On Large-Scale Peer-to-Peer Streaming Systems with Network Coding

Chen Feng, Baochun Li*
Dept. of Electrical and Computer Engineering
University of Toronto
Toronto, ON M5S 3G4, Canada
{cfeng,bli}@eecg.toronto.edu

ABSTRACT

Live peer-to-peer (P2P) streaming has recently received much research attention, with successful commercial systems showing its viability in the Internet. Nevertheless, existing analytical studies of P2P streaming systems have failed to mathematically investigate and understand their critical properties, especially with a *large scale* and under *extreme* dynamics such as a *flash crowd* scenario. Even more importantly, there exists no prior analytical work that focuses on an entirely new way of designing streaming protocols, with the help of *network coding*. In this paper, we seek to show an in-depth analytical understanding of fundamental properties of P2P streaming systems, with a particular spotlight on the benefits of network coding. We show that, if network coding is used according to certain design principles, *provably* good performance can be *guaranteed*, with respect to high playback qualities, short initial buffering delays, resilience to peer dynamics, as well as minimal bandwidth costs on dedicated streaming servers. Our results are obtained with mathematical rigor, but without sacrificing realistic assumptions of system scale, peer dynamics, and upload capacities. For further insights, streaming systems using network coding are compared with traditional pull-based streaming in large-scale simulations, with a focus on fundamentals, rather than protocol details. The scale of our simulations throughout this paper exceeds $200,000$ peers at times, which is in sharp contrast with existing empirical studies, typically with a few hundred peers involved.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; C.4 [**Performance of Systems**]: Design Studies

General Terms

Algorithm, Design, Performance

*This work was supported in part by Bell Canada through its Bell University Laboratories R&D program.

MM'08, October 26–31, 2008, Vancouver, British Columbia, Canada.

Keywords

Peer-to-Peer Streaming, Performance Analysis, Network Coding

1. INTRODUCTION

There is no need to convince any reader in multimedia networking systems that live peer-to-peer (P2P) multimedia streaming systems over the Internet are widely deployed with commercial success in real-world applications. The essential advantage of peer-to-peer streaming is to dramatically increase the number of peers a streaming channel may sustain with dedicated streaming servers. Intuitively, as participating peers contribute their upload bandwidth capacities to serve other peers in the same channel, the load on dedicated streaming servers is significantly mitigated.

Even with wide research attention, however, a number of fundamental performance metrics that characterize "good" P2P streaming systems are not yet well understood. *First,* if media segments do not arrive in a timely fashion, they have to be skipped at playback, which degrades the *playback quality*. How do we maintain a high playback quality at all participating peers? *Second,* an *initial buffering delay* must be experienced by a peer when it first joins or switches to a new channel. How do we improve user experience with the shortest initial buffering delay? *Third,* how do we encourage maximum bandwidth contribution from participating peers, which in turn minimizes *server bandwidth costs* — a sizable operational expense? *Finally* but not the least important, how do we design a system that *scales* well to accommodate a large flash crowd and a high degree of peer dynamics?

At first glance, designing a new P2P streaming system that enjoys good performance with respect to all of these performance metrics may appear too good to be true. Our rescue comes from the use of *network coding*. *How would network coding be applied to P2P streaming?* In traditional P2P streaming protocols, the media stream to be served is divided into *segments,* such that they can be better exchanged among peers. In a new protocol that uses network coding, each segment is further divided into m *blocks* $[b_1, b_2, \ldots, b_m]$. Whenever a peer is to serve a segment to another peer, it sends linear combinations of blocks within this segment with random coefficients. After m linearly independent combinations of blocks in a segment have been received, the original segment can be reconstructed using Gaussian elimination.

How would network coding be helpful in P2P streaming? In traditional pull-based streaming protocols, a segment can only be served by one upstream peer at a time. With the help of network coding, however, a missing segment on a downstream peer can be served by *multiple* upstream peers simultaneously. This allows the use of much larger segments, leading to smaller segment availability bitmaps to be exchanged. More importantly, this also implies that peers may be able to serve more, as a peer has an opportu-

nity to serve others as long as it possesses any segment that is not completely received by its neighbors. With peers contributing more upload bandwidth, less server bandwidth is consumed.

In this paper, we seek to mathematically analyze and understand new streaming systems with network coding, with a focus on playback quality, initial buffering delays, server bandwidth costs, as well as extreme peer dynamics. Rather than protocol details, we focus on the fundamental properties that network coding helps achieve. Rather than static systems that previous papers attempted to analyze, we focus on highly dynamic systems, including flash crowds. Rather than typical simulations with hundreds of peers involved, our simulations throughout the paper reach $200,000$ peers at times, and routinely exceed $10,000$ peers.

We identify several simple design principles for streaming systems with network coding and show that any streaming protocol following our design principles is sufficient to achieve *provably* good overall performance in realistic settings. In particular, we have proved that our design principles naturally lead to a near-optimal streaming rate and very short initial buffering delays during flash crowds. We also show that a reasonable server bandwidth cost is enough to handle even the most volatile peer dynamics with ease when adopting our design principles. To our knowledge, there has been no existing work in the literature that provides a thorough analytical understanding of highly dynamic P2P streaming systems with large scales, with or without network coding.

The remainder of the paper is structured as follows. In Sec. 2, we highlight our original contributions in the context of related work. Sec. 3 discusses our design principles for streaming systems with network coding, and develops the theoretical framework in realistic settings. Sec. 4 presents our main analytical results with a focus on the provably good performance with network coding. Sec. 5 conducts a series of large-scale simulations to compare our protocols with traditional pull-based streaming protocols. Finally, Sec. 6 concludes the paper.

2. RELATED WORK

With a large number of P2P streaming protocols proposed, they generally fall into two strategic categories. *Tree-based push* strategies (*e.g.*, [14]) organize participating peers into one or more multicast trees, and push streaming content along these trees. *Mesh-based pull* strategies (*e.g.*, [18]), instead, organize peers into a directed graph referred to as a *mesh*, in which segment availability bitmaps of streaming buffers are periodically exchanged among neighboring peers. Compared to tree-based strategies, mesh-based pull strategies are more resilient to peer dynamics and simpler to implement. However, such resilience is achieved at the cost of increased delay of distributing live content from servers to all participating peers, due to delays caused by periodic exchanges of segment availability [17]. Nevertheless, all real-world systems (*e.g.*, PPLive) are implemented using mesh-based pull strategies, mostly due to its simplicity.

Unique in the literature, this paper *analytically* studies a design alternative to existing strategies: *network coding*. Rather than designing a particular protocol using a network coding and trying to evaluate the design with empirical studies involving hundreds of peers (as we have performed in our previous work [15, 16], as well as Annapureddy *et al.* [1] for Video-On-Demand systems), we focus on an analytical study to demonstrate provably good performance when network coding is used. Rather than developing a segment scheduling algorithm based on network coding to satisfy strict playback deadline constraints (*e.g.*, [3] in the context of VoD systems), we focus on fundamental design principles and asymptotic properties, and do not discuss protocol details on purpose.

