UUSee: Large-Scale Operational On-Demand Streaming with Random Network Coding

Zimu Liu, Chuan Wu, Baochun Li, and Shuqiao Zhao

Abstract—**Since the inception of network coding in information theory, we have witnessed a sharp increase of research interest in its applications in communications and networking, where the focus has been on more practical aspects. However, thus far, network coding has not been deployed in real-world commercial systems in operation at a large scale, and in a production setting. In this paper, we present the objectives, rationale, and design in the first production deployment of random network coding, where it has been used in the past year as the cornerstone of a large-scale production on-demand streaming system, operated by UUSee Inc., delivering thousands of on-demand video channels to millions of unique visitors each month. To achieve a thorough understanding of the performance of network coding, we have collected 200 Gigabytes worth of real-world traces throughout the 17-day Summer Olympic Games in August 2008, and present our lessons learned after an in-depth trace-driven analysis.**

I. Introduction

Network coding, since its inception in information theory [1], has attracted a substantial amount of attention in networking research. It has widely been accepted that, with network coding, the cut-set bound of information flow rates in a multicast communication session can be achieved. As content distribution systems in the Internet attempt to disseminate content from a single source to a number of subscribing receivers, they naturally resemble multicast sessions. Gkantsidis and Rodruiguez [2] showed how to use network coding to improve download time by a 30% margin in peer-to-peer file content distribution. In this context, network coding helps to eliminate the need for finding rare data blocks. Small-scale trials with around a hundred testing nodes have been conducted to verify this claim [3].

In the context of peer-assisted live and on-demand video streaming systems, it has also been shown that network coding is beneficial in peer-assisted live streaming systems, especially in cases where server bandwidth supplies barely meet user demand [4]. R^2 [5] has proposed a set of design principles for new peer-assisted live streaming protocols to take full advantage of network coding. Feng *et al.* [6] have sought to mathematically analyze peer-assisted live streaming systems with network coding, with a focus on playback quality, initial buffering delays, server bandwidth costs, and extreme peer dynamics. It has been theoretically shown that network coding helps achieve provably good overall performance. Finally, Annapureddy *et al.* [7], [8] have shown similar advantages with the use of network coding, when it evaluated the use of network coding in peer-assisted on-demand streaming.

Zimu Liu and Baochun Li (contacting author: *bli@eecg.toronto.edu*) are with the Department of Electrical and Computer Engineering, University of Toronto. Chuan Wu is with the Department of Computer Science, The University of Hong Kong. Shuqiao Zhao is with UUSee Inc. This work is supported in part by NSERC Discovery, CRD and Strategic Grants (RGPIN 238994-06, CRDPJ 379623-08, STPGP 364910-08).

Despite the foundation established by extensive existing research in the literature, thus far, network coding has not yet been reported to be deployed in real-world operational video streaming or content distribution systems — or any large-scale operational systems in a production setting. What are the lingering challenges that prevent such a deployment? Would the benefits brought forth by network coding justify the computational costs on both servers and users? We believe it is time to begin a reality check of network coding in a commercial deployment at a scale involving millions of unique online users.

In this paper, we present the first production deployment of random network coding as a core technology in the UUSee on-demand video streaming system, operated by UUSee Inc. [9], one of the leading peer-assisted media content providers in China. The UUSee on-demand streaming protocol is specifically designed from the ground up to incorporate network coding, following the rationale and design objectives that we will illustrate. We seek to present an impartial, extensive, and in-depth analysis of 200 Gigabytes worth of operational traces, which we have collected throughout the 17-day Summer Olympic Games in August 2008, with UUSee as one of the official online broadcasting partners in China. The goal of our analysis is to make critical observations and draw conclusions about the suitability of using network coding in an operational environment with a large number of users. We also wish to locate new challenges to be addressed, and aspects of the protocol to be improved.

The remainder of this paper is organized as follows. In Sec. II, we present the objectives, rationale, and system design of the UUSee peer-assisted on-demand streaming system with network coding. In Sec. III, we describe details of our measurement traces, collected over the 2008 Beijing Olympics. In Sec. IV, we present a thorough analysis of our traces, with a focus on the performance of our new on-demand streaming system based on network coding. With the hope of identifying new challenges to be addressed to improve system performance, we make impartial and critical observations about the lessons learned in our trace-driven analysis, and then conclude the paper in Sec. V.

II. Objectives, Rationale and System Design

At UUSee Inc., the ultimate design objective is to design, test, and implement the best possible on-demand video streaming system from the ground up, that scales up to thousands of on-demand video channels and millions of users, without the penalty of excessive server bandwidth costs. It was a daunting and risky task to venture into the unchartered territory of network coding, as its performance benefits, while extensively studied and theoretically sound, had never been tested in a real-world, large-scale deployment at the time of our design. In this section, we begin with our performance objectives in the UUSee design, and then illustrate the factors that have motivated us to deploy random network coding.

A. Design Objectives

Following the success of peer-assisted live streaming, on-demand streaming has recently been deployed in a peer-assisted fashion, as documented in the PPLive case study [10]. While it is certainly feasible to deploy on-demand video streaming at a large scale, in our view, existing systems have not yet achieved the *best possible* performance feasible in on-demand streaming systems. What constitutes critical criteria that can be used to judge if a new system is *better* than existing ones, then? In the process of redesigning the UUSee on-demand streaming system, we seek to achieve the following design objectives.

▷ *Minimized server bandwidth costs.* Server bandwidth available from data centers and content distribution networks (CDNs) is a *finite* and *expensive* resource. For example, it has been estimated that Google consumes \$500 million every year on YouTube bandwidth costs. No matter how much bandwidth we provision at the dedicated streaming servers, it has to be utilized judiciously to serve the largest number of users possible. Using peer assistance — by designing peer-to-peer (P2P) protocols to facilitate bandwidth assistance from peer uploads — can substantially reduce the volume of server bandwidth used, but the strategy will not invalidate the need to conserve server bandwidth. The UUSee design of peer-assisted on-demand streaming seeks to conserve server bandwidth to the maximum extent possible, by maximizing peer bandwidth contributions to serve other peers.

