Novasky: Cinematic-Quality VoD in a P2P Storage Cloud

Fangming Liu[†], Shijun Shen[§], Bo Li[†], Baochun Li[‡], Hao Yin[§], Sanli Li[§]
[†]*Hong Kong University of Science & Technology*, [§]*Tsinghua University*, [‡]*University of Toronto*

***Abstract*—In this paper, we present *Novasky*, a real-world Video-on-Demand (VoD) system capable of delivering cinematic-quality video streams to end users. The foundation of the *Novasky* design is a peer-to-peer (P2P) storage cloud, storing and refreshing media streams in a decentralized fashion using local storage spaces of end users. We present our design objectives in *Novasky*, and how these objectives are achieved using a collection of unique mechanisms, with respect to caching strategies, coding mechanisms, and the maintenance of the supply-demand relationship when it comes to media availability in the P2P storage cloud. The production Novasky system has been implemented with over 100,000 lines of code. It has been deployed in the Tsinghua University campus network, operational since September 2009, attracting 10,000 users to date, and providing over 1,000 cinematic-quality video streams with bit rates of 1 – 2 Mbps. Based on real-world traces collected over 6 months, we show that *Novasky* can achieve rapid startups within 4 – 9 seconds, and extremely short seek latencies within 3 seconds. Our empirical experiences with *Novasky* may bring valuable insights to future designs of production-quality P2P storage cloud systems.**

I. Introduction

With its great potential to bring a rich repository of video content to end users on-demand, video-on-demand (VoD) systems have not only been the target of a substantial amount of research, but also been core industry products in both startup and established corporations alike. Existing research has extensively studied peer-to-peer (P2P) VoD systems from theoretical (*e.g.,* [1]), simulations (*e.g.,* [2]), and measurement perspectives (*e.g.,* [3]).

It has been well known that such systems require each user, called a *peer*, to dedicate a certain amount of the non-volatile storage space on her local host. Such local storage space from users will be used in typical P2P VoD streaming systems as a semi-persistent *cache* of the media content that has just been played. The benefits brought forth by such local caching are two-fold: to assist other users from the cache in a peer-to-peer fashion, and to improve the quality that the user experiences with respect to important quality metrics, such as random seek latencies — the time it takes for any random seek operations to complete. Caching strategies in typical P2P VoD streaming systems have been discussed extensively in the literature, with a focus on improving the utility of cached content with respect to performance.

*The research was support in part by a grant from RGC under the contract 615608, by a grant from Huawei Technologies Co. Ltd. under the contract HUAW18-15L0181011/PN, and by grants from project 60873254, 60873202 supported by NSFC and the project 2011CB302600 supported by the National Basic Research Program of China (973 Program).

More recently, the advent of cloud computing has introduced the concept of a *storage cloud*, which represents a large-scale pool of storage space that can be accessed conveniently by users who need storage space in the cloud. The amount of storage space reserved by a P2P VoD system in each end host is quite substantial in the real world, typically more than 1 GB. Connecting these two "dots" together, if there exists a fair number of concurrent users in a traditional P2P system, wouldn't it be intriguing to take advantage of the local VoD storage space that belongs to a large number of participating peers, pool them together, and form a *P2P storage cloud?* This is especially feasible and interesting when participating peers are inter-connected with a local, high-bandwidth network — such as a typical campus network.

One may naturally wonder why such a P2P storage cloud may offer anything more interesting than a traditional P2P VoD streaming system with local caching. We believe that, as peers are inter-connected with a high-bandwidth network, such a P2P storage cloud, if designed, implemented, and managed appropriately, would be able to offer a stellar level of performance when streaming on-demand video to participating users, one that is much better than traditional P2P VoD systems over the public Internet.

In this paper, we present *Novasky*, a production-quality VoD system that we implemented with over 100,000 lines of code (LOC) in C++, based on the foundation of our P2P storage cloud that we have designed and implemented from scratch. In retrospect, when we brainstorm to finalize the design of the *Novasky* P2P storage cloud architecture a year ago, no one would ever imagine that its production deployment in the campus network at Tsinghua University has delivered over 1,000 video streams to over 10,000 users at *cinematic quality*, streaming at bit rates of 1 – 2 Mbps. We are pleasantly surprised by (and advertising the product for) the fact that, when played back, these video streams can be enjoyed at a quality level of DVD or even 720p video. In contrast, most traditional P2P VoD systems operate at low bit rates, such as 400 kbps [2]. Beyond sustainable streaming rates, *Novasky* users are also experiencing typical random seek latencies of 3 seconds, while traditional systems suffer from random seek latencies of 15 – 40 seconds, or even longer [2].

Rather than typical emulation-based experiments, with *Novasky*, we choose to design, implement, deploy, and measure a complete VoD system based on a P2P storage cloud over a high-bandwidth network. The *Novasky* P2P storage cloud architecture features a number of unique highlights, which are the focus of this paper. *First*, we believe that simple replication

strategies typically used in cache design are not efficient when a high level of video availability needs to be maintained across the P2P storage cloud. To maintain a well-designed and balanced tradeoff between the performance of streaming "hot" videos and the diversity of available videos, we have decided to use Reed-Solomon codes [4], tightly coupled with a new coding-aware storage replacement algorithm. *Second*, to maintain a healthy relationship between the "supply" of and "demand" for content and bandwidth, we designed an adaptive server-feeding strategy that actively pushes content to peers in the P2P storage cloud.

During the brainstorming stage, we were worried about the use of Reed-Solomon codes in the *Novasky* P2P storage cloud. Though MDS codes, such as Reed-Solomon codes, have been widely adopted in traditional mass storage systems (*e.g.*, RAID 6), and have been conceptually shown to be effective in distributed storage systems [5], [6], they have not been used in practice in any production-quality P2P storage cloud systems. We were mainly concerned with their computational complexity.

The good news is that our worries were dissipated by evidence from our extensive measurement studies presented in this paper. Throughout subsequent sections in this paper, we evaluate the scale, quality, and lessons learned from practically every conceivable aspect in *Novasky*: its quality perceived by the users, its performance as a system, especially with respect to the efficacy of coding, as well as its operational overhead. To the best of our knowledge, *Novasky* is the first attempt in the literature towards designing and implementing an operational P2P storage cloud for on-demand streaming of cinematic-quality videos over a high-bandwidth network. We are convinced that results reported in this paper can offer valuable insights towards a new genre of cinematic-quality VoD services in real-world systems.