Table 1: Theoretical Studies on P2P Streaming

	Decentralized	Dynamics	Heterogeneity	Coding
This paper	$\surd$	$\surd$	$\surd$	$\surd$
Massoulie *et al.* [13]	$\surd$	$\times$	$\surd$	$\times$
Bonald *et al.* [2]	$\surd$	$\times$	$\times$	$\times$
Zhou *et al.* [19]	$\surd$	$\times$	$\times$	$\times$
Kumar *et al.* [8]	$\times$	$\surd$	$\surd$	$\times$
Y. Liu [12]	$\times$	$\times$	$\surd$	$\times$
S. Liu *et al.* [11]	$\times$	$\times$	$\surd$	$\times$

Beyond protocol design, there exist a small number of previous papers that focused on analytical studies of P2P streaming protocols, without using network coding. Massoulie *et al.* [13] proposed a push-based streaming protocol and proved it is able to achieve the optimal streaming rate for static streaming systems with known edge or node capacities. It remains an open problem how this protocol performs in dynamic systems, which is precisely the focus of our system model in this paper. Although Massoulie *et al.* gave a rate optimality result, they did not provide any performance guarantees on the delay performance, which we believe are more important than the maximum sustainable streaming rate, as bit rates of most multimedia streams are known *a priori*. In fact, Bonald *et al.* [2] (with Massoulie as a co-author) later pointed out that the delay performance of the original Massoulie protocol is poor, and proposed several new push-based protocols in order to improve the delay performance. Bonald *et al.* proved that some of these new protocols can successfully achieve near-optimal rate and delay in static streaming systems. However, their theoretical results rely heavily on the assumption of homogeneous peer upload capacities, which we do not make in our system model. In addition, they only focused on the delay of distributing live content, whereas our work explore initial buffering delays as well.

Instead of push-based protocols (which were not widely implemented in real-world applications), Zhou *et al.* [19] studied pull-based streaming protocols, with a focus on the impact of segment selection strategies. They further proposed a mixed strategy in order to achieve both good playback quality and good delay performance for a given streaming rate. However, they again considered *static* streaming systems with *homogeneous* peer upload capacities, while we focus on highly *dynamic* systems with *heterogeneous* upload capacities. In addition, they did not investigate the bandwidth costs on dedicated streaming servers, whereas our work analytically quantifies such costs and further reveals several interesting finds.

With respect to performance bounds in centralized P2P streaming systems, Kumar *et al.* [8] studied the maximum sustainable streaming rate in dynamic mesh-based systems. Liu [12] provided the minimum delay for static mesh-based systems. Liu *et al.* [11] derived the minimum server bandwidth costs and maximum streaming rate for static tree-based systems. However, all of them critically depend on *centralized* scheduling algorithms to approach the optimal values, whereas the streaming protocols that we analyze in this paper is fully *decentralized*, while still achieving provably good performance. Intuitively, centralized protocols are impossible to implement realistically, especially in large scale systems.

Different from previous analytical papers on P2P streaming systems, as outlined in Table 1, we provide a rigorous performance analysis on new streaming systems that follow our design principles, simultaneously considering a number of performance metrics: playback quality, sustainable streaming rate, initial buffer-

ing delays, resilience to peer dynamics, as well as bandwidth costs on dedicated streaming servers. In addition, the simplicity in our design principles enables us to use a more *realistic* system model without restricting our analysis to centralized protocols. In particular, we consider highly dynamic systems with volatile peers and heterogeneous peer upload capacities.

3. STREAMING WITH NETWORK CODING

Before we venture into theoretical analysis, we first present an overview of our design principles for live P2P streaming systems with network coding. In this paper, we focus on fundamental characteristics of streaming systems using our design principles, and do not discuss protocol design details on purpose. For brevity, a streaming protocol that uses network coding following our design principles in this section is referred to as CODING.

The first motivating factor of using network coding in a new streaming system is to improve playback quality. Intuitively, as coded blocks are received from multiple senders, it is helpful to receive a media segment in a timely manner. It turns out that network coding can also help to simplify the protocol design, shorten initial buffering delays, and minimize server bandwidth costs by allowing more bandwidth contributions from peers. In this section, we are able to intuitively explain these benefits of network coding.

3.1 Random Push on a Random Mesh Structure

In traditional mesh-based pull streaming strategies (henceforth referred to as PULL for brevity), the live stream to be served is divided into *segments*, such that their availability can be better exchanged among peers (usually as a bitmap). In CODING, much larger segments are used, and each segment is further divided into m *blocks*, with a fixed number of bytes in each block. The coding operation is only performed within each segment, but not across different segments. This is primarily designed for the purpose of reducing the number of blocks to code, leading to much reduced encoding and decoding complexity.

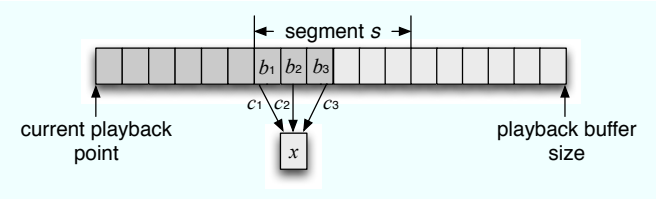


Figure 1: An example to illustrate the coding operation on peer p, where peer p has received 3 coded blocks within the segment s, and each segment consists of 6 blocks.

Suppose a peer p has received k ($k \leq m$) coded blocks within a segment s so far, denoted as $[b_1, b_2, \ldots, b_k]$. When peer p is to encode segment s for its downstream peers, as shown in Fig. 1, it independently and randomly chooses a set of coding coefficients $[c_1, c_2, \ldots, c_k]$ in a Galois field for each coded block, and then produces one coded block x in the form of $x = \sum_{i=1}^{k} c_i \cdot b_i$. Results from random network coding [7] ensure that with high probability, the coded block x is useful to such downstream peers that have not completely received the segment s (refer to Sec. 4.4 for a detailed discussion).

By sending coded blocks instead of an entire segment, multiple upstream peers may serve a missing segment on a common downstream peer simultaneously without any explicit coordination. This excellent property of cooperative transmission is illustrated in Fig. 2. In this way, peers are able to perform *push* operations on a random mesh structure, without assuming the risk of sending duplicate segments. Intuitively, such push operations not only eliminate the need of request messages, but also lead to much shorter initial buffering delays, as demonstrated by their counterpart—tree-based push operations.

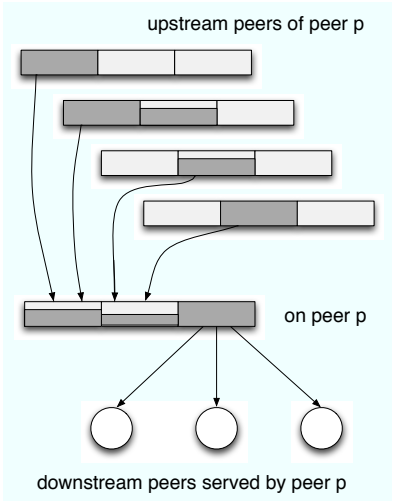


Figure 2: An illustration of random push on a random mesh structure. With network coding, multiple upstream peers are able to perform push operations on coded blocks within a segment without any explicit coordination.