▷ *Minimized buffering delay after a random seek.* In on-demand video streaming systems, one of the most important performance metrics is the *buffering delay* after a random seek operation, which is the time a user would have to wait for after seeking to a new point in the video stream for on-demand playback. We wish to minimize the average buffering delays across all channels.

▷ *Consistently satisfactory playback quality.* In on-demand streaming systems, to maintain a consistently satisfactory level of playback quality for a streaming user, the system only needs to make sure that the average download rate for a video stream is higher than the streaming playback rate of that stream. In the UUSee design, we certainly wish to maintain a satisfactory level of playback quality, as a basic requirement to maintain the quality of user experience.

B. Rationale of Applying Network Coding

Random linear network coding has been well established in recent research literature (*e.g.,* [11]), and has been shown to be able to take advantage of redundant network capacity that is otherwise left untapped. In the context of on-demand media streaming, each media segment is divided into n blocks $\mathbf{b} = [b_1, b_2, \ldots, b_n]^T$, where each block has a fixed number of bytes k (referred to as the block size). In practical random network coding [12], to code a new coded block x_j, the source first independently and randomly chooses a set of coding coefficients $[c_{j1}, c_{j2}, \cdots, c_{jn}]$ in $GF(2^8)$, one for each original block it has buffered. It then produces one coded block $x_j = \sum_{i=1}^{n} c_{ji} \cdot b_i$. While peers are free to send out new coded blocks by recoding the coded blocks received, a receiving peer decodes as soon as it has received n linearly independent coded blocks $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$. It first forms an $n \times n$ coefficient matrix $\mathbf{C}$, using the coefficients of each block b_i. Each row in $\mathbf{C}$ corresponds to the coefficients of one coded block. It then recovers the original blocks $\mathbf{b} = [b_1, b_2, \ldots, b_n]^T$ as $\mathbf{b} = \mathbf{C}^{-1}\mathbf{x}$, computed using Gaussian elimination or Gauss-Jordan elimination, the latter of which can be performed progressively as coded blocks are being received. The inversion of $\mathbf{C}$ is only possible when its rows are linearly independent.

It has been proposed that random linear network coding can be deployed in peer-assisted live streaming systems [5]. What is the most important benefit of random linear network coding in live streaming systems, and is such a benefit also applicable in on-demand streaming?

The upshot of network coding is that it makes "perfect coordination" possible, where an arbitrary number of serving peers (referred to as *seeds*) can be used to serve *the same* segment to a receiving peer, illustrated in Fig. 1. When multiple peers serve the same segment, it does not matter how bandwidth availability varies on each seed-to-peer connection. The receiving peer simply "holds a bucket" for the segment until it is "full." With Gauss-Jordan elimination, the receiving peer can even progressively recover original blocks as it receives coded ones in a "pipelining" fashion, so that the entire segment is immediately playable after the last block is received — from *any* of the seeds. This is due to the fact that, with random linear network coding, randomly generated coding coefficient vectors from different seeds are linearly independent with high probability. There is no need for seeds to explicitly exchange protocol messages with one another or with the receiving peer.

What are the immediately intuitive benefits derived from such "perfect coordination" using network coding? We believe that the advantages are two-fold.

▷ *Being oblivious to seed qualities.* With multiple seeds serving the segment to one receiving peer, dedicated streaming servers and ordinary peers can collaborate with ease, and there is no need to design an elaborate protocol to *select* a high-quality peer or lightly-loaded streaming server. As long as a seed (a server or a peer) has the segment in its cache, its upload capacity can be utilized to serve such a segment. Even the "slowest" seed can be tapped to serve at least a coded block, if the block size is sufficiently small.

▷ *Rateless erasure codes.* Since random linear codes are rateless erasure codes and are near-optimal, they transform an n-block message into a practically infinite encoded form, identical to known fountain codes (such as the LT code). In other words, encoded blocks can be generated *ad infinitum*,

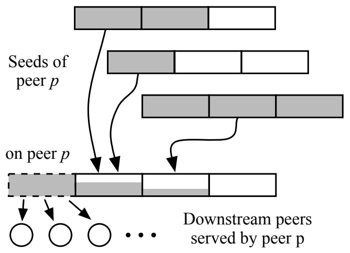

Fig. 1. Perfect coordination.

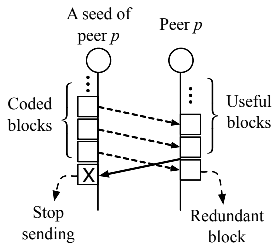

Fig. 2. "Braking:" the end game.

and as long as n linearly independent blocks are received, they are sufficient to recover the original segment. In the context of peer-assisted streaming systems over the Internet, this property directly implies that UDP flows can be used to communicate these blocks, rather than TCP connections, since losing coded blocks between a sending and a receiving peer is no longer a concern — more will be received from the same sending peer or a different one. Since UDP is stateless, it scales well on dedicated streaming servers and ordinary peers alike.

The good news is that, while these advantages hold in the context of live streaming, we see no reason why they do not hold in the peer-assisted on-demand streaming system to be designed in UUSee.

C. Challenges of Applying Network Coding

What, then, is the flip side of the coin? Tradeoffs accompany any advantages, and it is important to identify challenges when random network coding is applied in a real-world operational streaming system.

The first challenge we have encountered is how the size of a block is determined. A coded block is the basic transmission unit used to serve a portion of a segment, and to accommodate slower seeds so that they serve at least one block in a reasonable amount of time, we inclined to use a smaller block size, such as 1 KB, so that a block directly corresponds to a UDP packet. This is a plausible choice if there were no overhead imposed by coding coefficients. With random linear coding on $GF(2^8)$, each coefficient occupies a byte, and such overhead depends on the number of blocks in a segment.

Shall we use a smaller number of blocks in each segment to mitigate such coefficient overhead in a small 1 KB UDP packet? The answer is that we cannot afford to, since a smaller number of blocks will lead to a different type of overhead related to the "end game" after a segment is completely received and decoded on the receiving peer. Illustrated in Fig. 2, "braking" acknowledgment messages need to be sent back to each of the seeds, respectively, to stop them from sending more coded blocks. Due to the latency for these braking messages to be received by sending peers, additional redundant blocks may be received by the receiving peer after the segment is complete, and will need to be discarded. As the number of blocks in a segment becomes smaller, the overhead of such redundant blocks is more significant.

