaving individual clients free to make local optimizations that increase efficiency.

We have integrated *Contracts* with PPLive, and find that our implementation both improves performance and strengthens contribution incentives. For example, in our

*U. of Washington; † U. of Mass; ◇ Yale; × PPLive.

experiments, the fraction of PPLive/*Contracts* clients experiencing loss-free playback is more than 4 times that of native PPLive, and clients that contribute more than others receive consistently higher quality of service.

The remainder of this paper is organized as follows. Section 2 provides an overview of live streaming in PPLive. Sections 3 and 4 describe the challenges of applying incentive strategies based on bilateral exchange to live streaming. These challenges motivate the design and implementation of *Contracts*, which we present in Section 5 and evaluate in Section 6. We discuss related work in Section 7 and conclude in Section 8.

2 PPlive overview

PPLive is a hybrid P2P system for streaming live and on-demand video. Clients are organized into channels, with members of a given channel redistributing video data to one another. Clients rely on two forms of infrastructural support: 1) coordinating *trackers* that provide a rendezvous point for users watching the same channel, and 2) seed bandwidth provided by a group of broadcast servers that source all content. Multiple channels can be managed by a single tracker and sourced by a single broadcaster. Currently, PPLive maintains roughly 600 public live channels daily.

The wire-level details of the PPLive protocol are similar to existing swarming systems like BitTorrent [5]. Clients maintain a large set of directly connected peers (50–100) to which they advertise their local data blocks and issue requests for missing blocks. Each block is 4–16 KB and is discarded shortly after being played.

Trackers maintain state that includes the set of clients in each channel, the properties of clients (e.g., reported bandwidth capacity and NAT status), and the overall health of channels. The health of an individual channel is monitored by its broadcast source. Clients use the source as a peer of last resort. Only when a block cannot be obtained from any other peer is a request sent to the data source. Thus, the load on the broadcaster provides a metric for the health of a given channel relative to others. By shifting capacity from channels demanding less load to those servicing more requests, PPLive allocates infrastructure bandwidth automatically.

Most relevant to our work is PPLive's servicing policy. By default, each client contributes its full available capacity and does not prioritize service for particular peers, i.e., download requests from a peer contributing little to the system and those from a peer making the highest contribution are treated equally.

3 Limits of bilateral exchange

The lack of contribution incentives means that PPLive's scalability depends on clients' good will and the faithful execution of its software. At present, these are largely effective due to flat-rate network pricing and the complexity of PPLive's proprietary implementation. Increasingly, however, the all-you-can-eat pricing model is giving way to the realities of network management [6]. Furthermore, there is continued interest in developing an open live video streaming standard supporting multiple implementations (e.g., IETF 73 PPSP BoF). These trends motivate the explicit consideration of incentives for live streaming systems to reward good (and discourage bad) behavior by coupling performance and contribution.

Why does live streaming necessitate revisiting the incentive design problem? To appreciate this, consider the most widely-used class of incentive strategies in P2P systems today—bilateral exchange—wherein a peer x determines the amount of upload bandwidth to peer y based solely on the amount of bandwidth that client y uploads to x, independent of the total bandwidth that client y uploads to all clients. Servicing policies based on bilateral exchange are compellingly simple. For example, tit-for-tat has been widely applied in bulk data distribution systems (e.g., BitTorrent [5]), and has also been studied extensively [16, 25]. More recently, bilateral exchange has also been proposed as a basis for providing incentives in live streaming systems (e.g., [17]). However, bilateral exchange schemes suffer from fundamental performance limitations in the context of live streaming.

For bilateral exchange to work, peers need to have trading opportunities (i.e., distinct data blocks of mutual interest). When distributing bulk data, trading opportunities are frequent. Each client seeks to acquire the entirety of a large set of blocks. Bulk distribution systems typically use block selection strategies such as local rarest first (e.g., BitTorrent [5]) or network coding of blocks (e.g., Avalanche [9]) to ensure that all blocks having roughly equal trading value over time. Ideally, once a new client has received just a few random blocks, it is bootstrapped into the trading system.

Live streaming differs radically from bulk data distribution in ways that significantly reduce the effectiveness of bilateral exchange. We consider four key challenges in live streaming that inform the design of *Contracts*.

1) Heterogeneity: Capacity heterogeneity poses a fundamental challenge to the efficiency of balanced exchange schemes. Live streaming offers a common download rate—the stream playback rate—to all peers regardless of their upload capacity. In practice however, peer capacities can vary by an order of magnitude. We verify this by measuring the capacity distribution of PPLive clients using logs of the reported bandwidth capacity of 99,184 clients. The distribution is highly skewed with a mean capacity (142 KBps) that is more than double the median (65 KBps). As a result of the skew, the majority of total aggregate capacity is provided by a minority of high capacity peers. The top 10% of clients account for

58% of total capacity.

Capacity heterogeneity implies a discouraging tradeoff between efficiency and balance. Insisting on near-perfect balance will either exclude many users that cannot support the stream rate or significantly underutilize capacity. Concretely, streaming at the average capacity of 142 KBps (the maximum possible) would would exclude 86% of PPLive clients in our trace when requiring balanced contribution and consumption. On the other hand, providing service to 95% of PPLive clients (with balanced exchanges) requires restricting the stream data rate to the 5th percentile of capacity at 21 KBps, which corresponds to an overall utilization of just 15%.

The fundamental tradeoff between efficiency and balance under skew can be quantified as follows. Let μ and σ respectively denote the mean and variance of the upload capacity distribution. We define the *skew*[1] S as σ/μ. For a stream rate of r, the *efficiency* E is r/μ, where a feasible broadcast implies $\mu \geq r$. We define the *imbalance* I as the deviation of peer upload rates with respect to the stream rate normalized by the mean, i.e., $\frac{1}{\mu}[\frac{\sum_i (x_i - r)^2}{N}]^{1/2}$, where x_i is peer i's upload rate (less than or equal to its capacity) and the sum indexes over all N peers. Note that all three of skew, efficiency, and balance lie between 0 and 1. The theorem below captures the stated tradeoff, the proof of which is available in a technical report [24].

THEOREM **1** *High efficiency and high skew imply high imbalance. Specifically,* **(a)** *If peers upload at a rate proportional to their capacity,* $I = E \cdot S$. **(b)** *For any feasible set of upload rates, I is bounded from below by a function that monotonically increases from* 0 *to S as E increases from* 0 *to* 1.

2) Limited bandwidth needs: In bulk data distribution using bilateral exchange, the incentive to increase upload rate is a corresponding increase in download rate. In live streaming, however, once a client is downloading data at the rate of production, a further increase in download rate is not possible (as additional data does not yet exist). Although one may consider rewarding increased contribution with improved video quality (e.g., at higher resolutions using layered coding), PPLive avoids such a scheme due to its increased complexity, reduced video coding efficiency, and the need for substantially higher bandwidth to produce visually-discernible quality differences. Thus, a challenge to incentivize users to contribute capacity in excess of their demands is to create a compelling reward with nonzero marginal utility.

The above points do not rule out bilateral exchange schemes that are not balanced, which we consider below.

3) Limited trading opportunities: Bilateral exchange depends on the existence of mutually beneficial trading opportunities to evaluate peers. Unfortunately, live streaming provides clients with limited opportunities for mutually beneficial trading. The key difference is that unlike bulk data distribution, where blocks have roughly equal value over time and among clients, *the value of blocks in live streaming varies over time and client*. A block has little value at a client if it is received after the playback point at the client. Thus, the data useful to an individual client is limited to a narrow range between the production point and the local playback point, i.e., the *lag*. The smaller the lag that a system targets, the fewer the trading opportunities. Furthermore, blocks in live streaming emerge at the data source one at a time at the production rate unlike bulk data distribution where data becomes available all at once. As a result, clients closer to the data source in the topology have inherent advantages in receiving rare (new) blocks first, creating a perpetual trade imbalance with clients further from the source. Although trade imbalance does not necessarily rule out bilateral exchange schemes, it makes evaluating peers significantly more challenging.