II. Novasky: Challenges and Design

In this section, we first identify important challenges as we design the architecture of the *Novasky* P2P storage cloud. In response to these challenges, we then present our system architecture and design, including code-based peer storage and adaptive push-to-peer mechanisms.

A. Design Objective and Challenges

Our ultimate design objective is to build a P2P storage cloud that is specifically tailored to the performance needs of cinematic-quality VoD streaming over a high-bandwidth network. To achieve such an objective, two important challenges need to be addressed.

▷ Despite a variety of cache replacement strategies in existing P2P VoD systems [2], [3], when taking a system-wide perspective, they are mostly restricted to replication of original video data across the local storage space of participating peers, due to its simplicity. However, given limited peer cache sizes, such strategies would become insufficient to maintain data availability and diversity, and are especially ill-conceived for on-demand streaming of cinematic-quality video. We need a new and cost-effective storage and replacement strategy, in order to improve the efficiency of utilizing valuable space in the P2P storage cloud, and to maintain availability of videos.

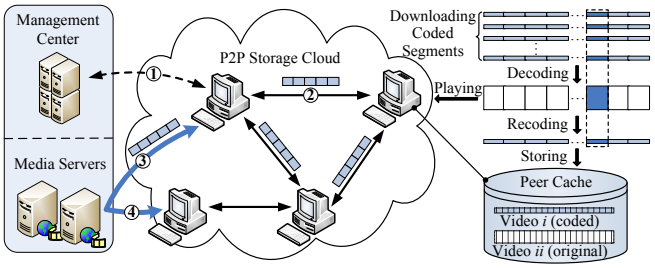


Fig. 1. The architecture of *Novasky*.

▷ With the presence of frequent peer departures and the evolution of diverse user interests across a rich repository of videos, it is not uncommon to have a lack of balance between the demand and supply of content and bandwidth, which brings adverse effects to the availability of videos and the streaming performance. We seek to proactively adapt the availability of videos in the P2P storage cloud based on their popularity, yet with effective utilization of limited bandwidth available at media servers.

B. Novasky: Architectural Design

Fig. 1 illustrates our system architecture, and interactions among major components in *Novasky*. Arrow 1 represents the interaction between peers and the Management Center to manage the P2P storage cloud. Arrow 2 represents the gossip communication and video data distribution among peers. By concurrently downloading coded video blocks from multiple serving peers, a viewing peer can progressively decode and playback the video. Each peer is able to cache both original and coded segments of multiple videos. Along arrows 3 and 4, the Media Servers adaptively and proactively push coded segments of under-supplied videos to selected peers.

The *Management Center* (MC) consists of a set of servers that have a number of responsibilities: (1) maintaining meta-data in all media files, including the MD5 hash value, length, and popularity of each file; (2) authenticating users using the RSA-based public key infrastructure; and (3) maintaining the current status of participating peers in the P2P storage cloud, including the status of their local storage space. Routine tasks such as NAT traversal are also handled.

The *Media Servers* (MS) cooperate with the P2P storage cloud to feed a rich repository of videos to a large number of users. As we shall elaborate in Sec. II-D, the objective of the Media Servers is to utilize limited server bandwidth in the best ways possible, and to maximize content availability and service quality in the P2P storage cloud.

The *P2P storage cloud* is designed to "pool" local storage spaces on all participating peers together, and collectively optimize the delivery of cinematic-quality video streams. The P2P storage cloud is derived from traditional P2P designs: all participating peers are organized into a mesh topology, and exchange information about the availability of video segments periodically.

Algorithm 1 The peer storage and replacement strategy using Reed-Solomon codes.

```
// when the current cache space is saturated and cannot store new incoming video data
sort currently cached original videos in LFU order;
for each original video do
    // code-based replacement
    use Reed-Solomon codes to generate a coded segment to be stored, with a randomly selected index from the Vandermonde matrix;
    evict the original video from the cache;
    if the cache space is enough for the new incoming data then
        store the new incoming data; return;
    end if
end for
// if cache space is still insufficient to accommodate new incoming video data
sort currently cached coded segments in LFU order;
for each coded segment do
    evict the coded segment from the cache;
    if the cache space is enough for the new incoming data then
        store the new incoming data; return;
    end if
end for
```

Finally, *Novasky* provides a feature-rich user interface. In addition to on-demand streaming and random seek functions, *Novasky* users are able to navigate and search in the video repository, rate videos, and upload user-generated content. From the perspective of a user using its interface, the *Novasky* P2P storage cloud architecture is completely transparent: it operates exactly like a cloud storage service. Users are not aware of the fact that, rather than storing videos in the "cloud," they are stored in the collective "pooled" storage that all users contribute to.

C. Peer Storage and Replacement using Reed-Solomon Codes

In a P2P storage cloud, the most critical design objective is to efficiently utilize limited peer storage cache capacities to maintain video availability and to maximize peer upload contribution. To achieve this objective, there are three important aspects that we have considered as we design *Novasky*: (1) Each peer is expected to cache a number of videos to serve as many peers as possible. However, given a limited cache size such as 2 GB, it is infeasible to store all complete videos. (2) A viewing peer is usually fed by multiple seeding peers with limited upload capacities, each of which is responsible for uploading a certain part of video data. Since a seeding peer only provides a part of video data, it is unnecessary to store the complete video. (3) The rich repository of high-quality videos with bit rates of $1-2$ Mbps could further strain peer storage and bandwidth resources, rendering simple replication strategies insufficient to maintain video availability and transmission efficiency.

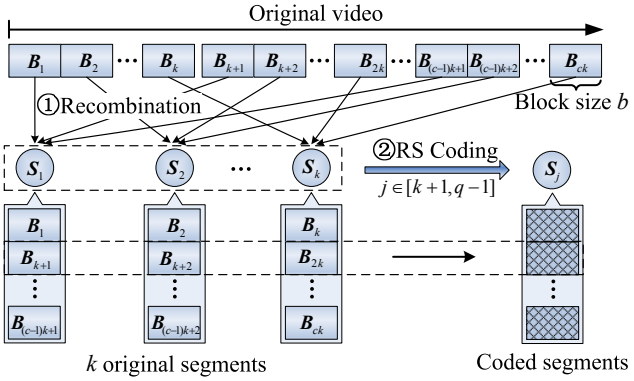


Fig. 2. The coding process from original video to coded segments.