Finally, the need for exchanging segment availability information between sending and receiving peers calls for a larger segment size, since a larger size reduces the number of bits required to represent segment availability in an on-demand video stream. However, our preliminary tests on coding complexity have shown that, even with the most optimized implementation of random linear coding, going beyond 512 blocks in each segment risks taxing a low-end CPU, typical in power-efficient notebook computers [13].

D. UUSee: System Design with Network Coding

Attracted by the rationale motivating the use of network coding, our objective is to redesign the on-demand streaming system from the ground up to take advantage of network coding, while mitigating its drawbacks in practical use.

Similar to a traditional peer-assisted video-on-demand system such as PPLive [10], thousands of channels are supported in the system, and we use *tracking servers* to maintain information about participating peers in all video channels, including their IP addresses, listening ports, and videos currently cached. Peers (streaming users) are organized in a mesh topology similar to any alternative peer-to-peer streaming system, with each peer serving cached video segments to a number of downstream peers, while being served by a number of seeds with respect to segments it needs to receive. Seeds can be dedicated streaming servers or ordinary peers, but the objective is to maximize contributions from ordinary peers so that server bandwidth costs can be minimized.

When a peer selects to watch a video in the UUSee on-demand streaming system, the tracking server introduces it to the existing mesh, involving both peers and servers, by assigning it a number of existing peers having contents around its starting playback point cached. A *cache* is maintained on the peer in non-volatile storage, which stores segments of video channels previously viewed and already downloaded. In UUSee, to fully utilize peer upload bandwidth, it is possible for a peer to serve segments to other peers, even if the user is not currently viewing any video at the time.

In response to the aforementioned challenges of applying network coding in operational on-demand streaming systems, we have adopted the following design choices in UUSee.

▷ *Overhead.* Applying the *lesser of two evils* principle, we believe that the overhead caused by redundant blocks—received after the segment is completely downloaded by a receiving peer—is a more significant concern. To mitigate such overhead, we must use a larger number of blocks in a segment, and a smaller block size. In UUSee, we use 1 KB as the size of a block, corresponding to a single UDP packet. We use 300 to 500 blocks in a segment, depending on the streaming rate of the video channel. With this design choice, if we continue to embed coding coefficients into coded blocks, overhead up to 50% of the data size will be incurred. As such, we have no choice but to embed only the *PRNG seed* that is used to produce the sequence of random coefficients with a known pseudo-random number generator (PRNG). This effectively reduces the overhead to just 4 bytes, regardless of the number of blocks, n, in a segment. The only side effect is that peers are no longer able to serve coded blocks to others before completely receiving a segment [4]. However, since on-demand streaming systems do not have any requirements on the playback lag between live events and their playback (an important concern in live streaming), such a side effect is a non-issue in on-demand streaming.

▷ *Exchanges of segment availability.* In any streaming mesh, peers need to exchange content availability information among one another, in order to select serving peers to stream segments from. In on-demand streaming, peers cache video segments in non-volatile storage, and can afford a larger cache size than live streaming by two orders of magnitude, *e.g.,* 2 GB in on-demand vs. 10 MB in live streaming. With design choices on block and segment sizes when network coding is used, the number of segments in the peer cache is simply too large for segment availability information to be exchanged, even with just a single bit needed per segment.

To alleviate the overhead, the UUSee on-demand streaming system has employed a two-level cache map design. At the coarse-granularity level, *group availability bitmaps* (*group maps*) are used to indicate the groups in a video stream a peer currently holds in its cache. Group maps are typically only a few bytes long. A corresponding bit is set to 1 when a peer has cached 80% of content belonging to a group. At the fine-granularity level, *segment availability bitmaps* (*segment maps*) are used to represent segments a peer currently holds within a specific group, which is typically $10-30$ bytes long. A corresponding bit is set to 1 when a peer has fully downloaded and decoded an entire segment.

With the two-level design, peers with the same group can be organized to serve one another, and segment availability information can be exchanged among peers holding the same group. In UUSee, the use of network coding helps substantially to mitigate the overhead of exchanging segment availability. Segment sizes in UUSee are as large as 500 KB, as compared to only 14 KB in PPLive [14]. This implies that the sizes of segment maps are up to 35 times smaller. Table I illustrates the sizes of media units at different granularities, adopted in the UUSee system design.

▷ *The push protocol for perfect coordination.* Since random linear network coding is used to achieve perfect coordination across multiple seeds, coded blocks from any of the seeds are equally useful. Upon the explicit request from a downstream peer p for a particular video segment, a seed starts to *push* freshly generated coded blocks consecutively to p as UDP packets (and with flow control), until the braking acknowledgment message arrives. The downstream peer p may send explicit requests to more than one seed, and may simply send one braking acknowledgment message (as the "stop" signal) to each seed after it has successfully decoded the segment progressively using Gauss-Jordan elimination.

With the push protocol and perfect coordination, upload bandwidth at ordinary serving peers can be fully utilized. Only one request is sent for each segment to each serving peer, whereas acknowledgment messages only need to be sent after completely receiving a segment ($300-500$ KB in UUSee), which has a much larger size than that without network coding.

TABLE I
UUSEE ON-DEMAND STREAMING: MEDIA UNITS

Unit	Functional Purpose	Size
Video	Unit for cache storage	> 100 MB
Group	Unit for content search in neighbor discovery	> 10 MB
Segment	Unit for fine-grained content exchange and playback	~ 500 KB
Block	Unit for coding and transmission	1 KB

If network coding is not used, an explicit request needs to be sent for every single small segment. To saturate available upload bandwidth from ordinary peers and to minimize server bandwidth usage, every packet matters — the significant savings in request messages amount to more coded blocks being served by peers. Coupled with perfect coordination, the small block size used, and careful overhead mitigation, the UUSee system is designed to scale to a large number of users with a finite pool of dedicated servers.