From the viewpoint of a downstream peer, after it has completely received m linearly independent coded blocks within a segment, this downstream peer can successfully decode the original segment with Gaussian or Gauss-Jordan elimination. Again, in this paper, we are not concerned with the detailed design of a particular streaming protocol using network coding.

3.2 Timely Feedback from Downstream Peers

Before sending coded blocks, an upstream peer should obtain precise knowledge of the missing segments on its downstream peers at any time. This requires any peer in the system to exchange its segment availability bitmaps of streaming buffers, which are commonly referred to as *buffer maps*. In PULL, to avoid excessive overhead, these buffer maps are exchanged periodically; the more frequent such exchanges are, the closer PULL is to a streaming protocol using the tree-based push strategy. It would be ideal to send a new buffer map whenever the buffer status changes — when it has played back a segment, or when it has completed the downloading of a segment. Due to the size of such buffer maps in PULL and the frequency of buffer status changes, such a "real-time" push strategy of buffer maps leads to excessive overhead.

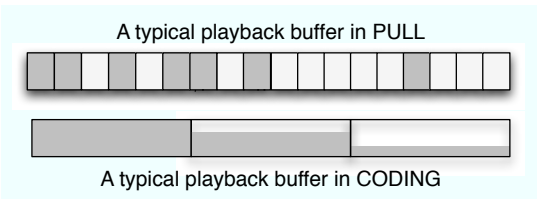


Figure 3: An illustrative comparison of playback buffers between CODING and PULL.

With network coding, however, such a "real-time" buffer map push is a feasible strategy. Since much larger segments are used,

not only the size of buffer maps is an order of magnitude smaller, but there are much fewer segments in the buffer as well. This leads to less frequent buffer status changes, as it takes much longer to finish downloading or to playback a larger segment. Such a "real-time" push of buffer maps makes it possible to *push* coded blocks to downstream peers without any explicit requests, until the segment is fully received at downstream peers.

Why is CODING able to use much larger segments than PULL? We emphasize again that a missing segment on a downstream peer in CODING can be served by multiple upstream peers, while a segment in PULL can only be served by one upstream peer. Thus, the process of receiving a larger segment is not adversely affected by the departure of a subset of upstream peers in CODING.

3.3 Synchronized Playback and Initial Buffering Delays

The use of much larger segments also makes it easier to synchronize playback buffers on all participating peers, so that all peers play the same segment at approximately the same time. One advantage of such *synchronized playback* is that, peers can help each other more effectively, as their playback buffers overlap as much as possible. With network coding, as larger segments are used, it is more important to feature synchronized playback. When a peer joins a streaming session, it first retrieves buffer maps from its neighboring peers, along with the information of the current segment being played back. The new peer then skips a few segments and *only* retrieves segments that are D seconds after the current point of playback. The peer starts playback after precisely D seconds elapsed in real time, regardless of the current status of the playback buffer. As shown in Fig. 4, the duration of D seconds corresponds to the *initial buffering delay*, which depends on the number of segments to be skipped and the current point of playback.

We naturally wish to shorten such an initial buffering delay, as it is one of the most important performance metrics that affect user experience when switching to a new channel. We thus let every peer skip only *two* segments when it first joins or switches to a new channel. In addition, we adopt a simple segment and peer selection strategy with network coding: a peer selects the most urgent segment (closest to the playback deadline) and pushes coded blocks in this particular segment to a limited number of its downstream peers. These limited number of peers are chosen uniformly at random from the downstream peers which have not fully received the particular segment. Since all upstream peers use the same strategy, a new peer who recently joined a streaming channel will naturally receive coded blocks from a large number of existing peers, saturating its downloading capacity, which leads to a shorter initial buffering delay.

3.4 System Model and Notations

Finally, for the sake of mathematical tractability, we make a few assumptions in our system model. The key notations introduced in the system model is summarized in Table 2 for easy reference. *First*, in accordance with measurement studies of existing P2P systems (*e.g.*, [6]), we assume peer upload capacities are the only bottlenecks in the streaming system. *Second*, to characterize the heterogeneity in terms of peer upload capacities, we adopt the two-class model in [8], in which peers in the system are broadly classified into two classes, with each class having approximately the same upload capacity. This assumption is reasonable as there are roughly two classes of peers in P2P streaming systems: high bandwidth Ethernet peers and low bandwidth DSL peers.

Although we assume only two classes in this paper, our analysis can easily be extended along the same lines to accommodate more classes of peer upload capacities. The upload capacity of a class-i peer is denoted as U_i ($i \in \{1, 2\}$). Without loss of generality, we assume the block size is 1 and $U_1 > U_2$. We use U_p to represent the *average* upload capacity of participating peers and u_p to represent the *ratio* of the average upload capacity U_p to the streaming rate R. Similarly, we denote by U_s the upload capacity of dedicated streaming servers and denote by u_s the ratio of the server capacity U_s to the streaming rate R.

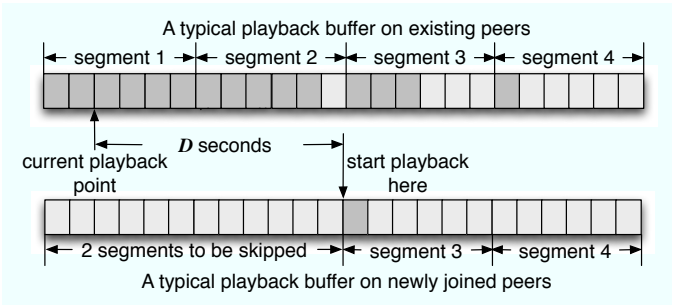


Figure 4: An illustration of initial buffering delays in CODING, which shows that the initial buffering delay on a newly joined peer is determined by the number of segments to be skipped and the current point of playback.

Table 2: Key Notations in the System Model

U_i	Upload capacity of a class-i peer (in blocks per second).
U_p	Average upload capacity of participating peers.
U_s	Server upload capacity (in blocks per second).
R	Streaming rate (in blocks per second).
D	Initial buffering delay (in seconds).
N	Scale of a flash crowd.
α	Fraction of redundant blocks induced by network coding.
m	Number of coded blocks in a segment.
u_p	Relative average peer capacity ($= U_p/R$).
u_s	Relative server capacity ($= U_s/R$).
δ	Server strength ($= \frac{U_s}{NU_p}$).

With respect to peer dynamics, we focus on two typical scenarios: the *flash crowd* scenario and *highly dynamic* scenario. During a flash crowd, most of the peers join the system in a short time period, just after a new live event has been released. In a highly dynamic scenario, peers join and leave the system in a highly volatile fashion, also referred to as *peer churn*. For a better flow of presentation, we defer detailed characterizations of peer dynamics in Sec. 4.

4. PERFORMANCE ANALYSIS OF CODING

Under our system model, we seek to investigate the overall performance of CODING. In particular, we provide quantitative answers to the following two questions:

- ▷ What are the sufficient conditions for CODING to achieve good overall performance?
- ▷ How far from optimality is the performance of CODING?

We believe such performance analysis is crucial to understand the fundamental properties and limitations of CODING, as well as to explore whether the performance gap between CODING and optimal streaming scheme is large enough to motivate more elaborated designs.

