Cinematic-Quality VoD in a P2P Storage Cloud: Design, Implementation and Measurements

Fangming Liu, *Member, IEEE,* Shijun Shen, Bo Li, *Fellow, IEEE,* Baochun Li, *Senior Member, IEEE,* and Hai Jin, *Senior Member, IEEE*

***Abstract*—In this paper, we explore the design space and practice of a new peer-to-peer (P2P) storage cloud, which is capable of replicating, refreshing and on-demand streaming of cinematic-quality video streams, in a decentralized fashion using local storage spaces of end users. We identify key design challenges and tradeoffs in such a P2P storage cloud, and how these are addressed by making informed design choices in a step-by-step fashion. Following our design choices, we have implemented a real-world Video-on-Demand (VoD) system with over 100,000 lines of code, called *Novasky*, which features new coding-aware peer storage replacement and server push-to-peer strategies, in order to maintain media availability and to balance the system-wide supply-demand relationship in the P2P storage cloud. Since September 2009, it has been deployed in the Tsinghua University campus network, attracting 10,000 users during our measurement studies from February to July 2010, 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, while maintaining reasonable operational overhead and server bandwidth costs. Our general understanding on the design tradeoffs of P2P storage cloud and practical experiences with *Novasky* may bring valuable guidelines to future designs of production-quality P2P storage cloud systems.**

***Index Terms*—Video-on-demand, peer-to-peer, storage replication, Reed-Solomon codes, quality of experience.**

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 [1], 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.,* [2]), simulations (*e.g.,* [3]), and measurement perspectives (*e.g.,* [4]). More recently, the ever increasing expectation of Internet users worldwide for cinematic-quality videos watched on bigger screens, such as online high-definition TV or movies in $1,920 \times 1,080$ pixel resolution [5], aggravates the challenges in guaranteeing the *availability of both popular and unpopular videos* and sustaining *high streaming bit rates*, over highly dynamic P2P networks.

To meet skyrocketing user requirements on both video quantity and quality [6], such systems usually require each user, called a *peer*, to dedicate a certain amount of 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 [7], [8]. The benefits brought forth by such local caching are two-fold: to assist the video content distribution among other users from the cache in a peer-to-peer fashion, and to improve the quality of user experiences with respect to important quality metrics, such as random seek latencies — the time it takes for any random seek operations to complete. As 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, how can system designers pool them together to form a *P2P storage cloud* custom-tailored for replicating, refreshing and on-demand streaming of cinematic-quality video contents? This is arguably believed to be one of the most critical design problems from the viewpoint of commercial VoD providers such as PPLive, which showed that a proper design choice of content replication and replacement strategies across peer caches has brought significant performance improvements [9].

In this paper, we seek to explore the design space and practice of a new P2P storage cloud from the ground up, that maximizes the utilities of cached content and contributions of peers to satisfy the availability and performance needs of cinematic-quality VoD, without the penalty of excessive server bandwidth costs. Our original contributions in this paper are two-fold:

First, designing peer storage and server bandwidth strategies by exploring representative design choices. On the peer side, we make our design choices between simple replication strategies commonly used in peer caches of most existing commercial systems, and a new coding-aware peer storage replacement strategy that we proposed to take advantage of theoretically-sound maximum distance separable (MDS) codes, such as Reed-Solomon codes [10]. Impartial trace-

Manuscript received February 28, 2012; revised June 1, 2012. The research was support in part by a grant from The National Natural Science Foundation of China (NSFC) under grant No.61103176, by a grant from the NSFC Key Program under grant No.61133006, by a grant from the Research Fund of Young Scholars for the Doctoral Program of Higher Education, Ministry of Education, China, under grant No.20110142120079, by a grant from NSFC/RGC under the contract N_HKUST610/11, by a grant from ChinaCache Corp. under the contract CCNT12EG01.

F. Liu and H. Jin are with the Services Computing Technology and System Lab, Cluster and Grid Computing Lab in the School of Computer Science and Technology, Huazhong University of Science and Technology. E-mail: {fmliu, hjin}@hust.edu.cn.

S. Shen is with National Computer Network Emergency Response Technical Team/Coordination Center of China. E-mail: shijun0504@gmail.com.

Bo Li is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: bli@cse.ust.hk.

Baochun Li is with the Department of Electrical and Computer Engineering, University of Toronto. E-mail: bli@eecg.toronto.edu.

Digital Object Identifier XXX/JSAC.XXX.XXX

driven performance comparisons are conducted to understand and test their suitability for maintaining a balanced tradeoff between the performance of streaming "hot" videos and the diversity of available videos. On the server side, to accommodate frequent peer departures and the evolution of diverse user interests across rich video contents, we switch between two extremes of the server bandwidth strategies — including the conventional passive mode driven by greedy viewing requests of unsatisfied peers, and an adaptive server push mode that we designed to proactively transmit content to peers in the P2P storage cloud. Trace-driven experiments in a wide range of settings are carried out to evaluate these strategies in maintaining a cost-effective relationship between the system-wide "supply" of and "demand" for content and bandwidth.

Second, building and measuring a real-world cinematic-quality VoD system. Following our design choices, we are able to implement, deploy, and measure a complete VoD system based on a P2P storage cloud over a high-bandwidth network, namely, *Novasky* [11]. Rather than typical emulation-based experiments, *Novasky* is a production-quality VoD system that we implemented with over 100,000 lines of code (LOC) in C++ from scratch to incorporate our proposed coding-aware peer storage replacement and proactive server push-to-peer strategy. We are pleasantly surprised by (and advertising the product for) the fact that, during our measurement study from February to July, 2010, its production deployment in the campus network at Tsinghua University has delivered over 1,000 video streams to over 10,000 users at a quality level of 720p or even 1080p, streaming at bit rates of 1 – 2 Mbps. In contrast, most traditional P2P VoD systems operate at low bit rates, such as 400 Kbps [9]. 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 [9].