III. MEASUREMENT TRACES

In January 2008, we have incorporated our system design in a production-quality implementation, from the ground up as a complete redesign of peer-assisted on-demand streaming. It has been tested and deployed in the UUSee operational streaming system shortly after. To closely monitor, inspect, and evaluate the run-time behavior of the system, we have implemented detailed measurement and reporting capabilities within each UUSee client. Each peer collects a set of its vital statistics, encapsulates them into "heartbeat" reports to be sent to our dedicated logging servers every 5 minutes via UDP. The statistics can be classified into three categories:

▷ *General information* includes the video it is playing, its instantaneous aggregate download and upload throughput from and to all neighbors, its download and upload bandwidth capacities, as well as a list of all its neighbors, with the number of blocks sent to or received from each neighbor within the past 5 minutes, and the type of source for the blocks exchanged. There are three types of sources, namely *media server*, *playing peer* (*i.e.*, the neighbor serves the block from a video it is currently watching), and *cached peer* (*i.e.*, the neighbor serves the blocks from a cached video it previously watched).

▷ *Network coding metrics* include the overall number of segments requested from neighbors in the past 5 minutes, along with the total number of linearly dependent blocks received within these segments. In addition, for each individual UDP connection between a peer and its neighbor, we record the average number of redundant blocks received per segment over all the requested segments in the past 5 minutes.

▷ *User behavior data* record critical operations of a peer, including joins, departures, and random seeks, as well as the buffering delays experienced upon its joins and random seeks.

Our UUSee measurements, to be presented in the remainder of this paper, feature 200 Gigabytes worth of operational traces, collected during the 17-day 2008 Summer Olympics, from 14:51:58, Friday, August 8, 2008 (GMT+8) to 23:43:56, Sunday, August 24, 2008 (GMT+8). With snapshots of the entire UUSee system every 5 minutes throughout this period,

TABLE II
STATISTICS OF THE UUSEE TRACE SET

Number of Videos	993
Number of Peer Sessions	31,836,830
Number of Video Sessions	142,924,594
Number of View Durations	274,415,220
Number of Heartbeat Reports	3,742,295,041

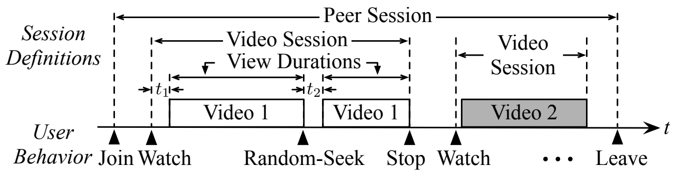

Fig. 3. Session model for an online peer: t_1 — the buffering delay after selecting a video to watch; t_2 — the buffering delay after a random seek.

we believe this set of traces best captured the characteristics of UUSee in a variety of typical scenarios, including large flash crowds gathered to watch many Olympic videos as soon as they were published on UUSee.

Basic statistics and session model

Table II presents a glance of the scale of the traces. We give a detailed session model in Fig. 3, to distinguish the different durations occurred during a peer's online time. We refer to the period between join and departure of a peer into and from the UUSee system as a *peer session*, and the duration in which a peer watches the same video as one *video session*. Within one video session, the peer may jump to watch different portions by issuing random seek commands, and the duration in which a peer continuously watches a part of a video is referred to as one *view duration*. From the statistics in Table II, interesting observations can be made: a peer on average switches videos to watch 4.5 times during one peer session, and issues 1.9 random seek commands within each video session.

Distribution of video popularity

Fig. 4 shows the distribution of user population across the 993 videos in our UUSee traces, in terms of the cumulative number of video sessions and the peak number of concurrent peers on a typical day for each video. The plots illustrate the large span of peer population across videos, from thousands of concurrent peers to only a few dozens.

In UUSee, videos are encoded into different streaming bitrates between 264 kbps and 1.3 Mbps, corresponding to different streaming qualities. In our study, we have categorized the videos into two classes, namely the *normal-quality* videos (914 videos in total), with streaming bitrates around 264 − 800 kbps, and *high-quality* videos (79 in total), with streaming bitrates larger than 800 kbps. We have observed that the high-quality videos are usually less popular — even the most popular high-quality video has only 127 concurrent viewers. This is possibly due to the less sufficient last-mile bandwidth at most UUSee users in China, which poses a significant challenge in streaming these videos: the higher the playback bitrate of a video is, the larger bandwidth a peer demands for a smooth playback; with few peers watching the same video, even scarcer bandwidth supply can be obtained from peers, and the deficiency has to be compensated by the servers. It has therefore been an important task in our measurement study to investigate the on-demand streaming performance of high-quality videos. In addition, videos of the 2008 Summer Olympics include a number of flash crowd scenarios with thousands of concurrent peers. The streaming performance during such scenarios represents another focus of our investigation.

TABLE III
STATISTICS OF THREE REPRESENTATIVE VIDEOS

Sample	Description	Length (min)	Bitrate (kbps)	Peak Population
NQ	A normal-quality video	122	471	284
HQ	A high-quality video	157	966	98
OG	An Olympic Games video	94	637	2491

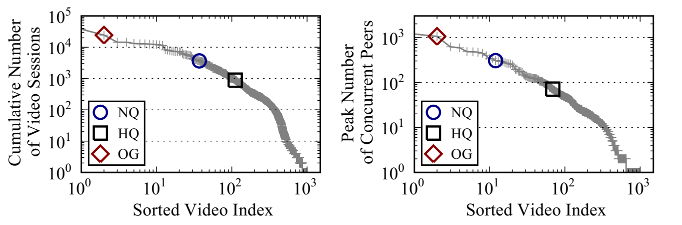

Fig. 4. Distribution of video popularity on August 12, 2008, with three representative samples marked.

We have zoomed into a large number of videos to explore a variety of performance characteristics in our study. For ease of illustration, we have chosen three representative videos to present our measurements at the microscopic level, but will also give results regarding all channels at the macroscopic level whenever needed. Table III lists the basic statistics of the three representative videos, the samples of which are also marked in Fig. 4.

IV. NETWORK CODING IN OPERATION: MEASUREMENTS AND ANALYSES

We now investigate the suitability of using network coding in the new UUSee system design, by exploring properties revealed by our 200 GB worth of real-world traces.