Motivated by these considerations, our design choice is to store a part of the video data instead of the complete video, in order to better utilize limited peer cache capacities, and to promote diversity. However, this may bring adverse effects on video availability and on-demand streaming performance. How can we carefully store a part of video data while mitigating such adverse effects, and in what form should the video data be stored to maintain video availability and storage efficiency?

In response, we would like to take advantage of Reed-Solomon codes when video is stored in the peer local storage cache. Algorithm 1 shows our strategy to replace original video segments with *one* coded segment, if we need to reclaim cache space to store a new incoming video. In particular, to combine coding with replication within the peer storage cache, *Novasky* performs the following: (1) When a peer has adequate cache space, it stores the complete video as usual so that it may serve more data to viewing peers that do not have a sufficient number of seeding peers. (2) When the cache space becomes fully utilized as a peer watches more videos, currently cached videos are sorted in the Least Frequently Used (LFU) order, and following that order, each video file will be coded to *one* coded segment, using a (n, k) Reed-Solomon code, in which n is the maximum number of unique coded segments possible, and k is the number of original segments to be coded. Since only one coded segment will remain, it will occupy a smaller amount of cache space as compared to the original video. The purpose is to free up a sufficient amount of storage space to accommodate a new incoming video, while still keeping the availability of coded videos.

There are, however, a number of more detailed challenges that the outline of our design does not discuss.

Applying Reed-Solomon codes to segments. The first natural question that arises is: How are *segments* defined, and how are Reed-Solomon codes applied to these segments? In *Novasky*, the original (complete) video of size L is divided into contiguous data blocks of size b, *i.e.*, $\{B_i | i = 1, 2, \ldots, ck\}$, where i is the sequence number of blocks, and $c = \lceil \frac{L}{kb} \rceil$ is the number of blocks within a segment. Of course, additional zeros can be added to the end of the last segment of a video. These blocks are recombined in an *interleaved* manner to form k original segments $\mathbf{S}_j = \{B_{ik+j} | i = 1, 2, \ldots, c-1; j =$

$1, 2, \ldots, k\}$ with indices $j \in [1, k]$, so that each segment consists of interleaved blocks across the entire video, as shown in Fig. 2. A seeding peer in *Novasky* independently and randomly chooses an index j and a corresponding row vector g_j in the Vandermonde matrix — which is the typical generator matrix used in systematic Reed-Solomon codes — and generate a coded segment $\widehat{\mathbf{S}_j} = g_j [\mathbf{S}_1, \mathbf{S}_1, \ldots, \mathbf{S}_k]^T$. When such a choice of index j is made, due to the nature of systematic Reed-Solomon codes, j must be in the range of $[k+1, n]$. Since there are k segments in a video, a coded segment has a size of only $1/k$ of the original video size, and will be stored by the seeding peer to replace the original video when storage space in the cache needs to be reclaimed. A viewing peer can decode and recover video blocks in a pipelined fashion using Gauss-Jordan elimination, as soon as k blocks with the same relative position x of any k linearly independent segments $\mathbf{S}$ are received, instead of waiting for k entire segments to be received.

Choosing parameters n ***and*** k ***in Reed-Solomon codes.*** How parameters n and k should be chosen in *Novasky* hinges upon the need that different peers needs to produce different coded blocks, *i.e.,* using different indices to select different coding vectors in the Vandermonde matrix that Reed-Solomon codes use. In order to minimize the probability of duplicated coded segments, we choose to use a sufficiently large size q of the Galois Field GF(q) in *Novasky*. A good choice would be $q = 2^{16}$, since it is the maximum size of the Galois Field that allows for efficient implementations in software [4], [5]. In this case, based on the requirement of Reed-Solomon codes, $n \leq q - 1$, and the maximum n is $2^{16} - 1$.

To minimize the probability of duplicate index choices by different peers, we choose a reasonably small k, in that $k = 16$. This implies that segments cached by peers are coded by randomly selecting a coding vector from $2^{16} - 1$ rows in the Vandermonde matrix, all of which are guaranteed to be linearly independent from each other, due to the property of the Vandermonde matrix. According to solutions to the birthday attack problem, the probability of having a duplicated selection with k different peers choosing indices from $n = 2^{16} - 1$ rows can be approximated as $1 - e^{-k^2/(2n)}$, which is only 0.20% under $k = 16$ and $n = 2^{16} - 1$.

Interestingly, the selection of $k = 16$ implies that the segment size is only $1/16$ of the original video size. By replacing the original video with a coded segment, our strategy can free up to 16 times less storage space to store new videos. Even larger values of k require a viewing peer to download segments from more seeding peers, which aggravates the scheduling overhead. As shown in recent measurement studies [2], the practical number of seeding peers falls in $[8, 32]$.

Estimating communication overhead. Another practical design consideration is whether we can keep the implementation of code-based storage strategy sufficiently lightweight, with acceptable communication overhead, even with the use of Reed-Solomon codes. With $n = 2^{16} - 1$ as we discussed above, the coding metadata in term of the segment index can be simply represented as an integer of 2 bytes within $[1, 2^{16} - 1]$: original segments with indices $[1, k]$ and coded ones with indices $[k + 1, 2^{16} - 1]$, all of which are linearly independent. The original video can be treated as k original segments, since systematic Reed-Solomon codes are used. In addition, coding vectors from the Vandermonde matrix can be readily buffered by each peer, without incurring extra computation and transmission overhead.

Summary. Our code-based storage and replacement strategy can improve the storage efficiency and video availability of a P2P storage cloud, in the sense that more storage space can be freed to cache and serve a larger number of videos with the same cache size constraints, while coded segments using Reed-Solomon codes can still maintain video availability and peer upload contribution with high probability. This essentially increases the data sharing opportunity among peers, compared to simple replication strategies. Although we can further allow a peer to cache more than one coded segment for a replaced video, our current design choice is to let the peer cache one coded segment. The rationale is that storing multiple coded segments of the same video will consume more storage resources and potentially incur peer load imbalance, without providing extra data availability [5].