Through extensive measurements, we validate the scale, quality, and lessons learned from practically every conceivable aspect in *Novasky*. In particular, 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 [12], [13], they have not been used in practice in any production-quality P2P storage cloud systems. Our measurements take a particular focus on the computational complexity and operational overhead of applying Reed-Solomon codes in the *Novasky* P2P storage cloud. To the best of our knowledge, our work represents the first attempt to provide not only a general understanding of peer-side and server-side design spaces, but also practical implementation guidelines of an operational P2P storage cloud towards a new genre of cinematic-quality VoD services.

II. Design Challenges and Methodologies

A. Design Objective and Challenges

In the context of VoD, a P2P storage cloud is designed to "pool" local cache storage spaces on all participating peers via an overlay network, and collectively maintain the availability and delivery of a rich repository of video contents. To compensate for the dynamic nature of distributed peers, a set of servers can be deployed to cooperate with the P2P storage cloud to optimize video availability and streaming quality, while utilizing limited and prohibitive server bandwidth resources in a cost-effective manner. In this section, we first identify both peer-side and server-side challenges in the design process of P2P storage cloud for on-demand streaming of cinematic-quality videos from the ground up:

▷ Despite a variety of cache replacement strategies in existing P2P VoD systems [4], [7]–[9], 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 simple replication strategies could become insufficient to maintain data availability and diversity, and are especially ill-conceived for on-demand streaming of cinematic-quality videos. 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 the availability of videos.

▷ 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 need 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 server(s).

B. Methodologies

To address the above challenges, we need to thoroughly explore different design choices of peer storage and server bandwidth strategies, verified from simulation-based "stress-tests" to practical system implementation "in the wild." In particular, such design choices are required to withstand impartial and extensive performance-comparisons *in a step-by-step manner*, in order to derive crystal-clear and rigorous engineering guidelines on *how well a certain building component of the system is doing and why*.

First, we build impartial and practical simulations to conduct fair comparisons between different design choices, driven by real-world traces from a popular VoD service of cctv.com [14], [15]: (1) Without loss of generality, suppose media server(s) host a rich set of $V = 1,000$ high-quality videos of 1 GB size and 1 Mbps streaming bit rate, which follow the observed long-tail popularity distribution of real traces [15]. (2) Empirically, the typical uplink and downlink capacities of peers are 384 Kbps and 1.5 Mbps, respectively, which is in accordance to the asymmetric characteristics of ADSL access in the public Internet. Like most existing commercial P2P VoD systems [9], the default cache space of peers is limited to 2 GB. The arrival/departure pattern and viewing interactivities of peers follow the reported statistics of user behaviors[1] of real traces [15]. (3) To exercise our

[1]Our empirical observations from both cctv.com traces and our realistic *Novasky* VoD system in Sec. V-A show that, the number of seeding peers (supply) is usually greater than the number of streaming peers (demand).

design choices to the extent possible, we emulate both stable and dynamic scenarios for performance evaluation throughout Sec. III and Sec. IV, by varying the system scale M (the total number of concurrently online peers) and the degree of system dynamics m in term of the portion of newly arriving peers (with empty caches and thus cannot make upload contributions to the system immediately) over the entire population M. Note that such flexible and fine-tuned "stress-tests" are imperative before, yet infeasible during, live operation of deployed systems.

Then, following the justified design choices, we are able to implement and deploy a new cinematic-quality VoD system based on a P2P storage cloud in the real world, with comprehensive measurements to validate its performance and overhead in Sec. V. Such a "closed-loop" research methodology can offer not only general understanding of peer-/server-side design spaces, but also practical implementation guidelines towards the best possible VoD service qualities.

III. Design Choices between Replication and Coding-Aware Peer Storage Strategies

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, two representative types of storage strategies characterize the availability-complexity tradeoff in system design:

First, most real-world systems organize peers in a mesh overlay [4], [9], wherein each peer replicates 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. Meanwhile, a viewing peer is fed by multiple seeding peers with limited upload capacities, each of which is responsible for uploading only a certain part of video data in a coordinated manner. Combining these aspects, the common design choice of peer caches is to *replicate certain segments of the original video data* instead of the complete video. Such a mesh-based replication strategy (*e.g.,* [7], [8]) is widely adopted due to its simplicity in system design and resilience to peer dynamics. However, as different segments become rare (even unavailable) or excessively duplicated among peer caches, video availability and on-demand streaming performance are adversely affected.

Second, to improve video availability for the same storage capacity, another theoretically-sound strategy is to take advantage of coding, such as MDS codes [10], [16], to generate and store video data in the form of encoded segments across the distributed peer caches. Compared to the replication of original video segments, all coded segments are equally innovative and useful to any peers with high probability, due to increased data randomness and diversity (Sec. III-A2). Nevertheless, such *coding-aware storage strategies* have rarely been deployed in peer caches of real-world operational VoD systems. How much performance benefit could sophisticated coding-aware storage strategies bring forth compared to simple replication strategies, in the context of on-demand streaming of cinematic-quality videos? Would such benefit outweigh the additional system complexity and computation/communication overhead in practical uses of coding?

A. Peer Storage and Replacement using Reed-Solomon Codes

In response to the questions above, we first propose a coding-aware peer storage and replacement strategy using Reed-Solomon codes, which then is experimentally compared with simple replication strategies under impartial trace-driven simulations.

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

```
1: // when the current cache space is saturated and cannot store new incoming video data
2: sort currently cached original videos in LFU order;
3: for each original video do
4:    // code-based replacement
5:    use Reed-Solomon codes to generate a coded segment to be stored, with a randomly selected index from the Vandermonde matrix;
6:    evict the original video from the cache;
7:    if the cache space is enough for the new incoming data then
8:       store the new incoming data; return;
9:    end if
10: end for
11: // if cache space is still insufficient to accommodate new incoming video data
12: sort currently cached coded segments in LFU order;
13: for each coded segment do
14:    evict the coded segment from the cache;
15:    if the cache space is enough for the new incoming data then
16:       store the new incoming data; return;
17:    end if
18: end for
```

As a peer watches more videos (or receives pushed video segments from proactive servers as we will elaborate in Sec. IV), new incoming videos will be stored until its cache space gradually becomes fully utilized. This will eventually trigger our replacement strategy in Algorithm 1 to replace original video segments with *one* coded segment, so that a peer can reclaim cache space to further accommodate more incoming videos. In particular, to combine coding with replication within the peer storage cache, our strategy 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[2], and following that order, each video file will be coded to *one*

[2]Specifically, each peer records the number of requesting times (from other peers) for each of its cached videos. Such statistics will be used by corresponding peers to form their respective LFU order of cached videos as needed in the replacement strategy.