4.1 Flash Crowd Scenarios

In a flash crowd scenario, most of the peers join the system at approximately the same time. For presentation clarity, we assume time is *slotted* in the sense that it takes one time slot to playback a segment. We further assume that most peers join the system within one time slot. We emphasize here these assumptions are not necessary, and can easily be relaxed in our analysis. We now introduce the following definitions.

DEFINITION 1. *The scale of a flash crowd, denoted by N, is defined as the maximum number of peers in the system during the flash crowd.*

DEFINITION 2. *The server strength, denoted by δ, is defined as follows:*

$$\delta = \frac{U_s}{NU_p},$$

where U_p is the average upload capacity of participating peers, and U_s is the server upload capacity.

The following theorem establishes the sufficient conditions on smooth playback at a streaming rate R during any flash crowd with scale N, for given server capacity U_s and average peer capacity U_p.

THEOREM 1. *Assume that the following conditions hold:*

$$U_s + NU_p = (1+\varepsilon)NR, \tag{1}$$

$$\varepsilon = \alpha + \frac{\ln(1+\delta) - \ln\delta}{m}, \tag{2}$$

where m is the number of coded blocks in each segment, and α denotes the fraction of linearly dependent coded blocks induced by network coding (refer to Sec. 4.4 for details). Then CODING *is able to achieve perfect playback quality at a streaming rate R for any flash crowd with scale N.*

Theorem 1 implies that heterogeneity in peer upload capacities is not an issue in CODING. Intuitively, this is a consequence of the random push operations, which naturally empower high bandwidth peers to contribute more bandwidth resources. For a better flow of presentation, we defer the formal proof of Theorem 1. We now apply Theorem 1 to understand the performance gap between CODING and optimal streaming scheme in terms of the sustainable streaming rate and initial buffering delay, which leads to the following two interesting theorems.

THEOREM 2. *In terms of the sustainable streaming rate,* CODING *is within a factor of $1+\varepsilon$ of the optimal streaming scheme, where ε is given by*

$$\varepsilon = \alpha + \frac{\ln(1+\delta) - \ln\delta}{m},$$

and α is typically in the order of 0.1% (refer to Sec. 4.4 for details).

Proof: Note that the maximum bandwidth supply during a flash crowd with scale N is given by $U_s + NU_p$. This maximum bandwidth supply, if could be achieved, needs to be distributed to all participating peers in the system. Thus the maximum sustainable streaming rate $R_{\max}$ is bounded by $(U_s + NU_p)/N$, which serves as an upper bound for *any* streaming scheme.

Theorem 1 shows that CODING is able to support a streaming rate given by

$$R = \frac{U_s + NU_p}{(1+\varepsilon)N}.$$

We conclude that $R \geq \frac{R_{\max}}{1+\varepsilon}$, that is, CODING is within a factor of $1+\varepsilon$ of the optimal scheme in terms of sustainable streaming rate. □

Theorem 2 demonstrates that CODING is near-optimal in terms of sustainable streaming rate during a flash crowd. We provide a simple numerical example here. Let us set the server strength δ to 0.001, and the number of coded blocks in each segment m to 100, then we can easily calculate the sustainable streaming rate in CODING, which satisfies $R \geq \frac{R_{\max}}{1.07}$. This is due to the near-optimal bandwidth utilization enjoyed by CODING. During a flash crowd, a newly joined peer is able to effectively utilize its upload capacity immediately after receiving one or more coded blocks. However, in traditional pull-based protocols, such bandwidth utilization is impaired by the need of exchanging buffer maps and waiting for explicit requests.

THEOREM 3. *In terms of the initial buffering delay,* CODING *is within a factor of $2(1+\varepsilon)$ of the optimal streaming scheme, where ε is given by*

$$\varepsilon = \alpha + \frac{\ln(1+\delta) - \ln\delta}{m}.$$

Proof: Note that in CODING, each peer buffers at least one segment (m blocks) in order to maintain smooth playback during the flash crowd. This process takes at least $\frac{Nm}{U_s+NU_p}$ seconds, since the maximum bandwidth supply is $U_s + NU_p$. Thus the minimum initial buffering delay satisfies

$$D_{\min} \geq \frac{Nm}{U_s + NU_p},$$

which holds for *any* streaming scheme trying to buffer at least m blocks on each peer.

Now recall that in CODING, every new peer skips only two segments before the actual playback, thus the initial buffering delay satisfies

$$D \leq 2\frac{m}{R}.$$

Combining these two results with Theorem 1, we conclude that $D \leq 2(1+\varepsilon)D_{\min}$. □

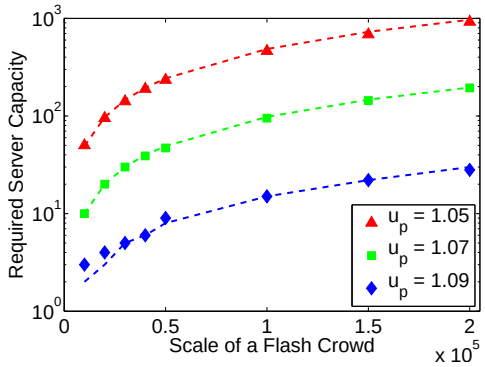
Theorem 3 shows that CODING manages to guarantee very short initial buffering delays during a flash crowd. This is in sharp contrast to PULL, which suffers from long initial buffering delays due to inherent design limitations [17]. Theorem 2 and Theorem 3, when taken together, suggest that the performance gap between CODING and optimal streaming scheme is surprisingly small with regard to user experience.

In addition to user experience, the server bandwidth cost is also an important metric, as it directly determines most of the ongoing operational expense for streaming companies. However, it is an open problem to determine the minimum server bandwidth cost during a flash crowd for mesh-based streaming systems, with or without network coding. We therefore only characterize the required server capacity in CODING without a comparison to the optimal scheme. For convenience, we use the relative upload capacity (the ratio of upload capacity to the streaming rate) in the remaining part of this section.

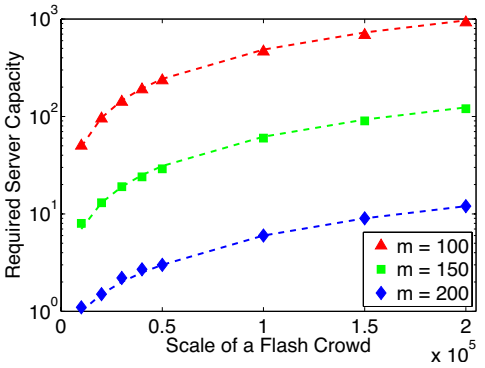
THEOREM 4. *Let u_s denote the required relative server capacity that supports perfect playback quality at a streaming rate R during a flash crowd with scale N, then $u_s = Nu_p\delta^*$, where δ^* is the minimum δ such that*

$$\big((1+\delta)u_p - 1 - \alpha\big)m \geq \ln(1+\delta) - \ln\delta,$$

and u_p is the relative average peer capacity.