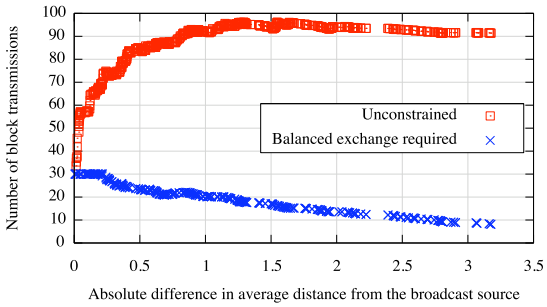

Figure 1: The impact of distance from the broadcast source on bilateral exchange. Requiring balanced exchange significantly limits trading opportunities as does distance from the source.

To make this concrete, we compare the number of trading opportunities for clients in a PPLive broadcast with 100 clients running on the Emulab testbed. Each client uses random block selection to maximize trading opportunities unless a block is near its playback deadline. Each client simultaneously joins a test stream and periodically logs its buffer state during playback. Figure 1 summarizes the trading opportunities among pairs of peers taken from a snapshot of buffer states collected several minutes into the broadcast. Each individual client's average distance to the broadcast source is the average number of overlay hops traversed by all of its received blocks, which correlates with lag. The number of trading opportunities is shown in terms of the absolute difference in these averages for pairs of peers (x-axis) without constraints and when a balanced number of mutually beneficial trades is required. These results show that the greatest opportunity for bilateral exchange is between peers that are at a similar distance from the broadcast source. But, such pairs of peers are in the minority. Most

[1] unlike the more standard definition based on the third moment.

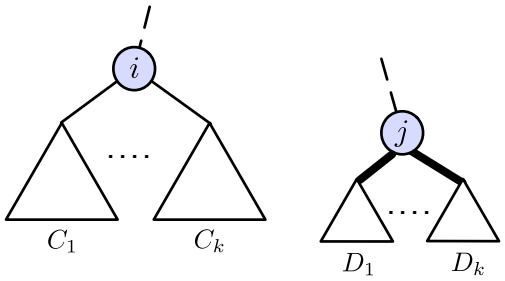

Figure 2: Illustration for Theorem 2: Node j has higher upload capacity than node i but has fewer descendants.

block transfer opportunities are between pairs of peers that have an imbalance of useful data.

4) Delay sensitivity: Live streaming requires low distribution delay so as to reduce lag (i.e., improve liveness) or, equivalently, reduce the playback miss rate for a given lag. Minimizing block dissemination delays imposes some structural requirements on the steady-state block distribution topology. We formalize this claim as follows. The dissemination topology traversed by a single block must be a tree as peers only request blocks they do not already have. Consider the dissemination tree T for a single block. Note that different blocks may have different dissemination trees, so a node may be at different distances from the source across blocks. We assume that in steady-state, the system can sustain the stream rate such that a block is never queued at a node behind another block.[2]

THEOREM **2** *Any topology in which a peer i has lower bandwidth than peer j but i has more descendants than j has higher average block delay than the topology obtained by swapping i and j if one of the following two conditions hold: (a) the topology is a balanced tree, or (b) i is an ancestor of j.*

Figure 2 illustrates the condition in the theorem above. The proof shows that, if either T is balanced or i is an ancestor of j, T can be transformed to a topology T' with lower delay by simply swapping i and j.

The structural requirement for low delays presents a design conflict for bilateral exchange schemes. Being closer to the source, a high capacity peer A is likely to receive newer blocks before a lower capacity child B, so A is unlikely to benefit from B. However, in bilateral exchange, A evaluates B solely by B's uploads to A, forcing B to try to upload to A even though that is detrimental to the average block delay. Note that bulk distribution does not face this predicament as it does not require individual block delays to be low, a crucial consideration in live streaming.

[2]The technical report [24] describes a procedure to construct such a dissemination tree packing.

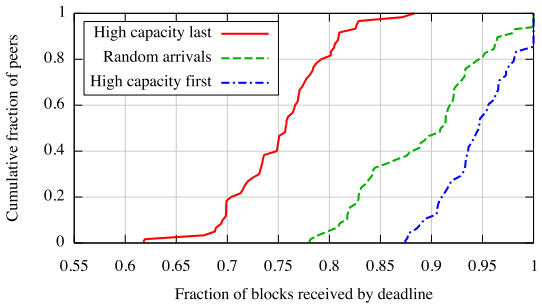

Figure 3: Cumulative fraction of clients with a given block delivery rate for different topologies. Placing high capacity clients near the source improves quality.

4 Structuring for performance and incentives

The limitations of bilateral exchange lead us to pursue a fundamentally different approach to providing incentives in P2P live streaming systems. Instead of rewarding higher upload rates with higher download rates, we craft a mechanism that incentivizes higher upload rates with robust playback quality; i.e., fewer missed playback deadlines, despite churn and capacity constraints. As observed in measurement studies [20], in a streaming system at a given channel rate, robust playback quality is the key determiner of user satisfaction. Our design of the incentive mechanism is enabled by a pleasant coincidence that aligns performance and incentive objectives: high capacity peers must be close to the source to keep block delays low, and peers closer to the source experience lower and more predictable block delays yielding better playback quality.

Topology: To maximize utilization, high capacity clients need to be placed near the data source so that they can quickly replicate useful data. To demonstrate the impact of topology and capacity heterogeneity on playback quality, we compare the block delivery rate for 120 instrumented PPLive clients running on Emulab under three scenarios: 1) clients joining in a random order, 2) high capacity clients joining first, followed by low capacity clients, and 3) low capacity clients preceding high capacity. In each scenario, the over-provisioning of capacity relative to demand is two. 50% of clients are assigned an upload capacity equal to the stream data rate (low capacity) with the remaining 50% having capacity $3\times$ the stream rate (high capacity). The results are summarized in Figure 3. Playback quality is best when high capacity peers join first, and are therefore closer to the data source. When high capacity peers join last, the quality degrades significantly. With no change in total system capacity, the median delivery rate drops from 0.95 to 0.75. In practice, the order in which clients join is likely to be random with respect to capacity, yielding playback quality in between the two extremes.

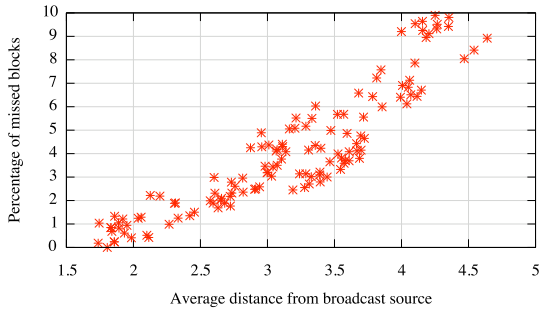

Figure 4: The fraction of blocks missing playback deadlines as a function of distance from the broadcast source. Playback quality is best for clients nearest to the source.

Buffering: When blocks miss playback deadlines, PPLive takes one of two actions: 1) if only a few blocks are missing, they are skipped; 2) but, if several blocks miss their playback deadline, PPLive pauses while waiting for downloads to complete. This buffering policy is designed to handle temporary degradation in quality of service. A single missed block has limited impact on video quality, and rebuffering suffices to recover from more significant fluctuations. But, if a client chronically experiences misses, it will eventually fall so far behind its directly connected peers that required blocks are no longer available. In this case, users need to manually rejoin the broadcast. Restarting is a simple recovery mechanism and requiring it is an explicit design choice in PPLive that is consistent with typical user behavior.