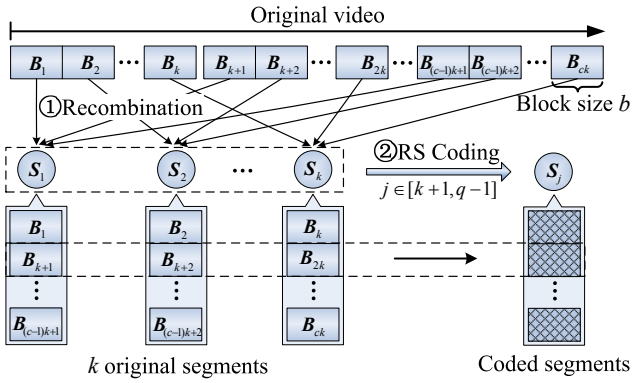


Fig. 1. The coding process from original video to coded segments.

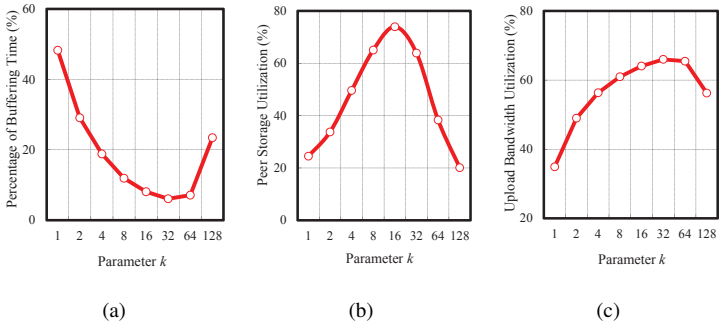


Fig. 2. Effects of number of original segments to be coded k: (a) percentage of buffering time during playback, (b) storage utilization ratio of peer caches, and (c) upload bandwidth utilization ratio averaged over all peers, as the value of parameter k varies.

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.

*1) **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 general, the original (complete) video of size L is divided into contiguous data blocks[3] 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 $\mathbf{S}_j = \{B_{ik+j} | i = 0, 1, \ldots, c-1; j = 1, 2, \ldots, k\}$ with indices $j \in [1, k]$, so that each segment consists of interleaved blocks across the entire video[4], as shown in Fig. 1. A seeding peer can independently and randomly choose 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 $\widehat{\mathbf{S}_j} = g_j [\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_k]^T$. 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.

*2) **Choosing parameters** n **and** k **in Reed-Solomon codes**:* How parameters n and k should be chosen hinges upon the need that different peers need 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 our storage strategy. 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 [10], [12]. In this case, based on the requirement of Reed-Solomon codes, $n \leq q - 1$, and the maximum n is $2^{16} - 1$.

By utilizing our trace-driven simulations to obtain practical guidelines on the choices of k, Fig. 2(a), Fig. 2(b) and Fig. 2(c) investigate the fraction of buffering time (streaming interruptions) during video playback, the storage utilization ratio of peer caches, and the upload bandwidth utilization ratio (versus peer upload capacity) averaged over all peers, respectively. As the value of k varies, we observe "sweet spots" of $k \in [16, 64]$ for achieving less buffering time when playback interruptions occur and higher utilization of peer upload bandwidth resources, and $k \in [8, 32]$ for reaching higher utilization of peer storage resources. These jointly give a favorable range of $k \in [16, 32]$ for system designers. To minimize the probability of duplicate index choices by different peers, one representative choice is a reasonably small $k = 16$ as we will verify in our real-world system implementation and measurement in Sec. V. 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

[3]The selection of block size b needs to cater to fine-grained decoding and streaming, while avoiding excessive computation overhead. It is empirically shown in [17] that feasible block sizes for code-based P2P content distribution systems fall in [2, 32] KB. Specifically, under a typical setting of $b = 8$ KB (smaller than 32 KB in BitTorrent-like file sharing systems) and $k = 16$ in *Novasky*, an amount of $k * b$ decoded data corresponds to one second of video content with a streaming bit rate of 1 Mbps.

[4]As both original and coded segments generated by our segmentation and coding approach involve data across the entire video, random seeks with either long or short distances may very possibly not cause the requesting peer to switch to new seeding peers. This can avoid potential increases of random seek latencies.

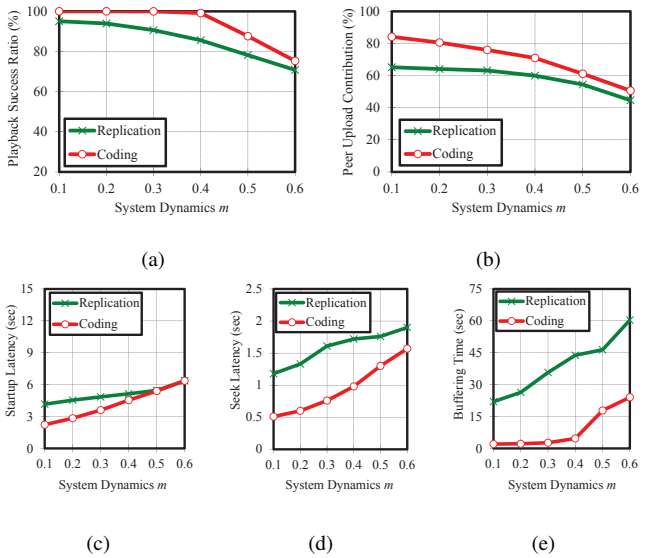


Fig. 3. Performance comparison of coding-aware and traditional replication strategies under different degree of system dynamics: (a) playback success ratio of videos, (b) upload contribution of peers, (c) startup latency, (d) seek latency, and (e) buffering time averaged over all video sessions.