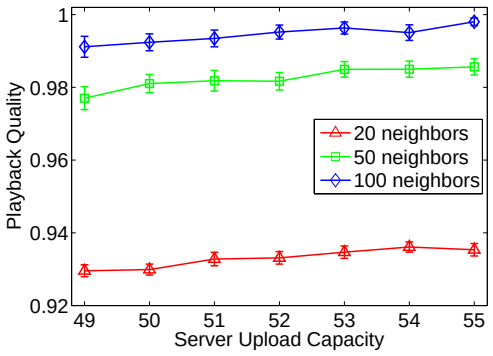


(a)

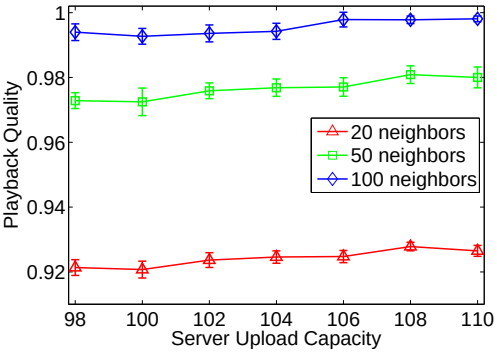


(b)

Figure 5: Validation of the required relative server capacity in several different flash crowd scenarios (theoretical results use dashed lines; simulation results use solid points). In (a), we set the number of coded blocks in a segment m to 100 and vary the relative average peer capacity u_p from 1.05 to 1.09; while in (b), we set u_p to 1.05 and vary m from 100 to 200. We can see that analytical results match simulation results quite well, especially when the system scale is larger than $100,000$.



(a) A flash crowd with scale $10,000$



(b) A flash crowd with scale $100,000$

Figure 6: Impact of restricted neighborhoods on the playback quality, with 95% confidence intervals. In (a), we set the relative average peer capacity u_p to 1.05 and the number of coded blocks in a segment m to 100; while in (b), we set u_p to 1.07 and m to 100. The required relative server capacities in these two cases are 49 and 98, respectively, under the assumption of complete graphs. By restricting the size of neighborhood, we observe that the playback quality is still close to 1, as long as the average size of neighborhood is larger than 50.

This theorem is a simple corollary of Theorem 1. We omit the proof here due to space constraints.

Fig. 5 compares the required relative server capacity obtained by Theorem 4 and by running large-scale simulations, in a number of different flash crowd scenarios. We observe that our analytical results correctly predict the required server capacity, both qualitatively and quantitatively, especially when the scale of a flash crowd is larger than $100,000$. This is because the proof of Theorem 4 requires the scale of a flash crowd N to be sufficiently large, which indicates the larger the scale, the better the prediction.

As seen from Fig. 5, the required server capacity is only about 10 times the streaming rate, in order to support a flash crowd with scale $200,000$ in the scenario of $\{u_p = 1.09, m = 100\}$ or $\{u_p = 1.05, m = 200\}$. We also observe that the required server capacity is sensitive to the system parameters u_p and m. This is again due to the near-optimal bandwidth utilization property.

So far, we have used a complete graph to represent the mesh structure in our analysis and simulations. However, a complete graph is hard to implement in practice because of overhead issues. In practical streaming systems, each peer maintains a limited number of neighbors in order to reduce the overhead of exchanging buffer maps. It is thus of great interest to investigate the impact of restricted neighborhoods.

Instead of a complete graph, we now use a random graph to model the mesh structure in our simulations. More specifically, when joining the system, each peer chooses a limited number of neighbors uniformly at random among all existing peers in the system. We let the average size of neighborhood vary from 20 to 100. We use *continuity* [19] as the metric of playback quality, which is defined as the number of peers that have successfully played the segment in each time slot divided by the total number of participating peers. Due to space limitations, we only present our simulation results of two scenarios in Fig. 6 , although we have repeated our experiments in many other scenarios and have obtained similar results. The required relative server capacities in these two scenarios, predicted by Theorem 4, are 49 and 98, respectively.

Fig. 6 suggests that the playback quality is close to 0.98, as long as each peer maintains around 50 neighboring peers, which is a substantially small number compared to the total number of participating peers. In addition, the playback quality is close to 1 (the case of complete graphs), if the size of neighborhood is increased to 100. Moreover, we observe that such a trend is insensitive to the scale of flash crowds. All of these imply that CODING only requires a small size of neighborhood in order to achieve good performance even in large scale systems. This is not a coincidence, for our proofs of Theorem 1 and Theorem 4 do not depend heavily on the complete graph assumption, as shown in Sec. 4.3.

4.2 Highly Dynamic Scenarios

Let us turn to the highly dynamic scenario, in which peers join and leave the system in a highly volatile fashion. Without loss of generality, we only focus on the system performance in current time slot. We first introduce some notations here. We denote by N_i the number of class-i peers in the system at the beginning of current time slot. To characterize peer dynamics in current time slot, we denote by A_i and W_i the number of arrivals and departures of class-i peers in current time slot. These peer dynamics would clearly degrade user experience to a certain degree. However, this effect is not a major focus in this section, as most commercial streaming systems have resorted to the over-provisioning server upload capacity to handle peer dynamics. A more interesting problem is therefore to quantify such over-provisioning server upload capacity.

Denote the most urgent segment (*i.e.*, the segment to be played in the next time slot) as segment s. First, we observe that the arrivals of A_i new peers in current time slot do not affect the playback quality of segment s, as these new peers simply skip segment s after they join the system. In contrast, the departures of W_i existing peers play a central role in the playback quality, which will be investigated in the sequel.

Recall that a class-1 peer is a high bandwidth peer, which produces more bandwidth supply than its bandwidth consumption. As a result, the earlier it leaves the system, the worse the playback quality may be. Similarly, the later a low bandwidth peer leaves the system, the worse the playback quality becomes. To summarize, the worst case in current time slot is as follows: all W_1 departures of high bandwidth peers happen at the beginning of current time slot, while all W_2 departures of low bandwidth peers occur at the end of current time slot. This observation leads to the following theorem.

THEOREM 5. *The additional server capacity, which is required to handle peer dynamics in current time slot, is strictly less than* $W_1U_1 - (1+\varepsilon)W_1R$.

Proof: On the one hand, suppose there is no peer dynamics in current time slot, then the required server capacity U_s, according to Theorem 1, is given as follows:

$$U_s + N_1U_1 + N_2U_2 = (1+\varepsilon)(N_1+N_2)R. \quad (3)$$

On the other hand, to handle the worst case peer dynamics, the required server capacity U_s' should satisfy

$$U_s' + (N_1 - W_1)U_1 + N_2U_2 = (1+\varepsilon')(N_1+N_2)R. \quad (4)$$

Notice that $\varepsilon > \varepsilon'$. Combining this with (3) and (4), we obtain

$$U_s' - U_s < W_1U_1 - (1+\varepsilon)W_1R.$$

That is, the additional server capacity is strictly less than $W_1U_1 - (1+\varepsilon)W_1R$. □

Theorem 5 gives an upper bound for the additional server capacity to handle peer dynamics. For convenience, we restate Theorem 5 in terms of relative capacity. That is, the relative additional server capacity is strictly less than $W_1u_1 - (1+\varepsilon)W_1$, where u_1 is the *ratio* of U_1 to the streaming rate R. Simulation results are shown in Table 3. We use a relative average peer capacity of 1.08 to represent the case where bandwidth supply outstrips demand, and a relative average peer capacity of 1.02 to represent an approximate match between the supply and demand. We observe our theoretical bound matches simulation results well when bandwidth supply barely exceeds bandwidth demand ($u_p = 1.02$), while the bound turns out to be loose when supply outstrips the demand ($u_p = 1.08$).