The buffering policy implies that the effects of quality degradation cascade. When a client near the data source stalls, more distant clients to which it forwards data also experience service disruption. Although the PPLive mesh contains significant path redundancy, failover is not instantaneous and may require rebuffering. We quantify the quality of service in terms of distance from the broadcast source in Figure 4. In this experiment, 127 clients with equal capacities (twice the stream rate) participate in a broadcast with one client joining every 10 seconds. Statistics are computed after all clients have been in the system for at least 20 minutes. As in previous experiments, we define the average distance of a client from the source to be the average number of hops traversed by all blocks received by the client. The results show that service quality degrades with distance from the source even in a static setting. Introducing churn will further degrade service quality for clients that are further away from the source.

5 Contracts

We now describe *Contracts*, a new scheme to provide contribution incentives in P2P live streaming systems. Our scheme is based on two key design choices that are motivated by the considerations unique to live streaming.

Contracts rather than bilateral reciprocation: Recognizing that bilateral exchanges are ineffective for live streaming, we develop a scheme that rewards each peer according to its global effectiveness. We borrow from economic theory, in particular the *principal-agent problem*, the idea of *contracts*—a method of structuring incentives in asymmetric or non-bilateral settings [15]. In *Contracts*, a data provider grants a level of service proportional to the consumer's ability to replicate the data further, as opposed to basing service simply on reciprocal contributions. The contract is thus designed to motivate consumers to contribute their bandwidth and also to hold them accountable for their respective servicing decisions. Further, since a provider is also a consumer in a P2P setting, it should be in the provider's self-interest to enforce the contract to obtain good service from its own providers. Put simply, a node's incentives as a provider should be aligned with its incentives as a consumer.

Global topology optimization: Instead of operating with an unstructured mesh, *Contracts* structures the overlay topology globally to account for the heterogeneity of peer capacities. Specifically, we introduce mechanisms that allow clients to identify their positions relative to the stream source and reorganize themselves, with high capacity peers percolating towards the source. Peers with disparate capacities develop asymmetric yet mutually beneficial relationships: low capacity peers benefit from the replication capabilities of high capacity peers, while high capacity peers are rewarded with better quality of service for their contributions.

In this section, we outline the following: (1) the single global contract for evaluating both the quantity and quality of each peer's contributions, (2) a default policy for updating the overlay topology, and (3) a wire-level accounting protocol for verifying contributions.

5.1 Contribution contracts

In *Contracts*, each peer is evaluated for both 1) the amount that it contributes to directly connected peers, and 2) the amounts those peers contribute in turn; i.e., the quality of its selections. Peers with higher valuations will have a greater likelihood of being added to the peer lists of high capacity nodes and also enjoy prompt service when requesting individual blocks from those nodes. We next describe the details of how peer evaluations are computed.

Performance metrics: For two peers x and y, we denote the contribution rate from x to y by $B(x \rightarrow y)$, and compute this using a weighted moving average. $B(x)$ represents the total bandwidth contributed by node x. Each of these values is mapped to discrete classes of contribution—the deciles of the observed capacity distribution from PPlive. We label the bandwidth class of a node x as $BC(x)$.

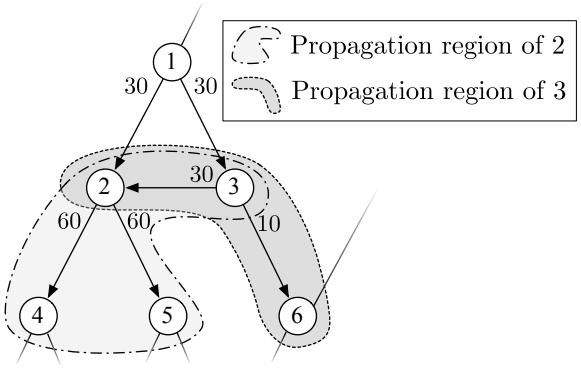

Figure 5: Evaluating I for client 1. Contribution from $1 \rightarrow 2$ is weighted by the rates from $2 \rightarrow 3, 4, 5$. Contribution of $1 \rightarrow 3$ is weighted by the rates from $3 \rightarrow 2, 6$.

To measure the effectiveness of contributions made by a client, we define $I(x)$ to be the one-hop propagation of x's contributions, calculating this as follows:

$$I(x) = \sum_{p \in \text{peers}(x)} B(x \rightarrow p) \times D_{\text{BW}}(BC(p)) \qquad (1)$$

where $D_{\text{BW}} \in [0,1]$ is a weight specified by the cumulative distribution function of peer upload capacities.

As an example computation of I, consider Figure 5. In this case, the effectiveness of contributions from node 1 are being evaluated. The total contribution rates of peers 2 and 3 are 120 and 40, respectively. Mapping the values 120 and 40 to their bandwidth classes and looking them up in our measured capacity distribution yields: $D_{\text{BW}}(BC(3)) = 0.1$, $D_{\text{BW}}(BC(2)) = 0.8$. Substituting these values allows us to compute $I(1) = 30 \times 0.1 + 30 \times 0.8 = 27$.

Taken together, measures of contribution (BC) and effectiveness (I) constitute the global evaluation function, $V(x)$, which we define as the tuple $[BC(x), I(x)]$ with the following comparison operator:

$$V(x) > V(y) \iff BC(x) > BC(y) \text{ or } BC(x) = BC(y) \wedge I(x) > I(y).$$

In other words, peers are compared by their bandwidth class first, and peers within a class are compared according to the effectiveness of their contributions.

Servicing policy: The metrics defined above are used by clients to identify which peers are selected to receive service, the priority of that service, and which potential peers to prefer. We distinguish between *connection* and *selection* in our discussion of service policy. Connection is a prerequisite for being selected to receive service, and only connected peers exchange the control traffic necessary to compute $V(\cdot)$.

- *Peer selection:* Each node periodically rank-orders its peers by their corresponding $V(\cdot)$ values, selects the top k of these, where k is a configurable parameter, and notifies each that block requests will be accepted.
- *Block request servicing:* Among peers with outstanding requests, each client prioritizes the request from the peer with the maximum $B(\cdot)$ value.

In Section 5.3, we describe how each client reliably ascertains the performance metrics of its peers. Before doing so, we first analyze the incentive structure arising from this servicing policy.

What are the incentives provided by the system? Contracts provides strong contribution incentives by linking quality of service to effective contribution. A peer increases its chances of being *selected* for service and its priority by increasing its upload contribution. It might appear that contribution incentives are weakened by the use of bandwidth classes, as a peer p can lower its contribution while still remaining in its class. However, doing so reduces its service priority for block requests, which is determined by $B(p)$ among peers in its bandwidth class.

Contracts also rewards peers for making globally beneficial contributions. A peer that transmits blocks to higher capacity peers will achieve a higher evaluation under $I(\cdot)$ (and hence $V(\cdot)$), increasing the likelihood of being *selected* by others.

Will a provider enforce contracts? One possible deviation is for the provider p to ignore the rank ordering of $V(\cdot)$ values when choosing peers. In this case, a deviating provider *selects* a node y rather than x even though $V(x) > V(y)$. This ordering implies either $BC(x) > BC(y)$ or $BC(x) = BC(y) \wedge I(x) > I(y)$. In the former case, the provider's deviation lowers its own $I(\cdot)$ value since its contributions to y will be weighted less than its contributions to x. Hence, the deviation is not in its self-interest. In the latter case, $I(p)$ is unchanged by enforcing the contract, and hence p does not benefit from deviating.

Another possible deviation is for the provider to not provide prioritized service to higher capacity peers. For instance, a provider could transmit a block to y instead of x even though $B(x) > B(y)$. Again, this is not in the provider's self-interest since it reduces the provider's evaluation under $I(\cdot)$.

Why not other incentive structures? Initially, our definition of $V(\cdot)$ may seem somewhat arbitrary. Why not make effectiveness (I) fully recursive? That is, by including the contributions of peers, peers of peers, and so on. And, why use bandwidth classes rather than bandwidth itself? Or, why not use bandwidth only and ignore effectiveness? We tackle each of these questions in turn to provide additional intuition behind the development of our global contract.