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 and analytical studies [9], [18], the practical number of seeding peers falls in $[8, 32]$.

*3) **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. We will further verify the coding overheads via real-world measurements in Sec. V-D.

B. Coding-Aware or Not?

Following our methodology in Sec. II-B, we conduct impartial trace-driven performance comparisons of coding-aware and traditional replication strategies in the context of VoD. Specifically, we choose the simple yet practical proportional replication algorithm [7], [9] with respect to video popularity to represent the family of traditional replication strategies in most existing P2P VoD systems. And recent analyses [7], [19], [20] have demonstrated that such simple replication strategies can work well.

First, we examine the video availability which 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. Fig. 3(a) compares the average playback success ratio of videos under coding-aware peer storage strategy in Algorithm 1 (with the chosen parameters in Sec. III-A2 and Sec. II-B) against traditional replication strategy. Under small and moderate degrees of system dynamics ($M = 3,000$ and $m \leq 0.4$, *i.e.,* the portion of newly arriving peers with empty caches is below 40%), our coding-aware peer storage strategy maintains a consistently higher playback success ratio (nearly 100%) than the traditional replication strategy. As the system becomes even more dynamic as $m > 0.4$, it is not surprising that the performance of both coding-aware and traditional replication strategies degrades, yet the former still outperforms the latter.

Given a limited amount of server bandwidth 50 Mbps provisioned versus a large system scale $M = 3,000$ (*i.e.,* only 50 peers can be concurrently supported by the server to playback high-quality videos of 1 Mbps bit rate), Fig. 3(b) examines the ratio of the amount of video data uploaded by peers to the overall upload traffic served by both the server and peers. It reveals that our coding-aware peer storage strategy can enable higher data sharing opportunities among peer caches, and make more sufficient utilization of peer upload bandwidth resources. This explains the improvement of video availability observed in Fig. 3(a).

Next, we estimate the user experience in terms of the average startup latency, seek latency and buffering time over all video sessions in Fig. 3(c), Fig. 3(d) and Fig. 3(e), respectively. We observe that our coding-aware peer storage strategy is capable of reducing all the three types of latencies under both stable and dynamic systems. In particular, the average buffering time under the traditional replication strategy steadily increases from around 22 seconds to even one minute as m increases. This leads to unacceptable playback interruptions for users. In sharp contrast, such a buffering time under our coding-aware strategy can be kept within 5 seconds for most cases (*i.e.,* $m \leq 0.4$), and less than 25 seconds even under highly dynamic systems such as $m = 0.6$.

In 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 [12].

IV. Design Choices between Proactive Server Push-to-Peer and Passive Server Strategies

Due to the dynamic nature of the P2P storage cloud, video availability and peer upload contribution can be degraded at run-time, as illustrated by Fig. 3. 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 [13]. 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 [16]. 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, it is natural to consider what is the best possible strategy for server bandwidth to be supplemented and its usage to be reduced. Generally, two extremes of the design space exist: (1) Traditionally, server bandwidth usage is *passively* driven by viewing requests that are not satisfied by peers alone, in order to maintain a baseline on content availability. This is commonly used in existing commercial systems [9], [21], due to its simplicity. Although this can temporarily compensate for *individual* demand of peers, it cannot balance the *system-wide* demand and supply of P2P storage cloud *in the long run*. As a result, expensive server bandwidth could be greedily consumed during peak viewing hours, while leaving idle server resources during non-peak hours (Fig. 6). (2) Beyond such passive server strategies, can we switch to *proactively* push "proper" video data to "proper" peer caches at "proper" times, with a better utilization of precious server resources? The expected benefit is to maximize the utilities of pushed data and contributions of peers, enabling the P2P storage cloud to satisfy most viewing requests and offload the server(s) eventually.

A. Adaptive Server using Push-to-Peer Strategy

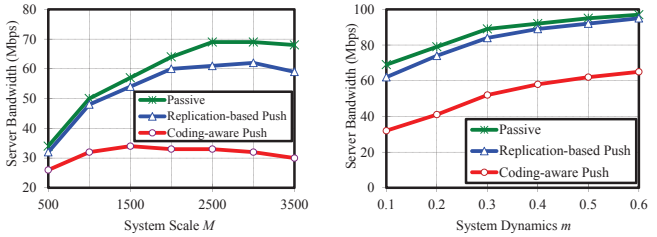
To answer the above questions, we first design an adaptive server push-to-peer strategy in the P2P storage cloud, shown in Algorithm 2, which is then impartially compared with traditional passive server strategies. Specifically, by monitoring online traces on video popularity and redundancy, as well as peer cache status and online duration, the media server(s) feeding the P2P storage cloud can be 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 server(s) host original videos, it can readily produce new coded segments using our coding approach in previous section, without incurring excessive repair bandwidth costs [16].

Algorithm 2 Adaptive server push-to-peer strategy.

1: Given a repository of videos $\mathbf{V} : \{v = 1, 2, \ldots, V\}$ seeded by the media server(s), 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** media server(s) 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 section, 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*



(a) (b)

Fig. 4. Comparison of server bandwidth demanded for maintaining 99% of average playback success ratio, using proactive server push-to-peer strategy and traditional passive server strategy driven by unsatisfied viewing requests from peers: (a) under different system scales, and (b) under different degree of system dynamics.

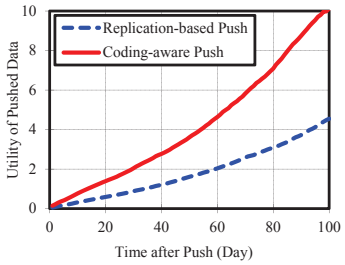


Fig. 5. Comparison of the utility of pushed data from server(s) over time, under replication-based and coding-aware server push-to-peer strategies.

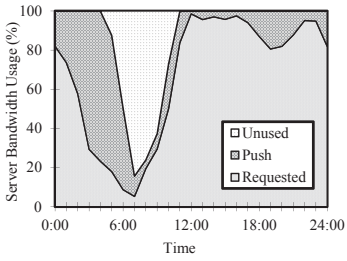


Fig. 6. Percentage of server bandwidth used for meeting viewing requests from peers and pushing coded segments to selected peers over time.