Table 3: Theoretical and simulation results for relative additional server capacity to handle peer dynamics in the worst case. We can see that the theoretical bound is tight when bandwidth supply barely exceeds bandwidth demand, while the bound is loose when supply outstrips demand. In our simulations, we set the system scale N to $10,000$, the ratio u_1 to 2, and the ratio u_2 to 0.5.

$\frac{W_1}{N_1}$	$u_p = 1.02$		$u_p = 1.08$	
	Bound	Simulation	Bound	Simulation
0.01	33	22	35	2
0.02	67	46	71	4
0.03	100	69	106	7
0.04	133	94	142	11
0.05	166	119	177	16

After performing the worst case analysis as above, we now turn to a study of the average case, where peer departures may occur at any time in current time slot, rather than only at the beginning or at the end. We run a set of simulation-based experiments by varying the total number of peer departures in current time slot. We use the term *churn rate* to represent the ratio of the number of peer departures to the total number of participating peers (set to $10,000$ in our simulation). The effect of churn rate is shown in Table 4. We observe that as churn rate increases, the additional required server capacity also increases. Moreover, CODING can survive a churn rate of 50% using an additional capacity of less than 10, which is in sharp contrast with the situation in the worse case analysis. The intuition is that, the high bandwidth peers, if not leaving the system at the very beginning of current time slot, are still able to contribute more bandwidth resources than what they have consumed, thereby mitigating the load on streaming servers.

Table 4: Simulation results for relative additional server capacity to handle peer dynamics in the average case. We can see that only a small amount of additional server capacity is required, even when 50% peers leave the system. We set the system scale N to $10,000$ in our simulations.

churn rate	10%	20%	30%	40%	50%
additional capacity	1	2	4	6	8

4.3 Formal Proof of Sufficient Conditions

We now proceed to present a formal proof of Theorem 1, which is instrumental to establish an in-depth understanding of CODING. Let u_i be the *ratio* of the upload capacity of a class-i peer U_i to the streaming rate R. Let N_i be the number of class-i peers during a flash crowd. Clearly, we have $\sum_i N_i = N$ and the relative average peer capacity $u_p = (N_1u_1 + N_2u_2)/N$, where N is the scale of the flash crowd.

Recall that the most urgent segment is denoted as segment s. We define a peer as a "working" peer if it has received one or more coded blocks within segment s and as an "idle" peer otherwise. We denote X_i as the number of class-i peers in the "idle" state. We further denote by W the number of peers that have completely received segment s. Recall that each working peer, when sending a coded block, selects a downstream peer uniformly at random from those that have not completely received segment s. Thus, the probability that an idle peer of class-i has been chosen by any working peer is given by $X_i/(N-W)$. This defines a continuous-time random process $\{X_i\}_{i\in\{1,2\}}$ with transition rates:

$$X_i : k \to k-1 \quad \text{at rate } \frac{X_i}{N-W}\big(u_s + \sum_j u_j(N_j - X_j)\big).$$

Here we need to make a technical assumption in order to apply this random process. That is, the transfer time of a block between two peers or a server and a peer is exponentially distributed, with the mean corresponding to the upload capacity of the sender. However, this random process is hard to deal with due to the complicate interaction between W and $\{X_i\}$. We instead construct a new continuous-time Markov process $\{Z_i(t)\}_{i\in\{1,2\}}$ with different transition rates:

$$Z_i : k \to k-1 \quad \text{at rate } \frac{Z_i}{N}\big(u_s + \sum_j u_j(N_j - Z_j)\big),$$

which is used to "bound" the original process $\{X_i(t)\}_{i\in\{1,2\}}$. Standard coupling arguments [10] yield $X_i(t) \leq_{st} Z_i(t)$ for all $t \geq 0$, when starting from the same initial conditions; here, $X_i(t) \leq_{st} Z_i(t)$ denotes that $Z_i(t)$ stochastically dominates $X_i(t)$. This implies that

$$E[X_i(t)] \leq E[Z_i(t)], \text{ for all } t \geq 0,$$

and

$$\Pr\{X_i(t) > k\} \leq \Pr\{Z_i(t) > k\}, \text{ for all } t \geq 0.$$

We analyze $\{Z_i(t)\}_{i\in\{1,2\}}$ under a *large population* asymptotic regime. Note that this is a *density dependent jump Markov process* [9]. Under technical assumptions that are trivially verified, we know that the rescaled process $N^{-1}Z_i(t)$ converges almost surely to the solutions of the following system of differential equations [5]:

$$\frac{\mathrm{d}z_i(t)}{\mathrm{d}t} = -\frac{u_s}{N}z_i(t) - z_i(t)\sum_j u_j\big(n_j - z_j(t)\big), \qquad (5)$$

with initial condition $z_i(0) = n_i$, where $n_i = \frac{N_i}{N}$.

Observe that the solutions of differential equations (5) are given as follows:

$$z_i(t) = \frac{n_i}{\frac{1}{1+\delta} + \frac{\delta}{1+\delta}e^{(1+\delta)u_p t}},$$

where $u_p = n_1u_1 + n_2u_2 = \frac{N_1u_1+N_2u_2}{N}$ is the relative average peer capacity, and $\delta = \frac{u_s}{Nu_p}$ is the server strength. From this, we can establish the following result:

$$X_i(t) \leq \frac{N_i}{\frac{1}{1+\delta} + \frac{\delta}{1+\delta}e^{(1+\delta)u_p t}},$$

which holds with high probability for a sufficiently large N.

Now we are ready to verify the sufficient conditions for smooth playback during a flash crowd. Without loss of generality, we only need to make sure that the most urgent segment s has been successfully received at all participating peers by the end of current time slot. To this end, we count the number of coded blocks to ensure smooth playback of segment s with the maximum supply of coded blocks within segment s in current time slot. On the one hand, as we will see in Sec. 4.4, the expected number of coded blocks to achieve smooth playback is given by $N(1+\alpha)m$. On the other hand, the maximum supply of coded blocks within segment s in current time slot is given as follows:

$$\begin{aligned} & u_s m + \sum_i \int_0^m u_i(N_i - X_i(t))\mathrm{d}t \\ = \;& u_s m + Nu_p m - \sum_i \int_0^m u_i X_i(t)\mathrm{d}t, \end{aligned}$$

which can be achieved when all the working peers and the streaming servers are serving segment s in current time slot. Notice that

$$\begin{aligned} & \sum_i \int_0^m u_i X_i(t)\mathrm{d}t \\ \leq \;& \sum_i \int_0^m \frac{u_i N_i}{\frac{1}{1+\delta} + \frac{\delta}{1+\delta}e^{(1+\delta)u_p t}} \\ = \;& u_p \int_0^m \frac{N}{\frac{1}{1+\delta} + \frac{\delta}{1+\delta}e^{(1+\delta)u_p t}} \\ = \;& N(1+\delta)u_p m \\ & -N\ln\Big(\frac{1}{1+\delta} + \frac{\delta}{1+\delta}e^{(1+\delta)u_p m}\Big) \\ \leq \;& N(1+\delta)u_p m - N\ln\Big(\frac{\delta}{1+\delta}\Big) \\ & -N\ln\Big(e^{(1+\delta)u_p m}\Big) \\ = \;& N\ln(1+\delta) - N\ln\delta. \end{aligned}$$

It follows that the maximum supply of coded blocks within segment s is no less than $N(1+\delta)u_p m + N\ln\delta - N\ln(1+\delta)$.