A. Playback Quality

A smooth playback quality represents the most desirable user experience in an on-demand streaming system, as provided by a consistent download rate no lower than the streaming playback bitrate of a video stream. To investigate such playback smoothness in UUSee, we have carefully examined the download rates at peers in different channels. Fig. 5 plots the average download rates across all peers in the three representative video channels, respectively, along with the evolution of the number of simultaneous peers in each channel, from 00:00, Sunday, August 10, 2008 to 23:55, Saturday, August 16, 2008.

The sample NQ and the sample HQ are both classic movies available on UUSee since the beginning. Fig. 5(a) and (b) show a regular daily evolutionary pattern of the concurrent peer population watching the two videos, respectively. We observe that for the NQ video, the average download rate is above the required streaming rate for the majority of times, during both daily peak hours and off-peak hours. For the HQ video,

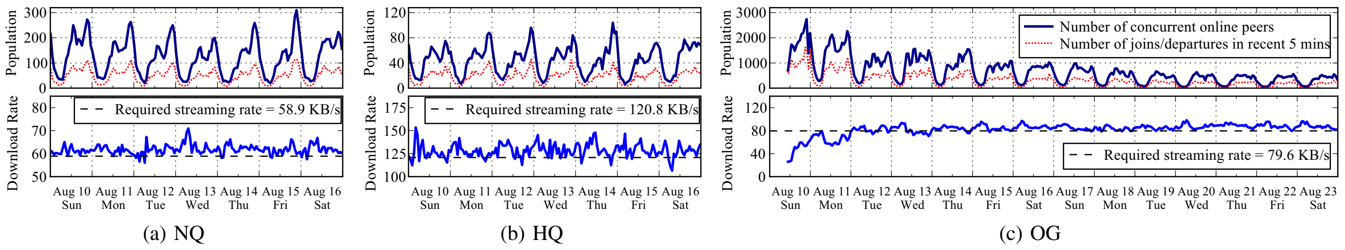

Fig. 5. Evolution of the concurrent peer population and average download rate (in KB/s) in three representative channels.

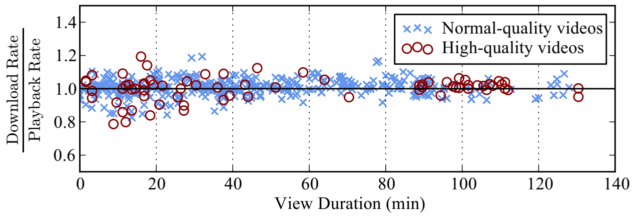

Fig. 6. Normalized playback quality vs. view duration for all channels.

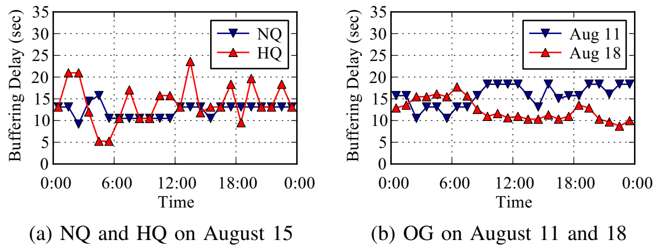

Fig. 7. Evolution of the average buffering delay in 3 representative channels.

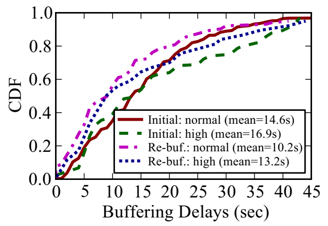

Fig. 8. CDF of two types of buffering delays over all videos in normal-quality and high-quality categories.

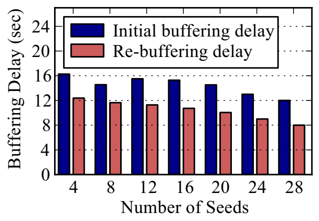

Fig. 9. Two types of buffering delays vs. number of available seeds during the buffering period at all peers.

download rates vary more significantly over time and there are more times when it is not up to the required streaming rate, that happen mostly during the hours with more severe peer churn (evaluated by the number of joins/departures in the recent 5 minutes in the figures). Considering the high bandwidth demand and few supplying peers in the HQ channel, the observations come without much surprise. Nevertheless, we are going to explore its detailed causes with respect to the server/peer bandwidth supplies in our later discussions.

The sample Olympic Games video (OG) was published on UUSee around noon on August 10, 2008. As shown in Fig. 5(c), there are flash crowds of viewers during the first two days of its release, with the interest decreasing over time. During the flash crowds, the average download rate is relatively low, as the sustainable threshold for the video has been breached, when the number of supplying peers in the system, who have watched and can thus serve the video from their caches, is small. Nevertheless, the download rate quickly picks up and goes beyond the required streaming rate starting from the third day on, when an increasing number of supplying peers emerge. We are going to explore the role of peer caches in the playback quality in details in Sec. IV-C.

We now investigate the impact of VCR dynamics within a channel on the playback smoothness of peers in the channel in Fig. 6. The level of VCR dynamics in a video channel is evaluated by the average view duration of peers watching the video across the trace period, where a shorter average view duration represents more frequent peer VCR operations in the channel. Fig. 6 plots the average view duration vs. the average ratio of the download rate over the required playback rate within all UUSee channels, where each sample represents one video channel. We observe only slightly worse playback smoothness in channels with severe VCR dynamics, than those with long staying peers.

B. Buffering Delay

Theoretically, the use of network coding achieves perfect coordination among peers, and is thus promising to shorten the buffering delay typically experienced at peers. To explore the gain of network coding in this aspect, we plot in Fig. 7 the evolution of average buffering delays in the three representative videos, including both the *initial buffering delay* upon peer joining the channel and the *re-buffering delay* after each random seek operation. We observe that the average buffering delay for the NQ video is always lower than 13 seconds, during both peak and off-peak hours. This is 28% shorter than the measurement result (18 seconds) in comparable videos on the PPLive VoD streaming platform [10]. As a peer starts to play after buffering 24 seconds of video content in UUSee, our result (24 seconds of video downloaded in 13 seconds) is also better than the measurement from GridCast [15], in which 70% of peers use 8 seconds to buffer 10 second worth of video content. On the other hand, the delay in the HQ channel could be larger, but is in general below a reasonable 20 seconds. Surprisingly, Fig. 7(b) shows that the buffering delay in the OG video is still reasonable low during the flash crowd on August 11, *i.e.*, 18 seconds, as compared to that on August 18 with much fewer viewers.