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 server 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. If the cache space of a selected peer is adequate to accommodate those pushed segments, then those pushed segments can be stored by the selected peer without triggering Algorithm 1 to replace other existing segments. Otherwise, when the cache space of a selected peer is insufficient to accommodate those pushed segments, Algorithm 1 will be triggered to replace some existing segments. The cost efficiency of our server push-to-peer strategy in terms of the utility of pushed video segments will be evaluated with both trace-driven simulations in Sec. IV-B and real-world measurements in Sec. V.

B. Proactive or Passive?

Using our trace-driven simulations for impartial tests, Fig. 4(a) compares the server bandwidth demanded for maintaining 99% of average playback success ratio, under proactive server push-to-peer strategy in Algorithm 2 and traditional passive server strategy. Specifically, the proactive server strategy can either push original video segments (henceforth referred to as *replication-based push*) under traditional replication strategy on peer caches, or push randomly coded segments (henceforth referred to as *coding-aware push*) under our coding-aware peer storage strategy. In contrast, the passive server strategy (without push) is completely driven by unsatisfied viewing requests from peers using replication strategy, which is representative of most existing deployed systems [9]. We observe that the passive server strategy greedily consumes tremendous server bandwidth resources as the system scale increases from $M = 500$ to $M = 3,500$, when the degree of system dynamics is set as $m = 0.1$ according to our empirical experiences from practical systems (Sec. V) and the other parameters follow Sec. II-B and Sec. III-A2. While the replication-based push strategy brings limited server bandwidth reduction, the coding-aware push strategy can significantly save around 55% of server bandwidth costs (compared to that of the passive strategy) as the system scale $M \geq 2,500$.

Even when the system becomes more dynamic (*i.e.,* $M = 3,000$ and m increases from 0.1 to 0.6) in Fig. 4(b), our coding-aware push strategy consistently consumes far less server bandwidth than both the passive strategy and the replication-based push strategy do. This further justifies the robustness of our coding-aware server push-to-peer strategy in conserving premium server bandwidth costs, without sacrificing the quality of user experiences.

To unveil in-depth rationales behind the above observations, Fig. 5 zooms into the *utility* of replenished segments pushed by server(s), in term of the ratio between the aggregate amount of data served by all the replenished segments (uploaded to some requesting peers) and the sum of the original sizes of the replenished segments. A good news for system designers is that the utilities of replenished segments can be "amplified" dramatically, as they could be requested by more peers as time progresses. Conversely, severe system dynamics could result in less time and chance for the peers caching replenished segments to make contributions. This may lead to the rise of server bandwidth under all the three types of server strategies in Fig. 4(b). Nevertheless, the utility achieved by our coding-aware push strategy surpasses that of the replication-based push strategy by around 2.3 times, due to the wide diversity and high randomness of coded segments in maximizing peer contributions. Thus, *it is more cost effective to push coded segments rather than original segments.*

Further, Fig. 6 examines the evolution of server bandwidth usages over a day, when the system scale $M = 3,000$ and system dynamics $m = 0.1$. Similar to the empirical observations in most existing VoD systems [4], [9], we observe a regular "daily variation pattern" of server bandwidth usage: the system has the lowest server bandwidth demand from fewer peers at the early morning, and reaches its peak bandwidth usage with an increasing population of peers during the afternoon and night. Given that an Internet service provider (or a content delivery network) often uses the 95 *percentile rule* [3]

to charge a VoD provider according to its peak bandwidth usage, the larger such a variation of bandwidth usage is, the greater amount of under-utilized server bandwidth resources exist. This implies a poor return-on-investment for a VoD provider. Using our proactive server push-to-peer strategy, a VoD provider can not only meet the *present* viewing requests from peers, but also make better use of idle bandwidth to optimize the *long-term* utility of P2P storage cloud (Fig. 5), without incurring extra server bandwidth cost under the 95 percentile rule. In essence, the idle server bandwidth resources are utilized to push coded segments into the P2P storage cloud with "amplified" utilities (Fig. 5), which in turn can alleviate the peak user demand and offload the server(s). Unsurprisingly, the server bandwidth usage still drops during 4:00 am to 7:00 am, due to the difficulty in finding enough number of candidate online peers to push video data to.

V. *Novasky*: Real-World Implementation and Measurement

A. System Deployment and Trace Collection

Motivated by the observed benefits and design guidelines of using coding-aware peer storage strategy and proactive server push-to-peer strategy, we have developed a new cinematic-quality VoD system, named *Novasky* [11], for both Linux and Windows platforms. It has been deployed and operational in the Tsinghua University campus network since September 2009[5]. Fig. 7 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 have elaborated in Sec. IV, 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 derived from traditional P2P designs: all participating peers are organized into a mesh topology, and exchange information about the availability of video segments periodically. In addition to on-demand streaming and random seek functions, *Novasky* provides a feature-rich user interface 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.

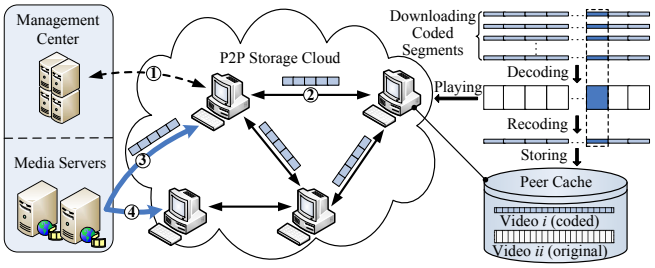


Fig. 7. The architectural implementation of *Novasky*.

▷ We have implemented detailed 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 playback durations. Such internal traces characterizing the live behavior of *Novasky* are not only used to guide the adaptive server push strategy in Sec. IV, but also analyzed to evaluate the design effectiveness and overhead of *Novasky*.

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

Table I summarizes the deployment statistics[6] of *Novasky*. In particular, *Novasky* is designed towards providing cinematic-quality VoD service. Fig. 8 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 [4], [9], [22] 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.