D. Adaptive Server Push-to-Peer Strategy

Due to the dynamic nature of the P2P storage cloud, video availability can be degraded at run-time. With the presence of frequent peer departures, new coded video segments need to be periodically replenished in the system to maintain video availability. However, solely relying on peers for such repairs in coded systems may incur tremendous computation overhead and repair bandwidth costs [6]. In particular, regenerating one coded video segment with a (n, k) MDS code requires a peer to download k distinct segments or the entire video. That is, the amount of data that needs to be transferred can be k times higher than the amount of redundancy lost [7]. Furthermore, due to the evolution of diverse user interests across a rich repository of videos, it is not uncommon to have demand-supply imbalance across videos. This requires the redundancy of video in a P2P storage cloud to be proactively adapted to the video popularity, in order to guarantee service quality and user experience.

To compensate for these adverse effects, we design an adaptive server push-to-peer strategy in the P2P storage cloud, shown in Algorithm 2, in order to optimize video availability and service quality, by effectively utilizing premium server bandwidth resources. Specifically, by leveraging online traces on video popularity and redundancy, as well as peer cache status and online duration, the MS in *Novasky* is designed to *proactively* transmit randomly coded segments of under-supplied videos to capable peers, in order to: (1) cope with redundancy loss due to peer churn, (2) alleviate video demand-supply discrepancy, and (3) release new videos. These replenished segments can improve data redundancy and diversity in the P2P storage cloud, so as to improve data sharing among peers, and thus mitigate costly requests to servers. This is complementary to the commonly used "best-effort" server

strategy which is passively driven by unsatisfied viewing requests from peers. In particular, as the MS hosts original videos, it can readily produce new coded segments using our coding approach in previous subsection, without incurring excessive repair bandwidth costs [7].

Algorithm 2 Adaptive server push-to-peer strategy.

1: Given a repository of videos $\mathbf{V} : \{v = 1, 2, \ldots, V\}$ seeded by the MS, monitoring the video popularity λ_v and redundancy $r_v, \forall v \in \mathbf{V}$, by collecting peer viewing statistics and cache status.
2: **for all** $v \in \mathbf{V}$ **do**
3: **if** v has been pushed within a previous period T_p **then**
4: **continue**;
5: **end if**
6: compute a demand-supply discrepancy index $d_v \leftarrow (\frac{kw}{w_v})\frac{r_v}{\lambda_v}$, where w_v is the streaming bit rate of video v, and w is a statistical measurement on peer upload capacity based on collected traces.
7: **end for**
8: select a candidate video v' with $d_{v'} = \min\{d_v | v \in \mathbf{V}\}$.
9: **if** $d_{v'} < d$, which is a tunable threshold controlled by the service provider **then**
10: generate $\lambda_{v'}$ coded segments using Reed-Solomon codes;
11: select candidate peers according to collected traces on available cache space, online duration and idle status;
12: **if** MS has available bandwidth and is not busy **then**
13: transmit new coded segments to selected peers.
14: **end if**
15: **end if**

Our server push-to-peer strategy can be implemented with flexibility in the P2P storage cloud. Specifically, video popularity $\lambda_v, \forall v \in \mathbf{V}$, can be captured by periodically recording file request counts, so as to adapt to the evolution of user interests. With our coding approach in the previous subsection, the video redundancy $r_v, \forall v \in \mathbf{V}$, can be reflected by the ratio of the total number of video segments cached on peers to the corresponding number of segments k. With our peer caching Algorithm 1, a video v with redundancy r_v implies that there exist $[r_v, kr_v]$ seeding peers caching coded or original segments of v. Given the video streaming rate w_v and a statistical measurement on the peer upload capacity w (*e.g.,* average peer upload capacity), the maximum concurrent number of viewing requests that can be supported by seeding peers can be ideally expressed as $(\frac{kw}{w_v})r_v$. Hence, maintaining video availability requires its redundancy r_v to meet its popularity λ_v as $(\frac{kw}{w_v})r_v \geq \lambda_v$, *i.e.,* $r_v \geq \max\{\frac{\lambda_v w_v}{kw}, 1\}$. This qualitatively implies that a higher video popularity and streaming rate requires a higher level of redundancy, whereas coding and peer upload contributions can help relax such a requirement.

Accordingly, we can express r_v as $r_v = d_v(\frac{\lambda_v w_v}{kw})$; alternatively, $d_v = (\frac{kw}{w_v})\frac{r_v}{\lambda_v}$ is referred to as *demand-supply discrepancy index* of video v. In practice, we expect to maintain $d_v \geq 1$ for a baseline of video availability and resilience to peer churns. The larger the value of d_v is, the higher redundancy, availability and churn resilience the video v has. Under-supplied videos with smaller d_v are preferred by the MS for replenishing coded segments. However, if a video has been replenished in a previous period T_p, it will be excluded to avoid aggressive replenishment of a particular video while ignoring others. To prevent server bandwidth abuse, a tunable threshold for demand-supply discrepancy d is controlled by the service provider. Given a flat amount of server resources, the selection of d needs to balance the trade-off between server resource utilization and coding overhead. A higher threshold may aggravate the burden on servers, while a lower one can lead to under-utilized server bandwidth. Finally, there is a substantial margin of flexibility in selecting candidate peers to push coded segments. The rule of thumb is to randomly select long-lived peers with a relatively large available cache space and available upload bandwidth. The cost efficiency of our server push-to-peer strategy in terms of the utility of pushed video segments will be evaluated with real-world measurements in Sec. III.

III. Novasky: Real-World Measurement and Performance Evaluation

A. Deployment Statistics and Trace Collection

Since September 2009, *Novasky*, which is developed for both Linux and Windows platforms, has been deployed and operational in the Tsinghua University campus network. Table I summarizes the deployment statistics of *Novasky*. In particular, *Novasky* is designed towards providing cinematic-quality VoD service. Fig. 3 plots the CDF of streaming bit rates over all videos provided by *Novasky*. The average video bit rate is 1.08 Mbps, and around 25% of all videos (especially 720p format videos) have bit rates higher than 1.5 Mbps. In contrast, the video bit rates in existing P2P VoD systems [2], [3], [8] fall in the range of $400 - 800$ Kbps, and very rarely exceed 1 Mbps. Despite high streaming bit rates, *Novasky* can provide high service quality levels as perceived by users, in terms of fluent playback and short startup and seek latencies, as we will demonstrate in the next subsection.