The sufficient conditions in Theorem 1 can be rewritten as

$$N(1+\delta)u_p m + N\ln\delta - N\ln(1+\delta) = N(1+\alpha)m,$$

which guarantees the maximum supply of coded blocks within segment s is no less than the total demand of coded blocks to support smooth playback. This completes the proof of Theorem 1.

4.4 On the Fraction of Redundant Blocks

When network coding is employed, a peer may receive redundant (linearly dependent) coded blocks from its upstream peers, leading to a waste of bandwidth resources. We denote by α the fraction of linearly dependent coded blocks in CODING. We are interested in an estimation of α, which is critical as we evaluate the bandwidth utilization of CODING.

Without loss of generality, we consider a downstream peer d to which multiple upstream peers are serving the same segment simultaneously. The following lemma states that with high probability, any coded block from an upstream peer is useful to peer d, as long as the space spanned by the coded blocks on the upstream peer is not a subspace of the space spanned on peer d.

LEMMA 1. *(Lemma 2.1, [4]) Let S_d denote the space spanned by the coded blocks on peer d and S_v denote the space spanned on one of peer d's upstream peer, namely peer v. Consider a coded block x sent from peer v to peer d. Then,*

$$\Pr(\textit{coded block } x \textit{ is useful } | S_v \subsetneq S_d) \geq 1 - \frac{1}{q},$$

where q is the size of the Galois field.

Assume that the probability of the event $\{S_v \subseteq S_d\}$, denoted as p, is the same for all upstream peers of peer d. We shall approximate the block accumulating process on peer d as follows. With probability $1-p$, an upstream peer v is helpful to peer d (*i.e.*, $S_v \subsetneq S_d$). With probability $1 - \frac{1}{q}$, a helpful upstream peer sends a useful coded block to peer d.

PROPOSITION 1. *In the simple model, the expected fraction α of redundant coded blocks is given as follows:*

$$\alpha = \frac{1}{(1-p)(1-\frac{1}{q})} - 1.$$

Proof: Let Y_i be the indicator function of the event that the ith coded block is useful to peer d. Clearly, Y_i is a binary random variable with $\Pr(Y_i = 1) = (1-p)(1-\frac{1}{q})$. Let M denote the number of coded blocks needed for peer d to successfully decode the original segment. We can then write

$$Y_1 + Y_2 + \cdots + Y_M = m.$$

Note that the random variable M is a *stopping rule* for $\{Y_n : n \geq 1\}$. Using Wald's equality, we have

$$E[M] = \frac{m}{E[Y_i]} = \frac{m}{(1-p)(1-\frac{1}{q})}.$$

The expected fraction α of redundant coded blocks is then given by

$$\alpha = \frac{E[M]-m}{m} = \frac{1}{(1-p)(1-\frac{1}{q})} - 1. \quad \square$$

Proposition 1 states that there are two reasons why peers might send redundant coded blocks. First, the randomized encoding algorithm on an upstream peer does not take into account the coded blocks accumulated on its downstream peers, and thus inevitably produces some redundant coded blocks due to this blind operation. Such redundancy is closely related to the size of the Galois field q. The larger the size, the less the redundancy. The second cause is the rare event that an upstream peer has no innovative coded blocks for its downstream peers. The probability of such event is substantially small, as the random push operations naturally create sufficient diversity.

If we choose the probability p of such event to be 0.1% and the size q to be 256, then the fraction α is less than 0.5% according to Proposition 1. To validate our estimation of α, we take advantage of Galois field computation functions in MATLAB for encoding and decoding implementation. We only present simulation results for the flash crowd scenario here, as similar results have been obtained for the highly dynamic scenario. We let the size q vary from 32 to 512. The results are shown in Table 5. As expected, the fraction of redundant coded blocks α decreases with the size q. Moreover, α is less than 0.3% in all cases, which agrees with our analysis well.

Table 5: Simulation results for fraction α of redundant coded blocks. We can see that the fraction of redundancy induced by network coding is in the order of 0.001, even when the field size q is as small as 64.

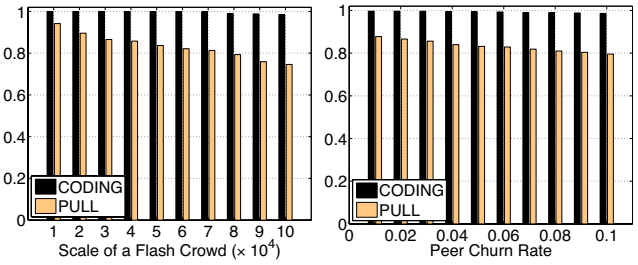
size q	32	64	128	256	512
redundancy α	0.0024	0.0016	0.0014	0.0013	0.0012

We have also investigated the impact of the scale of a flash crowd. We have observed that the fraction α of redundant coded blocks decreases as the scale increases, which suggests large-scale systems often create greater diversity, which in turn further reduces the redundancy.

5. COMPARISON WITH PULL

To further evaluate the performance of streaming protocols with network coding (CODING) in a comparison study with traditional mesh-based pull protocols (PULL), we have implemented an event-driven simulation tool in C to conduct a series of large-scale simulations. The simulation tool models many important peer characteristics and strategies, such as peer joining and leaving, segment selection, peer selection, and buffer map exchanging. For the sake of scalability to a large scale, we intentionally choose not to model P2P streaming systems in the finest details. Instead, our simulation tool captures a carefully selected set of important properties and strategies that we focus on in this paper, such as peer arrivals and departures, heterogeneous peer upload capacities, and buffer map exchanges . Its internal data structures are specifically tailored to scale well to a large number of peers within a reasonable simulation time, and it is used throughout this paper for the purpose of simulations.

We use a random graph to represent the mesh structure formed by participating peers, and the average size of neighborhood is set to 50. The duration of each segment in CODING is set to 5 seconds, which is further divided into 100 blocks, while the duration of each segment in PULL is set to 1 second. To simulate the heterogeneity of peer upload capacities, we use three types of peers, whose capacities are 3 Mbps, 384 Kbps and 128 Kbps, respectively. The streaming rate is set to 300 Kbps. By adjusting the fractions of different types of the peers, we obtain several different average upload capacities. For instance, a combination of 10% 3 Mbps peers, 38% 384 Kbps peers, and 52% 128 Kbps peers leads to an relative average peer capacity of 1.05. Unless otherwise specified, the number of participating peers is set to $10,000$. We also scale to $100,000$ in some simulation scenarios.



(a) (b)

Figure 7: A comparison of playback quality between CODING and PULL under different peer dynamic scenarios. In (a), we set the relative server capacity u_s to 50 and the relative average peer capacity u_p to 1.08; In (b), we set u_s to 60 and u_p to 1.05.