[5]Note that although our system is deployed over a campus network, the effectiveness and advantages of our design choices are not restricted to such cases, as verified by our impartial trace-driven simulations with public Internet settings for fair performance comparisons in previous sections.

[6]A user session is defined as the online period between a pair of arrival and departure events of a user in *Novasky* VoD system, during which the user may playback a certain number of videos (defined as video sessions). On average, the ratio of playback time of a user to its total online time is below $1/3.5$ in *Novasky* VoD system, which implies that there are usually more seeding peers (supply) than streaming ones (demand).

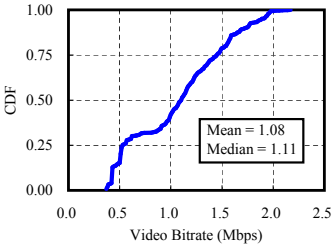


Fig. 8. CDF of streaming bit rates over all videos provided by *Novasky*.

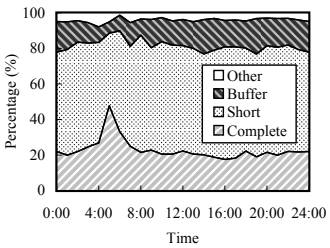


Fig. 9. Percentage of completed, short and aborted sessions, and other cases due to miscellaneous failures.

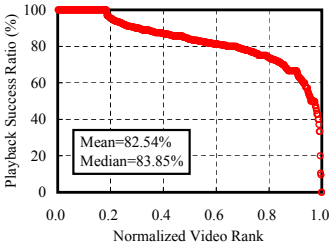


Fig. 10. Playback success ratio of each video in a descending order (normalized) on a representative day.

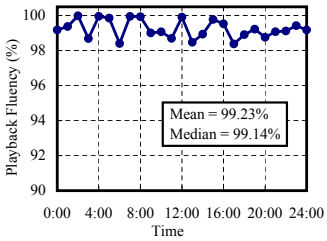


Fig. 11. Playback fluency index averaged over all video sessions vs. time on a representative day.

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****:* Compared to conventional data availability measurements [16], we measure the playback success ratio of videos defined in Sec. III-B, which is more representative of the *durable video availability* in the context of VoD. Fig. 9 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 [9], [22]. 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. 10. 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[7] T_{st} between the time a user issues a viewing request and the time the video playback commences; (2) seek latencies[8] 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 [9], 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. 11 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. 12 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 [4], [9]. 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. III 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 [4]. In contrast, the startup latencies in existing P2P VoD systems are up to $15-40$ seconds [4], [9]. (3) The buffering time during playback interruption is around 1 second. Furthermore, Fig. 13 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 contribution, and 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. 14 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

[7]The startup latency of a viewing peer comes from three aspects, including (1) bootstrapping for retrieving the seed list and media metadata from management server(s), (2) establishing connections with seeding peers, and (3) downloading initial video data from multiple seeding peers to the playback buffer. Specifically, an amount of data corresponding to at least 16 seconds of a video needs to be buffered before a peer plays the video. Suppose a streaming bit rate of 1 Mbps, the size of such buffered data is 2 MB.

[8]The seek latency of a viewing peer is primarily attributed to the above third part (playback buffer), so that it is usually shorter than the startup latency as shown by our measurement results in Fig. 12 and Fig. 13.

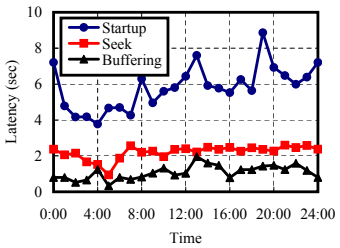


Fig. 12. Latencies of startup, random seeks and buffering averaged over all video sessions vs. time on a representative day.

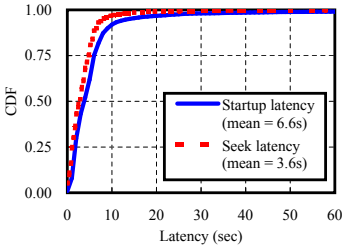


Fig. 13. CDF of startup and random seek latencies of all video sessions over six-month trace period.

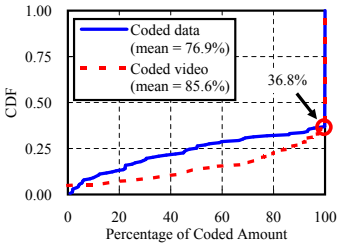


Fig. 14. CDF of the percentage of coded data volume and number of videos over all peers' caches within 48 hours.

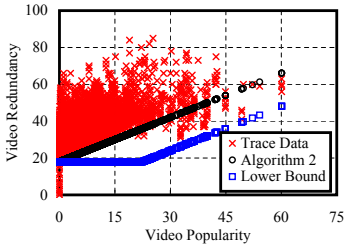
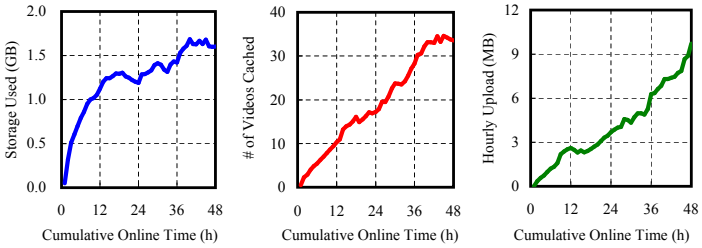


Fig. 15. Video redundancy vs. video popularity under server push strategy, compared to the expectation and lower bound.

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. 16 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. 16(a) and 16(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. 16(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.



(a) (b) (c)

Fig. 16. 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.

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) **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. 15 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. IV.

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 "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, Fig. 17 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 [22], and comparable to that of UUSee with network coding [4]. 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

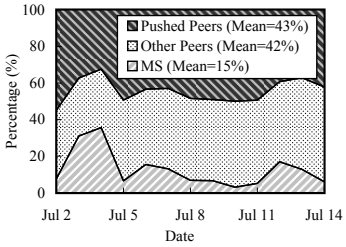


Fig. 17. Percentage of server bandwidth usage and peer bandwidth contributions over all videos in *Novasky*, within a two-week period from July 2 to July 14, 2010.