To explore the situation across all videos, we plot in Fig. 8 the CDF of buffering delays at all peers in all video channels across the trace period. The initial buffering delays and re-buffering delays in both the normal-quality and high-quality categories are averaged below 17 seconds, with those for the latter category only $2-3$ seconds longer than the first category.

While one would expect a longer buffering delay when peers

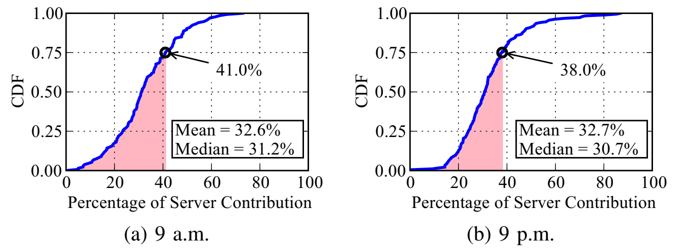

Fig. 10. CDF of the percentage of server bandwidth contribution over all videos in two snapshots on August 15.

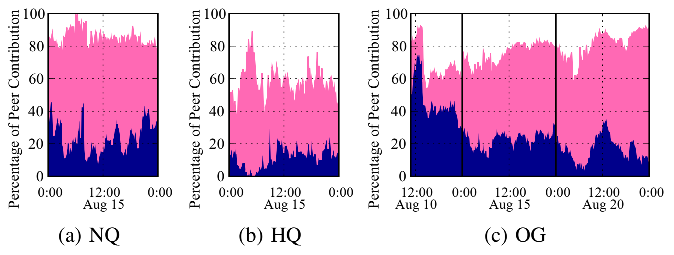

Fig. 11. Evolution of the percentage of peer bandwidth contribution in three representative video channels: the darker area represents contribution from playing peers; the lighter area represents contribution from cached peers.

have fewer suppliers in a typical streaming system, we observe from Fig. 9 that even with only $1-4$ seeds available during the buffering period, the average buffering delays are still at similar levels as cases with over 20 seeds. This shows that UUSee can generally provide users a short and stable buffering delay regardless of their neighborhood sizes, by using network coding that enables prefect coordination among peers.

C. Server Bandwidth Consumption

Having observed the benefits of network coding in enhancing peer streaming qualities, we next examine whether perfect coordination with network coding is able to conserve server bandwidth consumption to the maximum extent possible, which is the most important design objective in peer-assisted on-demand streaming. We investigate the percentages of bandwidth contributions from servers and peers within all the downloads, respectively, inside each channel.

Fig. 10 shows the CDF of the percentage of server bandwidth contribution for all the channels, in two snapshots of the UUSee system at 9 a.m. and 9 p.m., August 15, 2008, respectively. We observe that the mean percentage of server contribution is around 30% and the 75th percentile is 41%. These numbers are impressively lower than the only known relevant measurement in [15], which shows that 73% of the video content viewed in GridCast was directly served by the servers. Fig. 10 also presents a slight drop of server bandwidth consumption at the peak hour (9 p.m.) than at the off-peak hour (9 a.m.), revealing the good scalability of UUSee, that peers are contributing more bandwidth relative to the servers even with a large number of concurrent viewers. This observation is further validated by Fig. 12, in which we plot the percentage of server contribution against channels with different populations, and observe a decreasing level of server consumption in channels with increasing numbers of peers.

Zooming into individual channels, we take a closer look at the percentage of bandwidth contribution from playing peers and cached peers in Fig. 11. We observe that 80% of the streaming bandwidth is served by peers in normal quality videos (NQ and OG) and 60% in the high-quality video (HQ). Among the bandwidth contribution from peers, a large portion (about 70%) is from cached peers. Especially for the Olympics video, cached copies place an increasingly important role over time when more peers have watched it. To achieve this significant alleviation of server bandwidth, our trace study has also shown that up to 80% of the upload capacity is efficiently utilized at any online peer at any time to serve useful blocks for

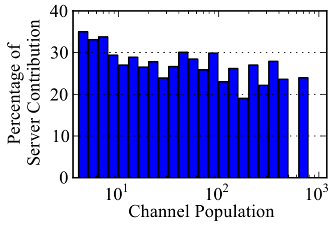

Fig. 12. Average percentage of server contribution vs. channel population for all channels over the trace period.

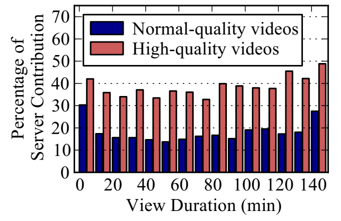

Fig. 13. Percentage of server contribution vs. view duration in two categories of channels.

videos in its cache — currently watched or previously viewed. All these could not have been possible without our design using network coding.

We further investigate the relation between server consumption and the level of VCR dynamics in the channels. Fig. 13 shows only slight server consumption increases in case of shorter view durations, for both normal-quality and high-quality video channels. This, again, benefits from perfect coordination with the use of network coding.

From Fig. 11(b) and Fig. 13, we observe the relative lack of contributions from a smaller number of peers in high-quality channels. A closer examination reveals that most ISPs (that UUSee peers belong to) have capped the upload capacities of users to 512 kbps, which is only 64% of the playback bitrate in a typical 800 kbps high-quality channel. Hence, to supplement the deficiency, the servers play a more significant role in serving video content in those channels. As UUSee deploys a limited number of servers to serve each channel, the available server capacity may not have been sufficient for these bandwidth-demanding channels, thus leading to the slightly lower playback quality as shown in Sec. IV-A.

D. Signaling Overhead

Any advantages may come with tradeoffs. We now start to look at the flip side of the coin, with respect to the overhead brought by applying network coding. We first investigate the signaling overhead in UUSee, which includes three types: (1) block header overhead, about 30 bytes per block transmitted, containing the four-byte PRNG seed, segment index, and other metadata; (2) group and segment maps; and (3) the segment request and braking acknowledgment messages. We evaluate the signaling overhead of each type using the percentage of their incurred traffic volume in the total traffic volume in each video channel. Fig. 14 plots the CDF of the overall signaling overhead in all the channels on a typical day. We observe that

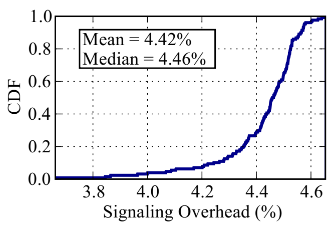

Fig. 14. CDF of the overall signaling overhead over all channels.