TABLE I
Deployment Statistics of *Novasky*.

# of users	$10,094$	# of videos	$1,000$
# of user sessions	$47,626$	# of video sessions	$63,445$
Viewing time	$30,288$ hours	Total traffic	17 TB
Measurement period	Feb-Jul, 2010	Trace volume	10 GB

We have implemented detailed instrumentation and measurement mechanisms within each Novaksy client to collect real-world traces over 6 months. Each peer periodically reports its run-time activities and status using UDP to the MC, which is responsible for logging: (1) user-related traces including online times and cache status, CPU and memory consumption, as well as upload and download traffic volumes; (2) media-related traces including video popularity, startup and seek latencies, as well as session durations. Such internal traces characterizing the live behavior of *Novasky* are not only used

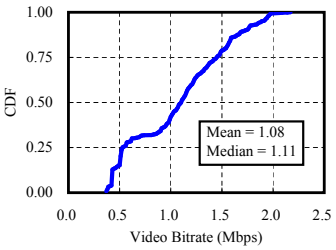


Fig. 3. CDF of streaming bit rates over all videos provided by *Novasky*.

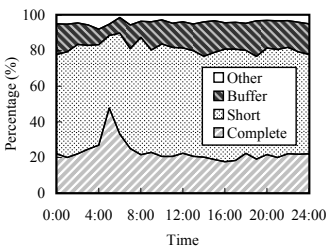


Fig. 4. Percentage of completed, short and aborted sessions, and other cases due to miscellaneous failures.

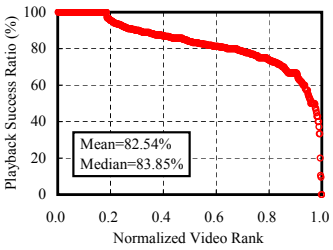


Fig. 5. Playback success ratio of each video in a descending order (normalized) on a representative day.

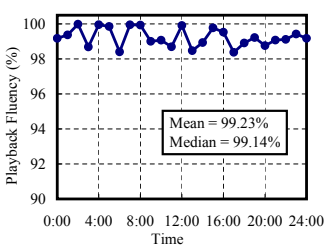


Fig. 6. Playback fluency index averaged over all video sessions vs. time on a representative day.

to guide the adaptive server-feeding strategy in Sec. II-D, but also analyzed to evaluate the design effectiveness and overhead of *Novasky*.

B. User-Perceived Service Qualities

We first evaluate the user-perceived service qualities provided by *Novasky* from various perspectives, including durable video availability, on-demand streaming fluency, as well as startup and random seek latencies.

1) ***Durable video availability****:* In the context of VoD, video availability is primarily related to three cases during a user's viewing session for a video: (1) completed session: the session successfully ends when the video is completely watched by the user; (2) short session: the user stops the session or switches to another video due to content reasons, even if it has not experienced any streaming interruption; (3) aborted session: the session is interrupted with unacceptable buffering time τ, such as $\tau > 1$ minute, which motivates the user to leave. By regarding the first two cases as successful, we define a *playback success ratio* as the percentage of successful sessions over all measured sessions of a video, in order to capture the video availability. Compared to conventional data availability measurements [7], our measurements are more representative of the *durable video availability* in the context of VoD.

Fig. 4 plots the average percentage of completed sessions, short sessions and aborted sessions over all videos in *Novasky* vs. time on a representative day. We observed that the average playback success ratio can be maintained up to 80%, which implies a stable level of durable video availability in *Novasky*. Specifically, 60% of sessions exit prior to the end of the corresponding video due to active video switching, whereas around 20% of sessions last until the videos are completely watched by users. This demonstrates a common "short session" nature in VoD services [2], [8]. Empirically, around 15% of sessions aborted because users cannot tolerate a buffering time exceeding one minute, when streaming interruptions occurred. We zoom into the durable video availability of each video in the large video repository of *Novasky*, by plotting the playback success ratio of each video in a descending order in Fig. 5. It shows that nearly 20% of videos enjoyed perfect availability, and more than 60% of videos achieved higher than 80% durable video availability.

2) ***On-demand streaming fluency and latencies****:* While the durable video availability is insufficient to reflect fine-grained user experiences (*e.g.,* playback jitters), we further investigate the video streaming fluency as sessions progress, as well as startup and random seek latencies. Specifically, for each viewing session s, our trace mechanism records: (1) the startup latency T_{st} between the time a user issues a viewing request and the time the video playback commences; (2) seek latencies T_{sk}^i after a user jumps forward or backward to arbitrary playback points, where i is the number of seek activities during the session; (3) buffering times T_b^j when playback interruptions occur, where j is the number of playback interruptions during the session; (4) the total session duration T_s. Inspired by [2], we define a *playback fluency index* as the fraction of uninterrupted watching time out of the total watching time, *i.e.*, $1-\sum_j T_b^j/(T_s-T_{st}-\sum_i T_{sk}^i)$. Fig. 6 plots the average playback fluency index over all video sessions in *Novasky* vs. time on a representative day. With our lightweight coding implementation, *Novasky* can achieve a superior level of playback fluency that remains higher than 98% and even up to 100%; and the average playback fluency over time is up to 99.23%.

Next we examine the startup and random seek latencies. Fig. 7 plots the average startup and seek latencies, as well as the buffering time during playback interruptions, over all video sessions in *Novasky* vs. time on a representative day. We can observe that: (1) Our coding-aware design can effectively support the timely retrieval of arbitrary playback points, with seek latencies within 3 seconds. This is remarkably shorter than the seek latencies ($10-30$ seconds) of existing P2P VoD systems [2], [3]. In *Novasky*, random seeks with either long or short distances may very possibly not cause the requesting peer to switch to new seeding peers, as both original and coded segments generated by our segmentation and coding approach in Sec. II-C involve data across the entire video. (2) *Novasky* also achieves fast startups to provide competitive user experiences. Specifically, the startup latencies varied between 4 and 9 seconds over time, due to the well known "daily pattern" of peer population [3]. In contrast, the startup latencies in existing P2P VoD systems are up to $15-40$ seconds [2], [3]. (3) The buffering time during playback interruption is around 1 second. Furthermore, Fig. 8 plots the CDF of startup and random seek latencies over all video sessions across our six-month trace period, which confirms that up to 90% of videos enjoyed short latencies within 10 seconds.