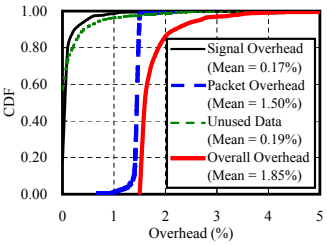


Fig. 18. CDF of the signaling overhead, packet overhead, unused data overhead and overall transmission overhead over all videos.

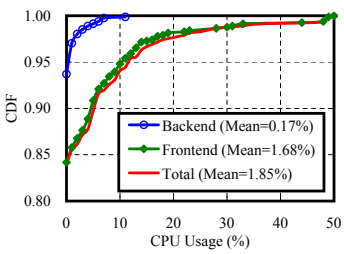


Fig. 19. CDF of the CPU usage of the *Novasky* client software, which is sampled from $1,500$ online peers over an one-week period from July 13 to July 20, 2010.

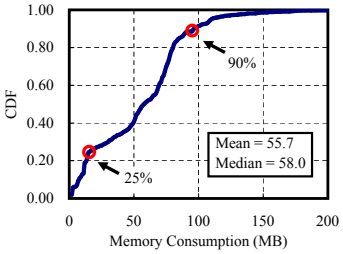


Fig. 20. CDF of the memory consumption of the *Novasky* client software, which is sampled from $1,500$ online peers over an one-week period from July 13 to July 20, 2010.

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. 18 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 [9] and comparable to that of UUSee with network coding [4]. (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. 19 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. 20 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 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.

VI. 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.,* [13], [16], [23]). One of the most relevant studies [12] 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, we constitute a coding-aware P2P storage solution that is custom-tailored to the on-demand streaming requirement of cinematic-quality videos, verified by not only impartial simulation-based comparative-tests but also a real-world system implementation. More recently, it has been shown by several emulated studies [24], [25] 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. Also, this is different from and orthogonal to other prior systems that apply scalable coding [26] for generating multiple layers of video streams to adapt to heterogeneous quality-of-services [27].

Differing from the conventional "best-effort" server strategy in existing P2P VoD systems [9], [22], we have devised, implemented and evaluated a new adaptive server push strategy, which is particularly useful for proactively balancing the system-wide demand for and supply of content and bandwidth in the P2P storage cloud. Suh *et al.* [2] 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.

Finally, several measurement studies of P2P VoD systems [4], [6], [9], [22] have extensively covered video characteristics, streaming performance, user behavior and engagement, as well as server load. As existing peer caching designs in such systems still resort to replications of original video data (*e.g.,* [7]–[9], [28]), our work is different from and

complementary to these studies: our measurements have a particular focus on the effectiveness and overhead of coding-aware storage strategy and proactive server push strategy incorporated into an operational P2P storage cloud, and closely examine how well the storage efficiency, video availability and streaming performance can be jointly optimized, without the penalty of excessive server bandwidth costs.

VII. Concluding Remarks

This paper explores the design space and practice of a new P2P storage cloud for on-demand streaming of cinematic-quality videos. By impartially understanding and testing representative design choices of peer storage and server bandwidth strategies, two unique mechanisms are proposed and justified from the ground up: (1) a coding-aware peer storage and replacement strategy that takes advantage of Reed-Solomon codes to achieve storage efficiency and durable video availability; and (2) an adaptive server push-to-peer strategy with video popularity-redundancy awareness to proactively balance the demand for and supply of content and bandwidth in the P2P storage cloud. This enables us to develop and deploy *Novasky*, a real-world cinematic-quality VoD system based on a P2P storage cloud over a high-bandwidth network. Through extensive measurement studies over 6 months, we demonstrated that our design choices applied in *Novasky* can achieve stellar performance and reasonable overhead, as perceived by its users in the P2P storage cloud. Our insights into not only the general design tradeoffs of P2P storage cloud but also the practical implementation experiences will be helpful to guide future designs of P2P storage cloud systems for a variety of purposes.

References

[1] Y. Liu, Y. Guo, and C. Liang, "A Survey on Peer-to-Peer Video Streaming Systems," *Peer-to-Peer Networking and Applications*, vol. 1, no. 1, pp. 18–28, 2008.

[2] 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.

[3] C. Huang, J. Li, and K. Ross, "Can Internet Video-on-Demand be Profitable?" in *Proc. of ACM SIGCOMM*, Aug. 2007.

[4] 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.

[5] [Online]. Available: http://www.webtvwire.com/category/internet-hdtv

[6] F. Dobrian, A. Awan, D. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang, "Understanding the Impact of Video Quality on User Engagement," in *Proc. of ACM SIGCOMM*, Aug. 2011.

[7] Y. Zhou, T. Fu, and D. Chiu, "A Unifying Model and Analysis of P2P VoD Replication and Scheduling," in *Proc. of INFOCOM*, Mar. 2012.

[8] W. Wu and J. Lui, "Exploring the Optimal Replication Strategy in P2P-VoD Systems: Characterization and Evaluation," in *Proc. of INFOCOM*, Apr. 2011.

[9] 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.

[10] 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.

[11] F. Liu, S. Shen, B. Li, B. Li, H. Yin, and S. Li, "Novasky: Cinematic-Quality VoD in a P2P Storage Cloud," in *Proc. of IEEE INFOCOM*, Apr. 2011.

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

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

[14] (2010) CCTV. [Online]. Available: http://www.cctv.com

[15] J. Luo, Q. Zhang, Y. Tang, and S. Yang, "A Trace-Driven Approach to Evaluate the Scalability of P2P-Based Video-on-Demand Service," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 1, pp. 59–70, 2009.

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

[17] M. Wang and B. Li, "How Practical is Network Coding?" in *Proc. of IEEE IWQoS*, Jun. 2006.

[18] F. Liu, B. Li, L. Zhong, B. Li, H. Jin, and X. Liao, "Flash Crowd in P2P Live Streaming Systems: Fundamental Characteristics and Design Implications," *IEEE Transactions on Parallel and Distributed Systems*, 2011.