We avoid a fully recursive definition of effectiveness for scalability. Accounting for the propagation of contri-

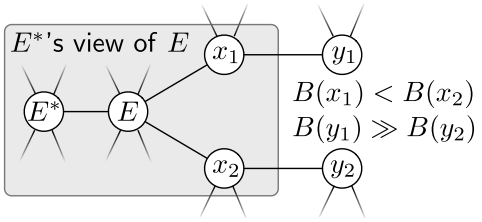

Figure 6: Under an alternate evaluation function $V'(\cdot)$, E has an incentive to unilaterally deviate when $B(x_2) > B(x_1)$ despite $V'(x_1) < V'(x_2)$.

butions globally creates significant overhead, both computationally and from increased control traffic. To reduce overhead, we limit propagation to one hop, with nodes percolating to their globally appropriate position through repeated cycles of evaluation and topology updates.

Unfortunately, limiting the propagation of accounting information creates an incentive to ignore the effectiveness of a peer's contributions, a crucial consideration when structuring the topology. As an example, consider a simpler evaluation function that uses bandwidth directly rather than bandwidth classes: $V'(x) = \sum_{p \in \text{peers}}(B(x \rightarrow p) \times D_{BW}(B(p))$. Under this evaluation function, a strategic client has an incentive to deviate. Consider the topology shown in Figure 6. In this case, the system would benefit from E contributing to x_1 since y_1 has much higher capacity than y_2, thus $V'(x_1) > V'(x_2)$. But, a client E^* evaluating E considers only the effectiveness of E, which is determined by the bandwidths of its peers only. Thus, when $B(x_2) > B(x_1)$, a rational E would contribute to x_2 rather than x_1, despite the greater redistribution capacity of x_1. This problem arises for any evaluation function with a limited view.

Bandwidth classes mitigate this problem by making peers with similar capacities incomparable at evaluators. When using V rather than V' to evaluate peers in Figure 6, E^* treats contributions to x_1 and x_2 equally (if they are in the same bandwidth class), eliminating the incentive for E to deviate. Our assumption is that clients are rational, but not Byzantine. Since deviating does not offer a local benefit, clients will split ties using contribution effectiveness, improving overall efficiency. We consider colluding and Byzantine peers in Section 5.4.

Finally, we incorporate effectiveness into our default contract rather than bandwidth alone in order to provide incentive-aware gossip, the topic we describe next.

5.2 Topology updating policy

Incentive-aware gossip: In addition to specifying how clients should make local servicing decisions, the contract also influences how the overlay topology is to be updated globally. To achieve a distribution structure where high capacity peers are closer to the source, *Contracts* uses peer gossip informed by $V(\cdot)$ as well as structural information provided by a hop count field in block messages. By maintaining the average hop count of blocks received from peers, each client can compare the average distance of its peers to the source, and we use this information to speed convergence when evaluating potential peers for *connection*.

Each client is aware of the capacities of its one-hop neighborhood of peers, and each client attempts to *connect* to the peers in this set with highest capacities. Universally applied, this results in the highest capacity peers percolating to the source, and lower capacity nodes being pushed to the periphery of the mesh through attrition.

Specifically, each client c sorts $p \in (\text{Peers} \circ \text{Peers})(c)$ by $BC(p)$, *connecting* to new peers in descending order. To split ties within a bandwidth class, c orders each potential peer by its average block hop count; i.e., a measure of the distance to the source, preferring the most distant of these. The intuition behind this policy is that the most distant peers within a bandwidth class are likely to be poorly clustered with respect to capacity and thus more likely to have outstanding block requests.[3] Preferential *connection* with misplaced peers in a bandwidth class speeds convergence of the topology. On the receiver's side, clients accept incoming *connections* optimistically, pruning those that have neither provided data nor warranted service in the recent past. Recall that connectivity does not imply that a peer will be *selected* for service. Exploring new nodes serves to expand the set of peers for which a given client can compute $V(\cdot)$.

This gossip strategy is *incentive-aware*; it incorporates the interests of clients seeking to maximize their V-value by contributing to the highest capacity peers. Consider the example topology in Figure 6. To compute $V(\cdot)$ for each of its peers, node E knows the bandwidth capacities of all labeled nodes (provided by our account mechanism). Suppose $BC(y_1) > BC(x_1)$. In this case, E would increase its V-value by sending to y_1 directly rather than through x_1, and so *connects* to y_1. Although x_1 might prefer to avoid this, revealing the bandwidth capacity of y_1 to E is required to demonstrate the effectiveness of its own contributions.

Bootstrapping new clients: For a newly joined, high capacity client to demonstrate its capability, it needs to receive stream data early enough to replicate that data widely. But, since a recently added client is typically placed far from the data source, the client might be overlooked simply because it could not receive enough useful data to replicate.

To address this problem, *Contracts* clients advertise an optional *bootstrapping block* comprised of random data. Advertising the bootstrapping block serves to inform directly connected peers that a client has excess capacity

[3] Preferring distant peers is a heuristic to increase trading opportunities. Recall that in a mesh with a capacity surplus, clients must compete to satisfy requests.

that can be verified through direct transfer. Each transferred bootstrapping block is worth half that of a normal block in terms of contribution value, although this value need not be precise. In light of the significantly skewed bandwidth distribution, our goal is to encourage meaningful contribution whenever possible while still allowing high capacity peers to demonstrate their capabilities.

Contracts's use of bootstrapping blocks exploits a key characteristic of the P2P environment, bandwidth asymmetry, to make a tradeoff beneficial to the overlay. Because many home broadband connections have significantly greater download capacity than upload capacity, identifying high capacity clients by downloading random data trades a reduction in abundant download capacity for an increase in upload capacity, the scarce resource. Of course, if a live broadcast has significantly more capacity than demand, bootstrapping blocks need not be transferred. To limit overhead, a *Contracts* client requests bootstrapping blocks only when it has excess capacity and with a probability that decreases as the number of its peers with excess capacity increases. This probability is given by: $1 - \frac{\text{Peers with excess capacity}}{\text{Total peers}}$.

5.3 Verifying contributions

The preceding topology update policies depend on the peers and the tracker obtaining truthful values of the global contributions of the peers (e.g., the calculation of Equation (1)). *Contracts* introduces *verifiable contributions* to support this task.

Each client P using *Contracts* completes a one-time registration step with the streaming infrastructure during which it is provided with a unique public/private key pair K_P. It should be clear in the context whether K_P represents the public key or the private key. The key pair serves as the client's identity and is persistent. Afterwards, clients are provided with two additional pieces of information when connecting to a channel: 1) a peer-specific nonce value and 2) the public key of the tracker of the channel. The nonce is used by several *Contracts* protocol messages to prevent replay attacks, and the tracker's public key is used to authenticate messages from the tracker that are forwarded in the overlay. We use $\langle M \rangle_K$ to denote a message M signed by key K.

When *Contracts* clients connect to one another, they exchange their respective public keys and nonce values.[4] Afterwards, data is exchanged normally. Periodically during data transfer, each *Contracts* client mints a signed receipt message for each of its peers. Each receipt accounts for the most recent contributions of that client and is sent to the remote endpoint. For example, if a client P with key pair K_P has received V blocks of data from a peer Q since it last sent a receipt to Q, P sends Q a receipt containing $\langle N_Q, K_P \rightarrow K_Q : V \rangle_{K_P}$. This includes a nonce (N_Q), the sender and receiver identities, and the number of blocks, signed by P.[5] Receipts are sent when a threshold on the volume of sent data is reached. This threshold is set by the tracker to control load and overhead.