C. Effectiveness of Coding and Push-to-Peer

We next examine the effectiveness of our proposed coding-aware storage and replacement mechanism and server push-to-peer strategy, in terms of storage efficiency, peer upload

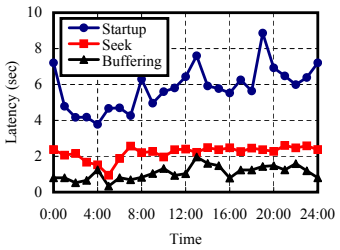


Fig. 7. Latencies of startup, random seeks and buffering averaged over all video sessions vs. time on a representative day.

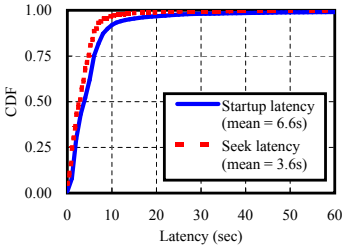


Fig. 8. CDF of startup and random seek latencies of all video sessions over six-month trace period.

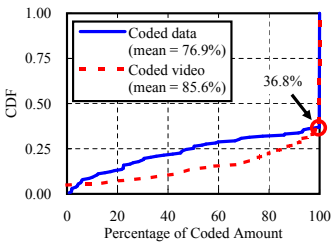


Fig. 9. CDF of the percentage of coded data volume and number of videos over all peers' caches within 48 hours.

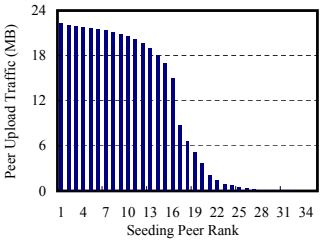
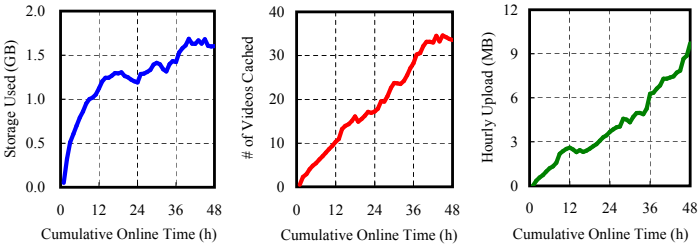


Fig. 10. Upload traffic from seeding peers in descending order, averaged over all video sessions within a representative day.



(a) (b) (c)

Fig. 11. Evolution of (a) storage usage, (b) the number of cached videos, and (c) peer upload contribution averaged over all peers, as cumulative online time increases to 48 hours.

contribution and load balancing, as well as the cost effectiveness of pushed segments.

*1) **Storage efficiency**:* First, we are interested in whether our coding-aware storage algorithm can operate effectively in the P2P VoD storage cloud, by comparing the amount of coded and original data in peer storage caches. Fig. 9 plots the CDF of the percentage of coded data and videos over all peer caches in *Novasky* within 48 hours. The average amount of coded data and videos are 76.9% and 85.6%, respectively; and 63.2% of peer caches are completely filled with coded data. This shows that our coding-aware storage and replacement strategy indeed plays an important role in *Novasky*.

With our coding-aware storage and replacement strategy, is it possible to increase the number of videos stored and peer upload contribution? Fig. 11 shows the evolution of storage usage, the number of cached videos, and peer upload contribution averaged over all peers, as cumulative online time increases to 48 hours. The default peer cache capacity is set to 2 GB in *Novasky*. We observed from Fig. 11(a) and 11(b) that the number of cached videos is not exactly in proportion to the storage space used. During the first 12 hours, both the storage usage and the number of cached videos increase quickly. In the subsequent $13-40$ hours, the increase of storage usage slows down, while the number of cached videos keeps increasing rapidly with the help of our algorithm. Finally, both the storage usage and the number of cached videos reach a saturation point at the 40-th hour, *i.e.*, 1.6 GB storage space and 34 videos, respectively. Fig. 11(c) shows the hourly peer upload traffic averaged over all peers. It is interesting to see that the increase of hourly peer upload contribution is in proportion to that of the number of cached videos, rather than that of storage usage.

This validates the effectiveness of our coded-based storage and replacement strategy in improving the efficiency of utilizing local storage. It "amplifies" the peer cache capacity to store more videos with the same cache capacity, while still maintaining the high availability of coded videos due to the use of Reed-Solomon codes. In addition, this in turn increases data sharing opportunity to enable higher peer upload contribution. Furthermore, we note that the hourly peer upload after 40-th hour keeps increasing even though the number of cached videos already reaches a saturation point. This is achieved by LFU-based cache replacement.

*2) **Load balancing**:* We now examine load balancing in Novaksy, by illustrating the upload traffic distribution across seeding peers (in descending order of traffic) of a requesting peer during a viewing session, in Fig. 10. To obtain a macroscopic view, this is averaged over all video sessions within a representative day. We find that for a viewing session, most traffic is uploaded by the top 16 seeding peers, among which the traffic load is fairly balanced with relatively small deviation. Recalling our choice of the coding parameter $k = 16$ in Sec. II-C, such observation implies that: (1) The data diversity brought by random coding can indeed help *Novasky* to approach load balancing, as coded segments cached on seeding peers are equally useful and independent with high probability. (2) The cases for a viewing peer to switch to new seeding peers, due to the lack of data availability or diversity, are infrequent with our coding-aware storage scheduling, as on average only a small portion of data is shown to be downloaded from peers other than the top 16 ones.