[19] J. Wu and B. Li, "Keep Cache Replacement Simple in Peer-Assisted VoD Systems," in *Proc. of INFOCOM*, Apr. 2009.

[20] F. Liu, B. Li, B. Li, and H. Jin, "Peer-Assisted On-Demand Streaming: Characterizing Demands and Optimizing Supplies," *IEEE Transactions on Computers*, 2011.

[21] B. Li, G. Y. Keung, S. Xie, F. Liu, Y. Sun, and H. Yin, "An Empirical Study of Flash Crowd Dynamics in a P2P-based Live Video Streaming System," in *Proc. of IEEE Globecom*, Nov. 2008.

[22] 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.

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

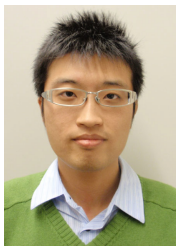
[24] M. Wang and B. Li, "R^ 2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, p. 1655, 2007.

[25] 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.

[26] D. Wu, T. Hou, W. Zhu, Y.-Q. Zhang, J. Peha, "Streaming Video over the Internet: Approaches and Directions," *IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on Streaming Video*, vol. 11, no. 3, pp. 282–300, Mar. 2001.

[27] H. Yin, C. Lin, Q. Zhang, Z. Chen, and D. Wu, "TrustStream: A Secure and Scalable Architecture for Large-scale Internet Media Streaming," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 12, pp. 1692–1702, Dec. 2008.

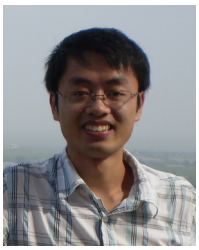
[28] B. Tan and L. Massoulié, "Optimal Content Placement for Peer-to-Peer Video-on-Demand Systems," in *Proc. of INFOCOM*, Apr. 2011.



Fangming Liu (S08-M11) received his B.Engr. degree in 2005 from Department of Computer Science and Technology, Tsinghua University, Beijing, China; and his Ph.D. degree in Computer Science and Engineering from the Hong Kong University of Science and Technology in 2011. He is currently an Associate Professor in the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. In 2011, he was awarded as the Chutian Scholar of Hubei Province, China; and he received the Best Paper Award from the IEEE GLOBECOM'2011.

In 2007, he worked as a research assistant at the Department of Computer Science and Engineering, Chinese University of Hong Kong. From Aug 2009 to Feb 2010, he was a visiting student at the Computer Engineering Group, Department of Electrical and Computer Engineering, University of Toronto, Canada. Since 2012, he has also been a StarTrack Visiting Young Faculty in Microsoft Research Asia, Beijing.

His research interests are in the area of large-scale content distribution systems, cloud computing and datacenter networking, peer-to-peer networks, green computing and communication. He is a member of IEEE and IEEE Communications Society, a member of ACM, and a member of the China Computer Federation (CCF) and CCF Internet Technical Committee. He has been a guest editor for IEEE Network Magazine, and served as a TPC for IEEE INFOCOM'2013 and GLOBECOM'2012-2013.



Shijun Shen received his Ph.D. degree in Computer Science and Technology from Tsinghua University, Beijing, China, in 2011. He is currently with the National Computer Network Emergency Response Technical Team/Coordination Center of China (CNCERT/CC). His research interests include peer-to-peer networks, video-on-demand and cloud computing.

Bo Li (S89-M92-SM99-F11) is a professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He holds a Cheung Kong Chair Professor in Shanghai Jiao Tong University, and is the Chief Technical Advisor for ChinaCache Corp., the largest CDN operator in China (NASDAQ CCIH). His current research interests include: large-scale content distribution in the Internet, datacenter networking, cloud computing, green computing and communications.

He made pioneering contributions in the Internet video broadcast with the system, Coolstreaming (the keyword had over 2,000,000 entries on Google), which was credited as the world first large-scale Peer-to-Peer live video streaming system. The work first appeared in IEEE INFOCOM (2005) has not only been highly cited, but also spearheaded a momentum in peer-to-peer streaming industry, with no fewer than a dozen successful companies adopting the same mesh-based pull streaming technique to deliver live media content to tens of millions of users in the world. He has been an editor or a guest editor for a dozen of IEEE journals and magazines. He was the Co-TPC Chair for IEEE INFOCOM 2004.

He received his B. Eng. Degree in the Computer Science from Tsinghua University, Beijing, and his Ph.D. degree in the Electrical and Computer Engineering from University of Massachusetts at Amherst. He is a Fellow of IEEE.

Baochun Li (S98-M00-SM05) received the B.Engr. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995 and the M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a Professor. He holds the Nortel Networks Junior Chair in Network Architecture and Services from October 2003 to June 2005, and the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include large-scale multimedia systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. Dr. Li was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the IEEE Communications Society, and a recipient of the University of Toronto McLean Award. He is a member of ACM and a senior member of IEEE.

Hai Jin is a Cheung Kung Scholars Chair Professor of computer science and engineering at the Huazhong University of Science and Technology (HUST) in China. He is now Dean of the School of Computer Science and Technology at HUST. Jin received his Ph.D. degree in computer engineering from HUST in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. Jin worked at The University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. Jin is the chief scientist of ChinaGrid, the largest grid computing project in China, and the chief scientist of National 973 Basic Research Program Project of Virtualization Technology of Computing System. Jin is a senior member of the IEEE and a member of the ACM. Jin is the member of Grid Forum Steering Group (GFSG). He has coauthored 15 books and published over 400 research papers. His research interests include computer architecture, virtualization technology, cluster computing and grid computing, peer-to-peer computing, network storage, and network security. Jin is the steering committee chair of International Conference on Grid and Pervasive Computing (GPC), Asia-Pacific Services Computing Conference (APSCC), International Conference on Frontier of Computer Science and Technology (FCST), and Annual ChinaGrid Conference. Jin is a member of the steering committee of the IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid), the IFIP International Conference on Network and Parallel Computing (NPC), and the International Conference on Grid and Cooperative Computing (GCC), International Conference on Autonomic and Trusted Computing (ATC), International Conference on Ubiquitous Intelligence and Computing (UIC).