Receipts serve as the foundation for verified contributions in *Contracts*, and we describe both distributed and centralized methods for using them to evolve the overlay topology. Distributed verification reduces load on the tracker, increasing scalability. Centralized verification speeds topology updates and reduces total network overhead, while also precluding several attacks from Byzantine users. These methods are not mutually exclusive; either (or both) can be used during a broadcast, and clients can switch between them freely depending on the level of contention for tracker resources. We describe each in turn.

Distributed verification: When using distributed verification, the tracker bootstraps new peers by providing a random subset of candidates to each client, and timestamps are used as nonce values. Each client forwards all receipts it receives due to contributions to its directly connected peers, including receipts collected from its one-hop neighborhood. Unlike the tracker, which generates the keys for all valid users in the broadcast, ordinary peers that receive receipts cannot distinguish valid identities from those generated by a strategic user. If any receipt is accepted, such users can manufacture an arbitrary number of receipts and claim any level of contribution. Thus, the challenge for verification in the distributed case is identifying valid receipts.

To do this, the tracker issues each user a small valid user message when the user first joins a given broadcast. This message is signed by the tracker and includes a timestamp, channel identifier, and the public key of the recipient. When peers connect to one another, they exchange and verify valid user messages, demonstrating to one another that they are a valid peer for the given broadcast.

Centralized verification: Although conceptually straightforward, distributed verification and contract enforcement assumes rational clients. A large number of Byzantine clients may undercut the convergence of our topology structuring algorithm, degrading performance. To address this, *Contracts* supports evaluating peers and enforcing topology updates at the tracker if necessary. Centralized topology updates also enable rapid and/or fine-grained adjustments to the topology during challenging workloads, e.g., flash crowds. The primary challenge to centralizing these functions is ensuring that the tracker is not overwhelmed with network traffic

[4] Man-in-the-middle attacks can be precluded by bundling peer keys with the peer list returned by the tracker. This increases overhead and is optional.

[5] For brevity, we omit the broadcast identifier, also included.

Computing the digest of receipts at peer Q

1: `Digest` $\leftarrow \emptyset$
2: `Hash` $\leftarrow 0$
3: Sort receipts by sender key
4: **for each** receipt $R : \{N, K_P \rightarrow K_Q : V\}$
5: from client P to Q with block count V; **do**
6: Increment counter for K_P in `Digest`
7: `Hash` $\leftarrow$ SHA-1(`Hash` $*$ SHA-1(R))
8: **done**
9: Send {`Digest`, `Hash`} to coordinator

Figure 7: Construction of the receipt digest message at a client Q. The $*$ operator indicates concatenation.

or computational demands, and the remainder of this section describes the mechanisms *Contracts* uses to achieve this.

Periodically, each client contacts the tracker to report its continuing participation in the broadcast and requests an updated set of peers. In the current PPLive implementation, this message also includes the client's maximum upload rate as measured by the client. *Contracts* piggybacks on this message, replacing the self-reported upload rate with a verifiable accounting of blocks contributed to specific peers during the previous update interval. Since public keys (and hence individual receipts) are lengthy, the naïve approach of simply forwarding all receipts to the tracker would amount to a de-facto DDoS attack. Instead, *Contracts* clients report a compact, plain-text *digest* of receipts.

The algorithm for constructing the receipt digest message is given in Figure 7. The key underlying technique is to trade optional computation at the tracker for a substantial reduction in network traffic. Instead of transmitting full receipts, each digest contains *claims* about receipts received and a verification hash. Claims are a plain-text list of contributions that allows the tracker to reconstruct the original contribution receipts by recomputing them. A digest contains claims for each receipt received since the last digest was sent (line 4). Each claim contains the first *n* bits of the public key of the receiver specified in the full receipt (line 6). Each truncated key serves as an index, allowing the tracker to map an identifier to a public/private key pair it previously generated for a particular user. Finally, a hash chain is computed over the original receipts (line 7) sorted by receiver identifier (line 3). This can be used by the tracker to verify that claims correspond to valid receipts.

A list of claims informs the tracker as to which receivers generated receipts, but to recompute those original receipts and verify the hash chain, the tracker also needs to know the number of blocks received and the receipt nonce. Both of these are set by the tracker when clients initially connect. The block threshold for dispatching receipts, V, is set to control overhead both at the tracker and among clients. Each client's nonce is selected at random by the tracker and incremented by clients per-peer for each receipt received. For example, if a client's initial nonce is 5 and it receives 2 receipts from peer *A* and 3 from peer *B* in a given reporting interval, subsequent receipts minted by *A* and *B* to this client will be stamped with nonce values of 7 and 8, respectively. The tracker verifies increments to nonce values to prevent replay attacks, and nonce values are maintained on a per-peer basis to prevent concurrent data transfers from producing receipts with the identical nonce values.

At the tracker, ranking clients based on the plain-text claims in digests requires little overhead relative to the existing processing already done by the tracker; table lookups provide the required information to compute Equation (1) (where the sum of contributed block claims per update interval provides contribution rates). Although *processing* is straightforward, *verification* is computationally intensive, requiring the tracker to regenerate and hash each signed receipt. But, since only the plain-text content of digests is needed to rank clients, the tracker can shed load at any time. While sampling digests may increase susceptibility to cheating, our evaluation shows that verifying all digests on the fly is feasible given PPLive's current infrastructure provisioning.

5.4 Collusion resistance

Contracts includes both centralized and distributed verification of receipts to allow the tracker to manage the tradeoff between protocol overhead and robustness to malicious behavior. In the absence of Byzantine behavior, distributed verification effectively rewards contribution without relying on centralized accounting. With isolated Byzantine agents, coordinating topology updates at the tracker enables convergence even while some nodes deviate from our default contract. This increases overhead, but as we show in our evaluation, not prohibitively.

In the remainder of this section, we describe the techniques used by a tracker to use its global perspective to mitigate security attacks, in particular, the well-known P2P attack: collusion, in which a group of participants work collectively to subvert our accounting mechanism. The collusion participants we consider may include both real users with interest in receiving stream data, as well as synthetic identities created strictly for collusion.

Limited identity creation: The tracker appeals to standard techniques used by other P2P proposals for inhibiting the creation of arbitrarily many synthetic identities, the so-called Sybil attack [7]. In particular, the tracker limits the creation of new identities on the basis of durable identifiers, e.g., cell phone number via SMS.

Flow integrity check: When a new client joins a broadcast, the tracker evaluates its maximum upload capacity. Although a client may choose to upload at a lower rate, it cannot exceed the capacity. This restricts potential false

claims on $BC(\cdot)$. In addition, live streaming imposes a known incoming rate bound on each client's long-term incoming data rate, which is the streaming rate. When verifying receipts, the tracker validates the upload capacity and incoming rate bounds. Such verification limits the collusion of a set of broadcast participants to issue fraudulent receipts. No group of colluders can form a loop and arbitrarily boost a colluder's contribution value. Specifically, consider a client x with the support of a total of K colluders. Assume that x is an actual broadcast participant that needs to receive actual data from non-colluders. Then x cannot issue any fraudulent receipts, as it needs to issue receipts for actual data. The capability of the colluders to help x is also limited. The value $B(x \rightarrow p)$, where p is a colluder, is limited by the streaming rate r due to incoming rate bound on p. Thus, with K colluders generating fraudulent receipts, x can claim at most $K \cdot r$ fraudulent contributions to these colluders. But, $K \cdot r$ cannot exceed the upload rate of x measured by the tracker for WAN traffic. Further, if a given colluder p helps x to claim contribution rate r, then $B(p' \rightarrow p)$ should be zero for any other client p', otherwise p violate its incoming rate bound. Thus, if a collusion scheme is to let $B(x \rightarrow p)$ be r for all K colluders, then $B(p)$ has to be zero for all of the colluders. This substantially limits $I(x)$.