*3) **Cost effectiveness of the server push-to-peer strategy**:* We first examine the video popularity-redundancy status under our server strategy in Algorithm 2. For each video in *Novasky*, Fig. 12 plots the video redundancy in term of the ratio of the total number of video segments cached on peers to the segmentation number of the original video k, vs. video popularity in term of daily request count, periodically sampled over a two-week period from July 2 to July 14, 2010. In comparison, we also plot the expected video redundancy as a function of video popularity using Algorithm 2, and the lower bound of redundancy to maintain video availability by peers alone, as a function of video popularity based on our discussion in Sec. II-D.

We observed that: (1) The redundancy of most videos have reached or surpassed the expected level, while the remaining ones have redundancy at least higher than the lower bound. This implies that our server push-to-peer strategy effectively adapted video redundancy to meet user demand, so as to guarantee video availability and user experience. Even some

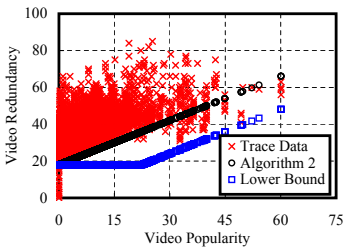


Fig. 12. Video redundancy vs. video popularity under server push strategy, compared to the expectation and lower bound.

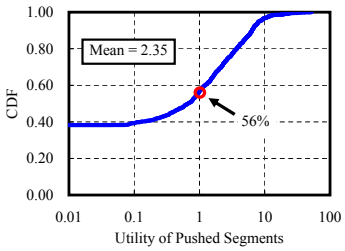


Fig. 13. CDF of the utility of pushed segments from servers, over a two-week period from July 2 to July 14, 2010.

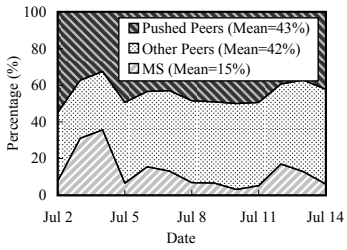


Fig. 14. Percentage of server bandwidth usage and peer bandwidth contributions over all videos in *Novasky*, within a two-week period from July 2 to July 14, 2010.

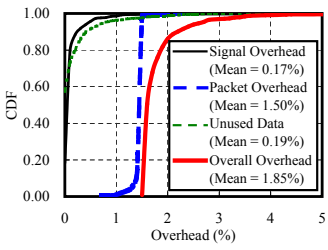


Fig. 15. CDF of the signaling overhead, packet overhead, unused data overhead and overall transmission overhead over all videos.

"cold" videos with a zero request count can remain available with certain nonzero redundancy. (2) There are many less popular videos with redundancy much higher than the expected level. We believe this is due to the dynamic changing of user interests in video content. For example, a highly popular video, which can result in many segments cached by peers, may become "cold" quickly.

Next, we take a closer look at the *utility* of replenished segments pushed by servers in Fig. 13. It plots the CDF of the ratio between the amount of data served by each replenished segment (uploaded to some requesting peers) and the size of the replenished segment pushed by servers, over the two-week period. Note that the utilities of replenished segments are very likely to increase further, as they could be requested by more peers as time progresses. We found that: (1) The average utility of replenished segments is 2.35. Although 56% of them have less than 1.0 utility, we believe they are still desirable to meet the corresponding user demand, which would otherwise be unsatisfied with degraded user experience. (2) Specifically, those segments with utility of 1.0 represent a balanced cost effectiveness between server bandwidth consumed and peer upload contribution afterwards. (3) The utilities of the remaining 44% replenished segments have been remarkably "amplified" with peer assistance. This implies that a substantial amount of server bandwidth costs can be saved by benefiting from our server push-to-peer strategy.

To validate this, Fig. 14 plots the percentage of server bandwidth usage and peer bandwidth contribution over all videos in *Novasky* within the two-week period. Specifically, the latter is further dissected into peers that are caching replenished segments pushed by servers (referred to as pushed peers) and others. It shows that server bandwidth usage only accounted for 10%-35% of the total traffic volume over time, which is remarkably lower than a previous measurement result of 73% in Gridcast [8], and comparable to that of UUSee with network coding [3]. Interestingly, pushed peers can make significant bandwidth contributions up to 43% on average. This clearly shows that our server push-to-peer strategy can indeed lead to effective utilization of server bandwidth, by increasing peer contributions.

D. Overhead with Coding

Finally, we examine the transmission and computation overhead incurred by coding in *Novasky*.

*1) **Transmission overhead**:* There are three types of transmission overhead in *Novasky*: (1) *packet overhead* containing the packet header of around 20 bytes per block transmitted using UDP, a 2-byte segment index, and a block sequence number; (2) *signaling overhead* including the seed list retrieval for bootstrapping, handshake for connection establishment, as well as media and security metadata; (3) *unused data overhead* referred to those downloaded yet unwatched video data, due to user activities such as session abortion, video switching and random seeks. Fig. 15 plots the CDF of all three types of overhead in terms of the percentage of their incurred traffic volume in the total traffic volume of each video, over a one-week period from July 13 to July 20, 2010. We observe that: (1) Thanks to our lightweight coding implementation, the overall transmission overhead of any video is lower than 5%, which is half of the overhead in PPLive [2] and comparable to that of UUSee with network coding [3]. (2) Among the three types of overhead, packet overhead accounts for a relatively larger portion with a mean of 1.5%, while the means of the other two are all lower than 0.2%. In summary, there is marginal transmission overhead incurred by our coding implementation in *Novasky*.

*2) **CPU and memory consumption**:* We now evaluate the computational and memory overhead of *Novasky* with coding "in the wild," by periodically sampling the CPU usage and memory consumption of the *Novasky* client software on each peer. Specifically, the CPU usage is broken down into: the *front-end usage* which is primarily accounted for by the media player to render videos, and the *back-end usage* which is incurred by downloading and decoding video data during a viewing session, or by encoding for storage replacement when the peer cache space is saturated.

Fig. 16 plots the CDF of two categories of CPU usage that are sampled from $1,500$ online peers during a one-week period from July 13 to July 20, 2010. Fig. 17 plots the CDF of memory consumption of the same set of peers. We observe that: (1) Most of the peers have less than 10% back-end CPU usage, and even the highest observed back-end CPU usage is lower than 15%. This shows that the computation overhead incurred by coding is indeed acceptable in practice. (2) By embedding the lightweight open-source VLC player in *Novasky*, the front end CPU usage is typically below 20%. Note that it is not uncommon to observe that the frond-end CPU usage is higher than back-end CPU usage during

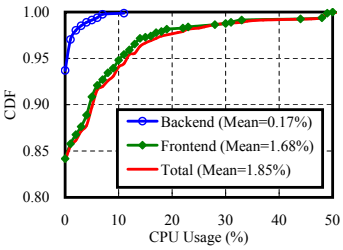


Fig. 16. CDF of the CPU usage of the *Novasky* client software, which is sampled from $1,500$ online peers over an one-week period from July 13 to July 20, 2010.