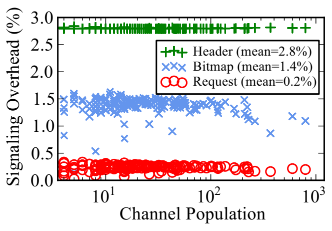

Fig. 15. Three types of signaling overhead over all channels.

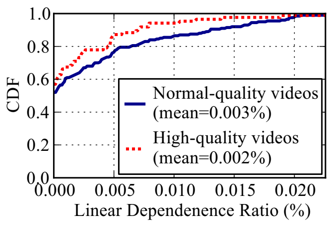

Fig. 16. CDF of linear dependence ratios over two categories of videos.

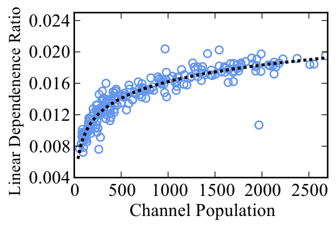

Fig. 17. Linear dependence ratio vs. population in the OG channel.

the overall signaling overhead in any channel is lower than 4.65%. Measurements of PPLive have shown a comparable overhead of 10% [10], which is twice as high as ours.

Fig. 15 breaks down three types of signaling overhead and plots them against channels with different populations. We observe that the segment request and braking acknowledgment messages only take up a minimal 0.2% of the overall traffic in the system, benefiting largely from UUSee's push protocol and its choice of large segment sizes (300−500 KB). In addition, the two-level cache map design and large segment sizes have also kept the overhead of exchanging group/segment availability bitmaps to as low as 1.4% on average. The block header overhead, around 2.8%, is already minimized as compared to carrying all coefficients with each block. Regardless of channel populations, the signaling overhead remains at similar levels, which further exhibits the excellent scalability of the UUSee design in a real-world deployment.

E. Linear Dependence

Though blocks coded with independently and randomly chosen coding coefficients are linearly independent with a high probability in theory [16], in a realistic system, more linear dependence may appear due to a number of practical implementation issues, *e.g.*, a sequence of coding coefficients generated from a PRNG seed may not be as "random" as those uniformly randomly selected from $GF(2^8)$. It is therefore an important task for us to evaluate how significant the ratio of redundant linearly dependent blocks over all received coded blocks at each peer is (referred to as *linear dependence ratio*), in order to evaluate the potential negative effects of network coding in a real-world operational system.

Fig. 16 plots the CDF of the average linear dependence ratio per peer in all the channels over the trace period. The ratio in all the channels is no larger than 0.00023 (0.023%), which is indeed very low, and more than half of the channels enjoy zero linear dependence. High-quality channels generally have lower linear dependence than normal-quality ones, mainly due to the lower probability of conflicts among PRNG seeds at a smaller number of peers.

We have further explored any possible correlation between the channel population and the linear dependence ratio in Fig. 17, using continuous snapshots over two weeks of the OG traces. We observe a slight logarithmic increase of the linear dependence ratio with the increase of the channel population. We may infer that even with millions of concurrent peers in a video channel, the linear dependence ratio will only be around 0.036%, which is still very low. All these observations show

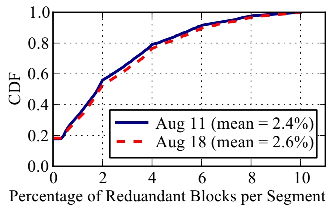

Fig. 18. CDF of the braking redundancy over all channels.

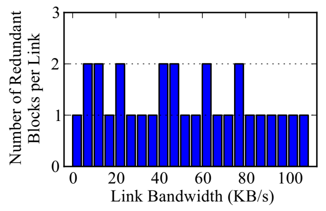

Fig. 19. The number of redundant blocks per link vs. link bandwidth.

that our practical network coding implementation produces few redundant blocks due to linear dependence.

F. Braking Redundancy

As it takes time for the braking messages to reach the seeds, a downstream peer may receive additional redundant blocks after a segment is complete, as discussed in Sec. II-C. To reduce braking redundancy within the constraints of coding complexity, we have carefully chosen a relatively large number of blocks, 300 − 500, in each segment. We now validate this design choice by evaluating the braking redundancy, as the average percentage of additional redundant blocks received for each segment, at peers in UUSee on-demand streaming.

Fig. 18 plots the CDF of the braking redundancy at all peers in all channels on two typical days. We observe an average redundancy percentage of about 2.5% per segment. Considering the number of neighbors each peer may have (10 − 30), we derive that the average number of redundant blocks per segment, received along each download link at a peer, is at most 1 − 2. As a sending peer pushes blocks faster when the link bandwidth towards the receiver is larger, we naturally wonder whether the braking redundancy is higher along links with high bandwidth. We investigate this in Fig. 19, where we group P2P links in all the channels according to their capacities, and plot the mean number of redundant blocks received per segment along the links in each group. We observe no evident relation between the two quantities, and the number of redundant blocks per link remains at only a few. We believe such a low redundancy level, 1−2 redundant blocks per link for each segment with 300 − 500 blocks, is already the minimum we can achieve, and the choice of a large number of blocks in a segment, subject to computational constraints of decoding, is appropriate to maintain a low percentage of redundancy.

V. Lessons Learned and Concluding Remarks

In UUSee, we have designed and implemented the first operational on-demand streaming system with random network coding, and have observed a superior level of real-world streaming performance. It has become evident that our design objectives have been achieved: multiple seeds are allowed to collaboratively serve a peer, leading to minimized buffering delays and server bandwidth costs. The playback quality has been satisfactory for normal-quality videos. For high-quality videos, the use of network coding has downgraded negative effects when the server bandwidth supply becomes tight. With an attempt to be impartial, we now wish to critically examine what has been *less* satisfactory from our measurements on 200 GB of traces, and inspect possibilities for improvement by revisiting our design choices.