Global and diversity weighting: In spite of the preceding checks, some clients might still be able to collude and/or acquire several synthetic identities to increase the overall value of $V(\cdot)$ of a client. To address this, the tracker detects a cluster of linked colluders. Also, *Contracts* can optionally weight the overall value of $V(\cdot)$ by the network-level address diversity of the peers to which a client contributes. As a consequence of registering for a broadcast, the tracker knows each client's IP address and port. For identities within the same IP prefix (/24), *Contracts* dampens the value of contributions when using centralized verification. For identities registered at the same address (e.g., users behind a NAT), contributions are further dampened. This policy restricts collusion by exploiting the scarcity of IP addresses.

Note that we do not adopt a universal notion of client utility, and we do not claim that *Contracts* is strategyproof, even given these defenses. An alternative approach to mitigating collusion and strategic behavior is to restrict each client's choice in peer selection. As shown in previous work [17, 18], limiting peer selection is a powerful tool for enabling formal analysis of gossip protocols since the potential for protocol deviations is restricted. But, such restrictions may limit the potential for grouping peers based on locality or bandwidth, e.g., high bandwidth, local exchange between peers on the same LAN. In practice, flexible peering significantly increases distribution efficiency in PPLive, leading us to eschew restrictions which may aid in formal analysis, leaving open these issues for future work.

6 Evaluation

Our evaluation of *Contracts* answers two main questions. First, is applying *Contracts* to streaming systems feasible? We find that it is; *Contracts* adds modest overhead but does not fundamentally limit scalability. Second, is *Contracts* effective? To confirm this, we report measurements of a modified PPLive client to demonstrate the performance improvement of *Contracts* relative to other systems and incentive strategies. Specifically:

- *Contracts* improves performance relative to unmodified PPLive. In experiments with heterogeneous capacities and churn, *Contracts* increases the number of clients with uninterrupted playback from 13% to 62%, an increase of more than 4×.
- *Contracts* provides robust contribution incentives. Experiments in bandwidth constrained environments show that quality of service improves with contribution. Moreover, *Contracts* provides a substantial and consistent improvement in quality of service relative to tit-for-tat.
- *Contracts* is scalable, even when using centralized verification. Using our default parameters, a single *Contracts* tracker can support the computational and network overhead of more than 90,000 concurrent clients.
- Clients are quickly integrated into the mesh. After only a few rounds of peer exchanges, newly joined clients percolate to their intended locations in the overlay with bandwidth clustered peers.

6.1 Performance and incentives

We first evaluate the performance of our *Contracts* implementation, which is built from modifications to the reference PPLive client. We show two main results: 1) PPLive with *Contracts* significantly outperforms both unmodified PPLive and one modified to support tit-for-tat (TFT). 2) *Contracts* provides our intended contribution incentives; when the system is bandwidth constrained, increasing contribution improves performance.

Performance: We define performance as the fraction of data blocks received by their playback deadlines, and compare performance for PPLive, PPLive using *Contracts*, PPLive with tit-for-tat, and FlightPath [17]. For each of these techniques, we measure the performance of 100 clients participating in a test broadcast on Emulab.[6] Each client initially joins the system separated by a ten-second interval. To evaluate *Contracts* under churn, each client disconnects and rejoins after participating for 20 minutes. All clients continue this process for two hours. To compare performance under realistic bandwidth constraints, client upload capacities are drawn from our mea-

[6] Emulab allows us to execute Windows binaries.

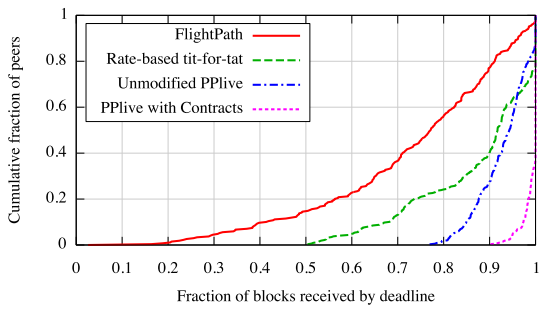

Figure 8: Performance comparison of unmodified FlightPath, PPLive, rate-based tit-for-tat, and *Contracts*.

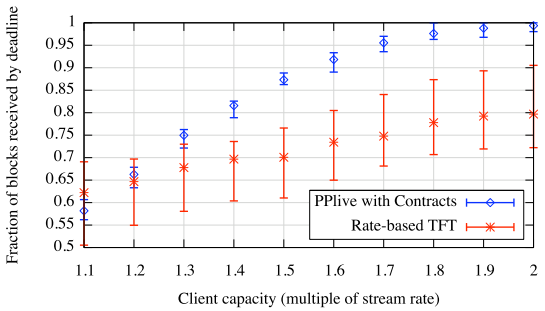

Figure 9: Delivery rate as a function of contribution.

sured capacity distribution of PPLive clients, normalized to provide an over-provisioning factor of 2; i.e., the sum of peer capacities is twice the aggregate demand. Crucially, however, many peers have capacity less than the stream data rate—a common occurrence in practice. Both TFT and *Contracts* clients actively exchange data with 10 directly connected peers and reevaluate these decisions every 10 seconds using the statistics of previous 30 seconds. For FlightPath trails, we use default configuration parameters described by Li, et al. [17].

Figure 8 shows our results. *Contracts* significantly improves performance relative to unmodified PPLive and FlightPath; 62% of *Contracts* clients experience loss-free playback compared with just 13% when using unmodified PPLive or 3% when using FlightPath. In other words, the fraction of PPLive/*Contracts* clients experiencing loss-free playback is more than 4 times that of unmodified PPLive. For clients that do miss playback deadlines, a larger fraction of blocks arrive in time when using *Contracts*. Relative to unmodified PPLive, tit-for-tat degrades performance for the majority of clients. This is consistent with our analysis in Section 3. Tit-for-tat benefits high capacity clients when they happen to be placed near the broadcast source ($y > 0.96$). But, more distant clients cannot collect enough useful data with which to trade. Even high capacity clients cannot prove their capabilities when far from the source, decreasing overall utilization and average performance.

Incentives: *Contracts* rewards contribution with increased robustness. We evaluate this by comparing the performance of PPLive using *Contracts* with that of PPLive using tit-for-tat. In both cases, the system is bandwidth constrained. We use 100 clients with capacities uniformly distributed between 1–2× the stream rate (over-provisioning factor 1.5) to connect to a test stream, participating in the broadcast for 10 minutes. We repeat this experiment 10 times.

Figure 9 shows the results. Averages are shown with error bars giving the full range of block delivery rates for clients with a given capacity. While tit-for-tat does provide some correlation between contribution and performance, the amount of improvement varies significantly because tit-for-tat does not update the topology. In contrast, *Contracts* combines both topology updates and local servicing rate decisions to provide a consistent improvement in performance, strengthening incentives.

6.2 Overhead

In this section, we describe implementation details and overhead related to verifying contributions, including: 1) state maintained by the tracker and clients, 2) computation required to verify receipts, and 3) network control traffic. We discuss each of these in turn.

State: When using centralized verification, the PPLive tracker maintains soft state including bandwidth capacity, client version, etc. of active clients. To these, *Contracts* adds a last digest update field which records the timestamp and content of the most recently received receipt digest message. This is used to compute contribution rates when new digests are received, and its size varies depending on content. We estimate the likely size of receipt digest messages when computing network overhead (described below).

Trackers also maintain hard state: the key pairs of registered clients. For cryptographic operations, *Contracts* uses SHA-1 with RSA, DER-encoded PKCS#1 and 1024 bit keys. Maintaining one million key pairs requires less than a gigabyte of storage on disk. A lookup table mapping truncated identifiers to keys easily fits in memory on modern servers. For distributed verification, clients associate public keys and nonces with connections and maintain counters of verified receipts received from each directly connected peer.