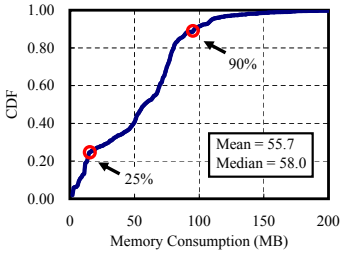


Fig. 17. CDF of the memory consumption of the *Novasky* client software, which is sampled from $1,500$ online peers over an one-week period from July 13 to July 20, 2010.

a viewing session, especially when watching 720p cinematic-quality videos (at around 2 Mbps). (3) Many online peers could have zero back-end CPU usage, when they are not watching or uploading any video. (4) Up to 90% of peers have less than 100 MB memory consumption, which is comparable to that of current systems without coding, such as PPLive. Around 25% of peers have less than 20 MB memory consumption, as most of them just joined the system. In summary, our lightweight coding implementation and fine-tuned parameter selection in *Novasky* work effectively in real-world systems, with marginal transmission and computation overhead brought by coding.

IV. Related Work

There exists a number of prior works on using network coding to improve the availability and performance of P2P distributed storage and file download systems (*e.g.*, [6], [7], [9]). One of the most relevant studies [5] has analytically investigated the use of erasure codes in a P2P storage cloud, by considering data reliability, computation efficiency and security issues. Differing from these works that mainly focused on elastic content distribution or general data availability, *Novasky* constitutes a real-world operational P2P storage cloud, with a coding-aware peer storage solution that is custom-tailored to the on-demand streaming requirement of cinematic-quality videos. More recently, it has been shown by several emulated studies [10], [11] that network coding can help improve playback quality and buffering delays in both P2P live and on-demand streaming systems. While these works have provided key design principles on the use of network coding for transmission designs, we are particularly focused on how to adopt MDS codes, such as Reed-Solomon codes, to improve storage efficiency and video availability in a P2P storage cloud that is being used on a daily basis.

Several measurement studies of P2P VoD systems [2], [8] have extensively covered video characteristics, streaming performance, user behavior, and server load. Our work in Novasky is different from and complementary to these studies, in that we focus on building an operational P2P storage cloud for on-demand streaming of *cinematic-quality* videos over a *high-bandwidth* network. In particular, while existing peer caching designs still resort to replications of original video data, we adopt Reed-Solomon codes, which are tightly coupled with new coding-aware storage and access mechanisms to jointly optimize storage efficiency, video availability and streaming performance. This enables Novasky to provide up to 1,000 DVD-quality or even 720p videos with bit rates of 1 – 2 Mbps — much higher than that of current popular P2P VoD systems over the public Internet.

Finally, *Novasky* has incorporated an adaptive server push strategy which is different from the conventional "best-effort" server strategy in existing P2P VoD systems, and is particularly useful for balancing the demand and supply of content and bandwidth. Suh *et al.* [1] in the literature proposed push-to-peer data placement schemes for VoD in cable networks. However, static set-top boxes in cable networks are radically different from a highly dynamic P2P storage cloud, and the work has focused on theoretical analysis without validating results from a real-world system.

V. Concluding Remarks

This paper designs and implements *Novasky*, an operational P2P storage cloud for on-demand streaming of cinematic-quality videos over a high-bandwidth network. Two unique mechanisms are proposed: (1) a coding-aware peer storage and replacement strategy that takes advantage of Reed-Solomon codes to achieve storage efficiency, durable video availability and load balancing; and (2) an adaptive server push strategy with video popularity-redundancy awareness to proactively balance the demand and supply of content and bandwidth in the P2P storage cloud. Based on real-world traces over 6 months, we demonstrated that *Novasky* can achieve stellar performance, as perceived by its users in the P2P storage cloud. We believe that insights offered in this paper will be helpful towards future designs of P2P storage cloud systems for a variety of purposes.

References

[1] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. Towsley, and M. Varvello, "Push-to-Peer Video-on-Demand System: Design and Evaluation," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, pp. 1706–1716, 2007.

[2] Y. Huang, T. Fu, D. Chiu, J. Lui, and C. Huang, "Challenges, Design and Analysis of a Large-scale P2P-VoD System," in *Proc. of ACM SIGCOMM*, Aug. 2008.

[3] Z. Liu, C. Wu, B. Li, and S. Zhao, "UUSee: Large-Scale Operational On-Demand Streaming with Random Network Coding," in *Proc. of IEEE INFOCOM*, Mar. 2010.

[4] J. Plank, "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-Like Systems," *Software: Practice and Experience*, vol. 27, no. 9, pp. 995–1012, 1997.

[5] J. Li and Q. Huang, "Erasure Resilient Codes in Peer-to-Peer Storage Cloud," in *Proc. of Acoustics, Speech and Signaling Processing*, 2006.

[6] A. Dimakis, P. Godfrey, M. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Systems," in *Proc. of IEEE INFOCOM*, May 2007.

[7] R. Rodrigues and B. Liskov, "High Availability in DHTs: Erasure Coding vs. Replication," *Peer-to-Peer Systems IV*, pp. 226–239, 2005.

[8] B. Cheng, L. Stein, H. Jin, and Z. Zhang, "Towards Cinematic Internet Video-on-Demand," in *Proc. of 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, Apr. 2008.

[9] C. Gkantsidis, J. Miller, and P. Rodriguez, "Comprehensive View of A Live Network Coding P2P System," in *Proc. of ACM IMC*, Oct. 2006.

[10] M. Wang and B. Li, "R^ 2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, p. 1655, 2007.

[11] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez, "Is High-Quality VoD Feasible using P2P Swarming?" in *Proc. of 16th International World Wide Web Conference*, May 2007.