In retrospect, one of the most pressing concerns in our measurement studies is the *overhead* incurred by network coding. We are eager to revise our design choices so that overhead can be minimized, especially the redundancy due to latencies of the "braking" acknowledgment messages after segments have been completely downloaded. While one or two redundant blocks do not seem to be excessive, the total number of these redundant blocks, however, increases linearly as more seeds are engaged in a perfectly coordinated fashion. One design alternative is to refine the protocol to allow an "early braking" mechanism, which allows a downstream peer to acknowledge a subset of its seeds even *before* all n linearly independent blocks are received. Of course, important design challenges abound as this possible alternative is explored in real-world streaming: When should a peer send out such "early braking" acknowledgment messages? What are the best candidates among the seeds that should be chosen to receive these messages? An incorrect design would easily swing the pendulum too far to the other extreme: a peer may be too aggressively stopping seeds, and suffering from a longer time completing the download.

While the "early braking" design remains an engineering heuristic, would it be possible to learn from recent theoretical advances in network coding? The primary roadblock towards using more blocks in each segment has been the computational complexity of decoding: $O(n^3 + n^2k)$ operations on $GF(2^8)$ would have to be performed, for a segment with n blocks, each of k bytes. Silva *et al.* [17] have proposed a sparse network coding approach with overlapped *classes* (loosely corresponding to *segments* in this paper), and can be seen as a combination of fountain coding and network coding. The basic idea is to decrease the decoding complexity by allowing blocks from decoded classes (using Gauss-Jordan elimination) to be back-substituted into still undecoded classes. The bad news, however, is that such faster decoding requires a block to belong to multiple overlapped classes (analogous to letters in a crossword puzzle), which leads to additional overhead to maintain more coefficient vectors. The new code is also designed to minimize overhead without feedback, and it is not clear how it can reduce redundancy when feedback is allowed but delayed.

Finally, it has been proposed in [18], [19] that, rather than acknowledging the number of packets received as in TCP, the *degree of freedom* can be acknowledged to the sender as feedback if network coding is used. The concept may very well be useful in the "early braking" heuristic, where feedback does not have to be *binary* (as in "green" vs. "red"), and can instead include finer-granularity information about the current state of downloading a segment, *i.e.,* the degree of freedom in the coded blocks received so far. When the degree of freedom reaches n, the download is complete and all seeds will stop sending. Before it reaches n, seeds can *reduce* their rate of sending to mitigate overhead, effectively corresponding to a new design of *flow control* with network coding. While promising, the degree of freedom itself needs more bits to code compared to a single bit in the "binary" feedback mode, and it is not clear if the additional overhead consumed on feedback can be justified. In upcoming revisions of the UUSee on-demand streaming system with network coding, we will continue to evaluate existing theoretical advances in network coding, leaving no stone unturned to realize its advantages and mitigate its drawbacks.

REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network Information Flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[2] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proc. INFOCOM'05*, Mar. 2005, pp. 2235–2245.

[3] C. Gkantsidis, J. Miller, and P. Rodriguez, "Comprehensive View of a Live Network Coding P2P System," in *Proc. IMC'06*, Oct. 2006, pp. 177–188.

[4] M. Wang and B. Li, "Lava: A Reality Check of Network Coding in Peer-to-Peer Live Streaming," in *Proc. INFOCOM'07*, May 2007, pp. 1082–1090.

[5] ——, "R^2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming," *IEEE J. Sel. Areas Commun.*, vol. 25, no. 9, pp. 1655–1666, Dec. 2007.

[6] C. Feng and B. Li, "On Large-Scale Peer-to-Peer Streaming Systems with Network Coding," in *Proc. ACM Int. Conf. on Multimedia (Multimedia'08)*, Oct. 2008, pp. 269–278.

[7] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez, "Exploring VoD in P2P Swarming Systems," in *Proc. INFOCOM'07*, May 2007, pp. 2571–2575.

[8] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. R. Rodriguez, "Is High-Quality VoD Feasible Using P2P Swarming?" in *Proc. Int. Conf. on World Wide Web*, May 2007, pp. 903–912.

[9] "UUSee," http://www.uusee.com.

[10] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang, "Challenges, Design and Analysis of a Large-Scale P2P-VoD System," in *Proc. ACM SIGCOMM'08*, Aug. 2008, pp. 375–388.

[11] T. Ho, R. Koetter, M. Médard, M. Effros, J. Shi, and D. Karger, "A Random Linear Network Coding Approach to Multicast," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4413–4430, Oct. 2006.

[12] P. Chou, Y. Wu, and K. Jain, "Practical Network Coding," in *Proc. Annual Allerton Conf. on Comm., Control, and Comput.*, Oct. 2003, pp. 40–49.

[13] H. Shojania and B. Li, "Parallelized Progressive Network Coding with Hardware Acceleration," in *Proc. IWQoS'07*, Jun. 2007, pp. 47–55.

[14] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Trans. Multimedia*, vol. 9, no. 8, pp. 1672–1687, Dec. 2007.

[15] B. Cheng, X. Liu, Z. Zhang, H. Jin, L. Stein, and X. Liao, "Evaluation and Optimization of a Peer-to-Peer Video-on-Demand System," *J. Syst. Archit.*, vol. 54, no. 7, pp. 651–663, Jul. 2008.

[16] T. Ho, M. Médard, J. Shi, M. Effros, and D. Karger, "On Randomized Network Coding," in *Proc. Annual Allerton Conf. on Comm., Control, and Comput.*, Oct. 2003, pp. 11–20.

[17] D. Silva, W. Zeng, and F. Kschischang, "Sparse Network Coding with Overlapping Classes," in *Proc. NetCod'09*, Jun. 2009, pp. 74–79.

[18] C. Fragouli, D. Lun, M. Médard, and P. Pakzad, "On Feedback for Network Coding," in *Proc. CISS'07*, Mar. 2007, pp. 248–252.

[19] J. Sundararajan, D. Shah, M. Médard, M. Mitzenmacher, and J. Barros, "Network Coding Meets TCP," in *Proc. INFOCOM'09*, Apr. 2009, pp. 280–288.