Network Coding for Peer-Assisted Multimedia Streaming
Chen Feng and Baochun Li, University of Toronto, Canada
{cfeng, bli}@eecg.utoronto.ca

Introduction

Peer-assisted multimedia streaming has recently witnessed unprecedented growth on the Internet, delivering live streaming content to millions of users in real-world applications. The essential advantage of peer-assisted streaming is to dramatically increase the number of peers a streaming channel may sustain with a limited pool of available bandwidth provided by dedicated streaming servers. Intuitively, as participating peers contribute their own upload bandwidth to serve one another in the same channel, the load on dedicated streaming servers is significantly mitigated.

There are a number of fundamental performance metrics that characterize “good” peer-assisted streaming systems. First, if streaming content does not arrive in a timely fashion, it has to be skipped at playback, which degrades the playback quality. Second, an initial buffering delay must be experienced by a peer when it first joins or switches to a new streaming channel. Third, how do we encourage maximum bandwidth contribution from participating peers, which in turn minimizes server bandwidth costs—a sizable operational expense? Finally, how do we design a system that scales well to accommodate a large flash crowd and a high degree of peer dynamics?

Network coding [1, 2] has been originally proposed in information theory, and has since emerged as one of the promising information theoretic approaches to improve multicast session throughput. In peer-to-peer content distribution systems, Avalanche [3, 4] has demonstrated that network coding may improve the overall performance. The intuition was that, with network coding, all pieces of information are treated equally, without the need to identify and distribute the “rarest piece” first. Can network coding be useful in peer-assisted media streaming systems as well?

Compared to peer-to-peer content distribution, peer-assisted media streaming has unique timing requirements. The advantages of network coding are less obvious. In fact, it has been shown in [5] that the success story of applying network coding in content distribution cannot be simply replicated in multimedia streaming. To take full advantage of network coding, a complete redesign of peer-assisted streaming protocols is required. This motivated the design and implementation of R² [6], an entirely new streaming algorithm with network coding.

R²: random push with random network coding

A traditional peer-assisted streaming protocol—one similar to CoolStreaming [7] and PPLive [8]—utilizes a pull-based streaming mechanism (later called “pull” for ease of reference). In pull, the streaming content to be served is divided into a series of data blocks, each representing a short duration of playback. Every peer periodically exchanges block availability information (often called buffer maps in the literature) with its neighbors. Based on such information, data blocks are pulled from appropriate neighbors, in order to meet their playback deadlines. Data blocks that are not received on time are skipped during playback, leading to a degraded playback quality.

In R², the streaming content is divided into large segments, with each segment further divided into smaller data blocks. Whenever a peer is able to serve a downstream peer p, it randomly chooses a segment that p has not completely received and then sends a coded block in that segment using random network coding [2]. Since all coded blocks are equally useful, a missing segment on a peer in R² can be served by multiple neighbors simultaneously without any explicit coordination, as illustrated in Fig. 1. In this way, participating peers in R² are able to perform push rather than pull operations, thereby fully utilizing available bandwidth resources.

Before pushing coded blocks, an upstream peer should obtain the precise knowledge of the missing segments on its downstream peers at any time. This requires participating peers in the system to exchange their buffer maps in a timely fashion. Since large segments are used in R², buffer maps that indicate segment availability information—as opposed to block availability information—can be exchanged, which may be
IEEE COMSOC MMTC E-Letter

an order of magnitude smaller. In addition, it takes much longer to playback or finish downloading a large segment, leading to a less frequent need to update buffer maps. As such, $R^2$ can afford “on-demand” exchanges of buffer maps without additional overhead (or even with less overhead!); buffer maps are sent to all neighboring peers immediately upon a local update.

![Diagram of R^2](image)

Figure 1: An illustration of $R^2$.

Why is network coding helpful in $R^2$?

We intuitively explain why the use of network coding in $R^2$ helps provide a good overall performance for streaming systems. A detailed theoretical analysis can be found in [9].

First, with network coding, $R^2$ is able to use large segments and small blocks. On one hand, the use of large segments naturally leads to “on-demand” exchanges of buffer maps. With up-to-date buffer maps in $R^2$, participating peers are able to serve more to one another. In contrast, buffer maps in pull are exchanged periodically with a longer time interval, in order to avoid excessive overhead. As shown in [10], the lack of timely exchanges of buffer maps may be a major factor that separates the performance of pull from optimality.

On the other hand, with random push operations performed on small coded blocks, the probability of saturating peer upload bandwidth capacities is much higher than that in pull. In particular, even slow connections may be fully utilized in $R^2$ due to much finer granularity in data blocks, which are generally impossible in pull [11]. Both of these factors contribute to a better utilization of peer bandwidth resources, leading to a higher playback quality and reduced server bandwidth costs.

Second, with network coding, robustness to peer departures in $R^2$ has been significantly improved. Since multiple upstream peers are serving each segment at the same time, the departure of a few of them does not constitute a challenge. In contrast, a missing block in pull can only be served by one upstream peer at a time. Whenever an upstream peer leaves the system, the downstream peer has to detect such a departure and request the missing block again. If this block is close to its playback deadline, the unlucky downstream peer is indeed under the risk of missing a deadline.

Finally, with network coding, $R^2$ scales well to accommodate a large flash crowd. Due to the use of large segments, it is easier to synchronize playback buffers so that participating peers are playing the same segment at approximately the same time. With playback buffers overlapping as much as possible, newly arrived peers during a flash crowd are able to serve one another immediately after they have received a few coded blocks. This allows fully utilization of upload bandwidth from newly arrived peers, which in turn reduces initial buffering delays and improves system scalability.

Conclusion

Inspired by the success of network coding in peer-to-peer content distribution applications, it is natural to explore the potential benefits of network coding in peer-assisted media streaming applications. Due to the strict timing requirement, however, the advantage of network coding is less obvious, and would certainly justify an in-depth study. In this article, we briefly illustrate the design principles behind $R^2$, a new set of streaming protocol design principles that combine random network coding with random push operations. With its intuitive advantages, we believe $R^2$ has shown the clear potential to help improve user experience and server costs in current-generation peer-assisted streaming systems.
IEEE COMSOC MMTC E-Letter

References


Chen Feng received the B.E. degree in Electrical Engineering from Shanghai Jiao Tong University, China, in 2006, and the M.S. degree in Electrical and Computer Engineering from the University of Toronto, Canada, in 2009. He is currently working towards the Ph.D. degree at the Department of Electrical and Computer Engineering, University of Toronto. His research interests fall in the areas of network systems and coding theory, in particular the design of new network protocols by using advanced coding theory.

Baochun Li received his B.Eng. degree in 1995 from Department of Computer Science and Technology, Tsinghua University, China, and his M.S. and Ph.D. degrees in 1997 and 2000 from the Department of Computer Science, University of Illinois at Urbana-Champaign. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a Full Professor. He holds the Nortel Networks Junior Chair in Network Architecture and Services from October 2003 to June 2005, and the Bell University Laboratories Endowed Chair in Computer Science.
IEEE COMSOC MMTC E-Letter

Engineering since August 2005. In 2000, he was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the IEEE Communications Society, and a recipient of the University of Toronto McLean Award. His research interests include large-scale multimedia systems, peer-to-peer networks, applications of network coding, and wireless networks. He is a senior member of IEEE, and a member of ACM.