Network traffic: Exchanging receipt and receipt digest messages is the main source of network overhead in *Contracts*. Three related parameters influence this. The tracker specifies a *digest interval* indicating how often digests are reported by clients. A lengthy interval reduces the number of such messages at the cost of delayed topology updates or delayed detection of cheating clients. The *receipt volume* specifies how much data each receipt acknowledges. Finally, the *stream data rate* controls how many receipts are exchanged among peers. To make our analysis independent of stream data rate, we define receipt volume in terms of how many seconds of

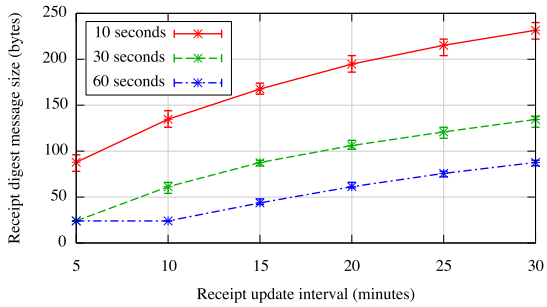

Figure 10: The size of receipt digest messages as a function of the digest update interval.

video data each receipt acknowledges. Currently, *Contracts* uses a digest update interval of 15 minutes and a receipt volume that acknowledges 30 seconds of stream data. In the remainder of this section, we examine the tradeoffs underlying these choices.

For a video stream of moderate quality (500 Kbit), sending a receipt acknowledging every 30 seconds of video data imposes less than 0.1% overhead relative to data transfer among peers when verifying contributions at the tracker. Distributed verification requires forwarding additional receipts from each peer's one hop neighborhood. This increases average network overhead to 1.2%, trading an increase in traffic among peers for a reduction in traffic at the tracker, which we consider next.

Network overhead at the tracker is determined by the number of receipt digest messages received. Each receipt digest message contains a 24 byte header and 6 byte tuples specifying a peer (4 byte truncated public key) and a receipt count (2 bytes). In the worst case, each digest would include an entry for every directly connected peer.

In practice, only a fraction of connected peers are included in a single digest update. To compute this, we measured the amount of data uploaded to directly connected peers by an instrumented PPLive client that participated in popular broadcasts for 10 minutes apiece. Contribution is highly skewed; for each client, the top 10% of its peers receive 60% of its contributed data, meaning that there are fewer entries in each digest.

Combining our measurements of skew with the typical number of directly connected peers allows us to compute the size of receipt digest messages. Figure 10 shows this data for several receipt volume values. Each line shows the growth in the size of a digest message as a function of the update interval. Each data point is averaged over 10,000 randomly generated digest messages using samples from measured distributions to specify the directly connected peers and capacity skew. To compute aggregate traffic at the tracker, we multiply the average receipt digest size by the total number of clients. For instance, processing digests for 100,000 clients with our default parameters requires 10 KBps of tracker overhead.

Computation: Computational requirements at the clients are dominated by the demands of video playback. At the tracker, the computational overhead of *Contracts* is dominated by receipt verification. Verification requires regenerating the receipt messages specified by receipt digest messages and computing the SHA-1 hash chain for the generated receipts to verify the hash specified in the digest (Figure 7). Thus, the computational overhead of verification depends on the number of receipts, which is determined by the stream data rate and receipt volume.

The total number of receipts per second generated by a channel is simply the ratio of data rate and receipt volume multiplied by the population. A micro-benchmark on a single commodity server using our current implementation can verify 3,200 receipts per second, and receipt verification is embarrassingly parallel. If receipts encapsulate 30 seconds worth of video data, our current implementation can verify receipts for more than 90,000 simultaneous clients in real-time using a single server. In practice, management of so large a broadcast is already distributed across several servers in PPLive, meaning that receipt verification with *Contracts* does not dominate resource usage when scaling the coordination infrastructure. As with network overhead, *Contracts* allows the tracker to shed computational load when required. receipt digest messages that are not cryptographically verified can still be used to evolve the topology and (optionally) stored for later verification. This increases the window of vulnerability to a cheating client but does not degrade the efficiency of distribution.

6.3 Convergence

We next consider the integration of new clients into the mesh. Convergence of clients to their intended location in the topology is determined by many factors. We consider two explicitly: 1) the capacity of a newly joined client, and 2) the number of newly joining clients; e.g., integrating a flash crowd may require additional peer exchanges relative to integrating a single client into a stable mesh. We measure convergence in terms of update rounds; i.e., the interval between peer gossip connections. To understand convergence at scale, we use trace-driven simulation of *Contracts* using default parameters.

We first evaluate convergence as a function of a newly joined client's bandwidth capacity. For each capacity, the new client joins a 10,000 user channel with stable membership, and we record the number of topology updates required for the newly joined client to reach a stable position. We consider a client to have reached a stable position when the average capacity of its net contributors (i.e., those that provide more blocks than they receive) is within 5% of the average capacity of net consumers. The vast majority of peers ($> 80\%$) reach a stable position in four update rounds or less. Broadly, the results are consistent with the variation in observed bandwidth capacity.

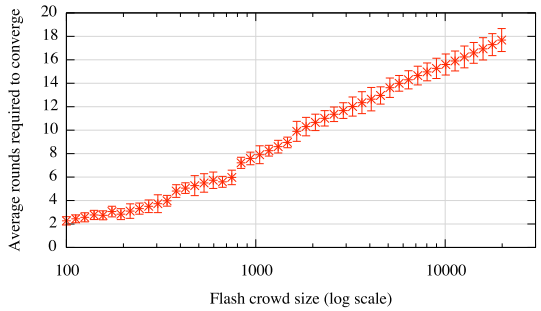

Figure 11: The number of peer exchanges required for a set of newly joined clients to reach stable matchings as a function of the number of arrivals in a flash crowd.

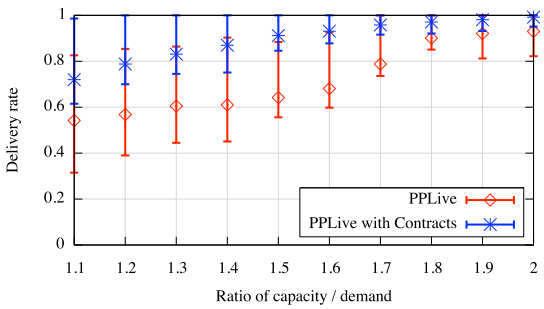

Figure 12: The impact of over-provisioning on PPLive's performance. Data points show the average fraction of blocks received by their playback deadlines.

Low capacity peers can quickly discover a stable set of similarly low capacity peers, while high capacity peers need several rounds to stabilize.

Next, we consider topology convergence for flash-crowd arrivals. In this case, we simulate a channel with 1,000 initial participants and vary the number of joining clients. Each new client is assigned a capacity drawn from the same distribution as the existing clients, providing a constant amount of resources in the system. Figure 11 shows the number of rounds required to achieve stability for the last newly joined client in the crowd. The number of rounds required increases logarithmically with the number of joining peers.

6.4 Over-provisioning

Our evaluation thus far has focused on the performance of *Contracts* in settings with a specific amount of over-provisioning; i.e., capacity in excess of total demand. We now evaluate over-provisioning directly, measuring the performance of PPLive and *Contracts* while scaling our measured capacity distribution to vary the ratio of capacity to demand. We measure the block delivery rate of 100 static PPLive clients running on Emulab. As in previous experiments, we record each client's block delivery rate. Figure 12 summarizes the results, with error bars showing the 5^{th} and 95^{th} percentiles of delivery rate across all clients. In each trial, the average delivery rate PPLive using *Contracts* exceeds that of unmodified PPLive. When capacity is limited, low capacity clients are penalized by *Contracts*, contributing to variations in performance. As capacity increases, however, *Contracts* delivers consistently higher quality of service for all peers. Taken together, these results show that *Contracts* achieves consistently higher performance for a range of operating conditions, and delivers on our overall goals of improving efficiency and providing contribution incentives. When the system is capacity rich, *Contracts* improves distribution efficiency, improving delivery rate for all peers. But, during periods of resource contention, high capacity peers receive better quality of service.