We compare the performance of CODING and PULL in terms of playback quality and initial buffering delays, under several different scenarios of peer dynamics. We first consider the flash crowd scenario. As shown in Fig. 7(a), the playback quality in CODING degrades very slowly with the scale of a flash crowd. Moreover, perfect playback quality is achieved if the scale of a flash crowd is less than $70,000$. In contrast, the playback quality in PULL is much lower than that in CODING. Further, it deteriorates significantly as the scale of a flash crowd increases. This is because newly joined peers in PULL are not able to utilize their upload capacities very effectively. Before a newly joined peer serves a segment, it has to inform its neighboring peers by exchanging buffer maps and to wait for the explicit requests. Moreover, a certain amount of segments may arrive after their playback deadlines due to these interactions, resulting in degraded playback quality.

We now turn to the highly dynamic scenario. To decouple the effect of system scale, we let the arrivals of new peers approximately match the departures of existing peers in each time slot. Fig. 7(b) shows the playback quality of both CODING and PULL under different peer churn rates (*i.e.,* the ratio of the number of departures in each time slot to the system scale, as defined in Sec. 4.2). As expected, peer churn has little impact on the playback quality in CODING, since the average peer capacity is not significantly affected. However, PULL suffers from such peer churn due to the difficulty of finding appropriate upstream peers.

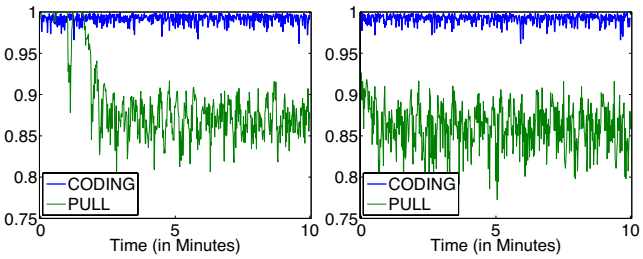
As to the initial buffering delays, we present the simulation results of PULL in Table 6, which use the same configuration as in

Table 6: Initial buffering delay (in seconds) in PULL using the same configuration as in Fig. 7(a) (denoted as *flash*) and in Fig. 7(b) (denoted as *churn*). In CODING, the initial buffering delay strictly ranges from 5 to 10 seconds regardless of peer dynamics.

flash	43	44	45	45	45	46	46	46	46	46
churn	19	22	23	26	27	27	29	32	33	35

Fig. 7. Specifically, we vary the scale of a flash crowd from $10,000$ to $100,000$ in the flash crowd scenario and the churn rate from 0.01 to 0.10 in the highly dynamic scenario. Recall that the initial buffering delay in CODING is strictly ranging from 5 to 10 seconds, regardless of peer dynamics. In Table 6, we observe that the initial buffering delay increases slowly in PULL with the scale of a flash crowd, and is substantially larger than that in CODING. In addition, peer churn affects the initial buffering delay significantly in PULL, as it usually takes much more effort for a newly joined peer to find appropriate upstream peers under a higher churn rate.

We also investigate the change of playback quality over time in both CODING and PULL. In Fig. 8, we see that the playback quality in CODING remains perfectly above 0.97 all the time and nearly 1 at most times, while PULL maintains a much lower and varied playback quality. These results show that CODING also enjoy excellent stability under peer dynamics.



(a) A flash crowd scenario (b) A highly dynamic scenario

Figure 8: The change of playback quality over time in CODING and PULL under a typical flash crowd scenario and a highly dynamic scenario. We set the relative server capacity u_s to 60 and relative average peer capacity u_p to 1.05 in both scenarios. The comparison shows that CODING has much better stability under peer dynamics than PULL.

6. CONCLUSION

In this paper, we have analytically investigated the performance of streaming systems with network coding that follow our design principles. The use of network coding not only eliminates some mathematical difficulties associated with previous theoretical models, but also leads to simple and effective streaming protocols. In particular, we have demonstrated that any streaming protocol using our design principles is sufficient to achieve provably good performance with respect to many important metrics, such as playback quality, initial buffering delay, resilience to peer dynamics, as well as bandwidth costs on dedicated streaming servers. The simplicity in the core functionalities of our design principles further allows us to use a more realistic system model, taking into account many of the essential features of P2P streaming, such as peer dynamics (peer joining and leaving) and heterogeneous peer upload capacities. With extensive large-scale simulations, we validate our analytical results and demonstrate clear advantages of network coding based protocols over traditional pull-based streaming protocols.

7. REFERENCES

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

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

[3] H.-C. Chi and Q. Zhang. Deadline-aware network coding for video on demand service over P2P networks. *J. of Zhejiang Univ. Science A*, 2006.

[4] S. Deb, M. Medard, and C. Choute. Algebraic Gossip: A Network Coding Approach to Optimal Multiple Rumor Mongering. *IEEE Trans. on Information Theory*, June 2006.

[5] S. N. Ethier and T. G. Kurtz. *Markov Processes: Characterization and Convergence*. Wiley, New York, 1986.

[6] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, Analysis, and Modeling of BitTorrent-like Systems. In *Proc. of Internet Measurement Conference*, 2005.

[7] T. Ho, R. Koetter, M. Medard, M. Effros, J. Shi, and D. Karger. A Random Linear Network Coding Approach to Multicast. *IEEE Transactions on Information Theory*, October 2006.

[8] R. Kumar, Y. Liu, and K. W. Ross. Stochastic Fluid Theory for P2P Streaming Systems. In *Proc. of IEEE INFOCOM*, 2007.

[9] T. G. Kurtz. *Approximation of Population Processes*. CBMS-NSF Regional Conf. Series in Applied Math, 1981.

[10] T. Lindvall. *Lectures on the Coupling Method*. Wiley, New York, 1992.

[11] S. Liu, R. Z. Shen, W. Jiang, J. Rexford, and M. Chiang. Performance Bounds for Peer-Assisted Live Streaming. In *Proc. of ACM SIGMETRICS*, 2008.

[12] Y. Liu. On the Minimum Delay Peer-to-Peer Video Streaming: How Realtime Can It Be? In *Proc. of ACM Multimedia*, 2007.

[13] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez. Randomized Decentralized Broadcasting Algorithms. In *Proc. of IEEE INFOCOM*, 2007.

[14] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous Unstructured Tree-based Peer-to-Peer Multicast. In *Proc. of IEEE International Conference on Network Protocols (ICNP)*, 2006.

[15] M. Wang and B. Li. Lava: A Reality Check of Network Coding in Peer-to-Peer Live Streaming. In *Proc. of IEEE INFOCOM*, 2007.

[16] M. Wang and B. Li. R^2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming. *IEEE J. on Sel. Areas in Communications*, December 2007.

[17] M. Zhang, Q. Zhang, L. Sun, and S. Yang. Understanding the Power of Pull-based Streaming Protocol: Can We Do Better? *IEEE J. on Sel. Areas in Communications*, December 2007.

[18] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming. In *Proc. of IEEE INFOCOM*, 2005.

[19] Y. Zhou, D. M. Chiu, and J. C. Lui. A Simple Model for Analyzing P2P Streaming Protocols. In *Proc. of IEEE International Conference on Network Protocols (ICNP)*, 2007.