7 Related work

Our work builds on a large body of prior work focused on live streaming, P2P data distribution, and incentives.

The notion of a P2P approach to data streaming was pioneered by Narada, Overcast and Yoid [4, 14, 8]. These designs tried to approximate multicast support using a tree structured overlay. SplitStream, a subsequent design, addressed the limited utilization of leaf nodes in tree-based schemes [2].

Subsequent work has applied swarming designs, borrowed from BitTorrent-like systems, to video-on-demand and live streaming. Coolstreaming/DONet applies a mesh-based network structure to live streaming [28]. Annapureddy, et al. argue that high quality video on demand is feasible using a P2P architecture, a point reinforced by recent work describing PPLive's video-on-demand P2P implementation [13] as well as other publicly available commercial streaming systems (e.g., PPStream, SopCast, TVAnts, and UUSee).

More recent work has studied incentives in bulk data distribution in widely deployed systems, particularly BitTorrent. Qiu and Srikant studied BitTorrent formally, finding that it achieves a Nash equilibrium under certain conditions [25], although more recent work has shown practical mechanisms for subverting BitTorrent's incentive strategy [22]. These advancements in understanding the subtlety of bilateral exchange motivated our consideration of its application to live streaming. In [1], Aperjis, et al. extend bilateral exchange to multilateral exchange by introducing prices. They compare their scheme with BitTorrent and show improvements in efficiency and robustness. One hop reputations [23] use limited propagation of contribution information to improve incentives in BitTorrent; we apply similar ideas to live streaming.

Most related to our work are systems that address incentives in live streaming (e.g., [10, 12, 18, 19, 21, 26, 27]). Sung, et al. describe a live streaming design that rewards contribution but depends on honest capacity reporting by peers [27]. SecureStream introduces proto-

col mechanisms to defend against several attacks (e.g., forged data and denial of service [12]). These techniques are largely complementary to our work, which focuses on verifiably rewarding contribution. BAR Gossip analyzes incentives in streaming formally and describes a protocol designed to induce contributions from rational users [18]. FlightPath relaxes several constraints of BAR Gossip (e.g., by allowing dynamic membership), but still requires the long-term balance of contribution and consumption to provide contribution incentives. Our experience applying rate-based tit-for-tat is consistent with that of Pianese, et al. who apply TFT to live streaming and experimentally confirm the need for significant altruism to achieve robustness [21].

Motivated by the practical challenges of client heterogeneity, we take a different approach in the design of *Contracts*, providing incentives via a global contract and including explicit topology restructuring in our algorithm design. Habib, et al. propose providing high capacity clients with additional peers to improve their service quality [10], but such improvements are not assured in environments with high levels of bandwidth heterogeneity.

Several live-streaming systems focus on providing robustness by enforcing contribution amounts. Chu, et al. propose mandatory, centrally enforced taxation in the context of multi-tree live streaming [3]. Haridasan, et al. provide a two-level auditing scheme for live streaming that ensures that peers contribute more than a threshold amount of data [11]. Local auditing and gossip provide an immediate but partial check on user's contribution, while global audit ensures that a misbehaving node is caught. Rather than punishing nodes that do not contribute a sufficient amount, *Contracts* rewards nodes for voluntarily contributing as much as possible.

8 Conclusion

We have examined performance and contribution incentives for live streaming systems. The unique features of the P2P live streaming environment limit the effectiveness of many widely-used incentive strategies based on balanced or bilateral exchange. These challenges motivate the design of *Contracts*, a new incentive strategy that rewards contribution with quality of service by evolving the overlay topology. Building on a protocol that provides verifiable contributions, we have shown that the use of *Contracts* both improves performance relative to PPLive and strengthens contribution incentives relative to existing approaches without curtailing scalability.

Acknowledgments

We would like to thank the anonymous reviewers and our shepherd, Lorenzo Alvisi, for their valuable feedback.

References

[1] C. Aperjis, M. Freedman, and R. Johari. Peer-assisted content distribution with prices. In *CoNEXT*, 2008.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: high-bandwidth multicast in cooperative environments. In *SOSP*, 2003.

[3] Y. Chu, J. Chuang, and H. Zhang. A case for taxation in peer-to-peer streaming broadcast. In *SIGCOMM PINS*, 2004.

[4] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *SIGMETRICS*, 2000.

[5] B. Cohen. Incentives build robustness in BitTorrent. In *P2P-ECON*, 2003.

[6] Comcast limits download volume. `http://online.wsj.com/article/SB122004003325884079.html`.

[7] J. R. Douceur. The Sybil attack. In *IPTPS*, 2002.

[8] P. Francis. Yoid: Extending the internet multicast architecture. Available at `http://www.icir.org/yoid/docs/`, 2000.

[9] C. Gkantsidis, J. Miller, and P. Rodriguez. Comprehensive view of a live network coding P2P system. In *IMC*, 2006.

[10] A. Habib and J. Chuang. Service differentiated peer selection: an incentive mechanism for peer-to-peer media. In *IEEE Trans. Multimedia*, 2006.

[11] M. Haridasan, I. Jansch-Porto, and R. van Renesse. Enforcing fairness in a live-streaming system. In *MMCN*, 2008.

[12] M. Haridasan and R. van Renesse. SecureStream: An intrusion-tolerant protocol for live-streaming dissemination. *Computer Communication*, 31(3):563–575, 2008.

[13] Y. Huang, T. Fu, D. Chiu, J. Lui, and C. Huang. Challenges, design and analysis of a large-scale P2P-VoD system. In *SIGCOMM*, 2008.

[14] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole. Overcast: Reliable multicasting with on overlay network. In *OSDI*, 2000.

[15] J.-J. Laffont and D. Martumort. *The Theory of Incentives: The Principal-agent model*. Princeton University Press, 2002.

[16] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. BitTorrent is an auction: analyzing and improving BitTorrent's incentives. In *SIGCOMM*, 2008.

[17] H. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robinson, L. Alvisi, and M. Dahlin. FlightPath: Obedience vs choice in cooperative services. In *OSDI*, Dec 2008.

[18] H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR gossip. In *OSDI*, Nov. 2006.

[19] Z. Liu, Y. Shen, S. Panwar, K. Ross, and Y. Wang. Using layered video to provide incentives in P2P live streaming. In *P2P-TV*, 2007.

[20] Z. Liu, C. Wu, B. Li, and S. Zhao. Distilling superior peers in large-scale P2P streaming systems. In *INFOCOMM*, 2009.

[21] F. Pianese, D. Perino, J. Keller, and E. W. Biersack. PULSE: An adaptive, incentive-based, unstructured P2P live streaming system. *IEEE Transactions on Multimedia*, 2007.

[22] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent? In *NSDI*, 2007.

[23] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson. One hop reputations for P2P file sharing workloads. In *NSDI*, 2008.

[24] M. Piatek, A. Krishnamurthy, A. Venkataramani, R. Yang, and D. Zhang. Contracts: Practical contribution incentives for P2P live streaming. Technical Report, UW CSE., 2010.

[25] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *SIGCOMM*, 2004.

[26] T. Silverston, O. Fourmaux, and J. Crowcroft. Towards an incentive mechanism for peer-to-peer multimedia live streaming systems. In *International Conference on Peer-to-Peer Computing*, 2008.

[27] Y.-W. Sung, M. Bishop, and S. Rao. Enabling contribution awareness in an overlay broadcasting system. *SIGCOMM*, 2006.

[28] X. Zhang, J. Liu, B. Li, and T.-S. P. Yun. CoolStreaming/DONet: A data-driven overlay network for live media streaming. In *INFOCOMM*, 2005.