1 Network Coding for Content Distribution and Multimedia Streaming in Peer-to-Peer Networks

Chen Feng, Baochun Li

Department of Electrical and Computer Engineering, University of Toronto

Peer-to-peer (P2P) networks have been one of the most promising platforms to realize the potential of network coding, since end hosts (referred to as peers) at the edge of the Internet have abundant computational resources with modern processors. In this chapter, we take a journey into the application world of network coding in P2P networks, with a focus on two important applications: *content distribution* and *multimedia streaming*. For each application, we explore the possible design space of P2P systems with network coding, and provide an intuitive explanation for advantages of using network the coding technique. We further unfold our journey through a discussion of several theoretical results and practical issues.

1.1 P2P Content Distribution with Network Coding

P2P content distribution has become increasingly popular in current-generation content distribution protocols. The basic idea in P2P content distribution protocols is surprisingly simple. Consider a single server distributing a file (usually hundreds of megabytes or even gigabytes) to a large number of end hosts (peers) over the Internet. Instead of uploading the file to every individual peer, the server first divides the file into r data blocks, and then distributes these data blocks in an efficient manner by letting participating peers exchange them with one another.

The essential advantage of P2P content distribution is to dramatically reduce the file downloading time for each peer. Intuitively, as participating peers contribute their own upload bandwidth to serve one another, the aggregate upload bandwidth in the system is significantly increased, leading to a much faster file distribution process.

1.1.1 How can network coding be applied to P2P content distribution?

BitTorrent [C03] has evolved into one of the most popular P2P content distribution protocols. A detailed description of the BitTorrent system can be found in [PGES05]. In BitTorrent, when a newly arrived peer joins the system, it contacts a central *tracker* to obtain a list of some participating peers that are currently downloading the file. Typically, the tracker provides 50 peers chosen at random among all participating peers. The new peer then attempts to establish and maintain connections to about 40 peers, which become its *neighbors*. If a peer fails to maintain at least 20 neighbors (due to peer departures, for example), it contacts the tracker again to receive a list of additional peers. In this way, a dynamic logical network — namely a P2P *overlay* network — is formed by all participating peers that are currently downloading the file.

Participating peers in BitTorrent cooperate to download the file using *swarming* techniques. The file is divided into r equal-sized data blocks $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r\}$. Each peer exchanges data blocks with its neighbors, until it has obtained all r data blocks and can depart the system. Note that the server is also a neighbor of a limited number of peers. After a peer downloads a new data block, it informs all its neighbors, so that every peer in the system knows the block availability information among its neighbors. When requesting a block from a particular neighbor, a peer typically asks for a *local rarest* block, that is, a block that is least common among all its neighbors. The purpose here is to ensure data blocks are propagated almost uniformly through the overlay network. Otherwise, with some very rare blocks in the system, the downloading time for each peer may be greatly affected by such information bottleneck.

In general, a participating peer in any P2P content distribution system has to answer the following question when requesting data blocks: *which blocks should be downloaded, and from which neighbors*? Referred to as the *block scheduling* problem, this question needs to be addressed in an efficient and distributed fashion, with local information only. BitTorrent employs the *local rarest first* strategy, hoping to avoid information bottleneck. With the help of network coding, however, this question can be solved in a surprisingly simple and effective manner.

Avalanche [GR05, GMR06] represents one of such P2P content distribution protocols with network coding. In Avalanche, the file is again divided into r equal-sized data blocks. This time, each block $\boldsymbol{b}_i$ ($i = 1, \ldots, r$) is regarded as a fixed-length vector over a finite field $\mathbb{F}_q$. Rather than uploading original data blocks to its neighbors, a peer (or the server) uploads coded blocks, where each coded block is a random linear combination of the coded blocks already received by the peer. With this change, there is no need for a peer to request specific data blocks. Instead, a peer in the system blindly downloads coded blocks from its neighbors, until it is able to reconstruct the original file.

Let us use a simple example in [Y07] to better illustrate how the system works. At the beginning of the file distribution process, a peer (say peer A) contacts the server and receives a number of coded blocks. For example, the server uploads two coded blocks $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ to peer A, where

$$\boldsymbol{x}_i = \alpha_1^i \boldsymbol{b}_1 + \alpha_2^i \boldsymbol{b}_2 + \cdots + \alpha_r^i \boldsymbol{b}_r,\ i = 1, 2 \tag{1.1}$$

with α_j^i $(1 \le j \le r)$ being chosen randomly from the field $\mathbb{F}_q$. When peer A needs to serve a neighboring peer (say peer B), it simply generates a coded block $\boldsymbol{x}_3$

$$\boldsymbol{x}_3 = \alpha_1^3 \boldsymbol{x}_1 + \alpha_2^3 \boldsymbol{x}_2, \tag{1.2}$$

where α_1^3 and α_2^3 are again randomly chosen from $\mathbb{F}_q$. Substituting (1.1) into (1.2), we obtain

$$\boldsymbol{x}_3 = \sum_{j=1}^{r} (\alpha_1^3 \alpha_j^1 + \alpha_2^3 \alpha_j^2) \boldsymbol{b}_j.$$

It means that $\boldsymbol{x}_3$ (and in general every coded block in the system) is some random linear combination of the original blocks $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r\}$ and the associated vector $(\alpha_1^3 \alpha_1^1 + \alpha_2^3 \alpha_1^2, \ldots, \alpha_1^3 \alpha_r^1 + \alpha_2^3 \alpha_r^2)$ is called the *global encoding vector* for $\boldsymbol{x}_3$. Continuing this process, a peer in the system is able to collect r linearly independent coded blocks. How can a peer recover the original file from these coded blocks? As explained in Chapter 1, the global encoding vector is attached to each block as a “header” and this information is used by a peer to reconstruct the original file as long as it has received r linearly independent coded blocks.

1.1.2 Why is network coding helpful in P2P content distribution?

What are the potential benefits of network coding for P2P content distribution? Recall that the difficulty of P2P content distribution is finding an optimal block scheduling algorithm, which should minimize the file downloading time in a distributed manner. This becomes even more challenging, when the overlay network is changing dynamically.

Without network coding, each peer has to decide which blocks to download from which neighbors, based on local information only. This may be suboptimal due to lack of a global view, since local rarest blocks may not be global rarest blocks. In contrast, with network coding, all coded blocks are almost equally useful to any peer and there is no need to locate and request global rarest blocks in the system. As such, information bottleneck is avoided, which in turn reduces the file downloading time.

Another important benefit of network coding is the robustness to peer departures. Without network coding, it is possible that a few data blocks are lost, due to departures of the server as well as the peers who have these data blocks. In this unfortunate event, the original file cannot be reconstructed by the remaining peers. On the other hand, with network coding, the risk of losing certain data blocks is no longer a concern. Intuitively, since data blocks are mixed together, each of them is spreading to a large number of coded blocks in the system.

In summary, the use of network coding solves the block scheduling problem in a surprisingly simple and effective manner, leading to a shorter file downloading time. Moreover, it provides desirable robustness to peer departures. There is no need to worry about losing certain data blocks anymore. All of these contribute to the usefulness of network coding in P2P content distribution. In the following, we will present a number of theoretical results to substantiate these claimed advantages of network coding.

1.1.3 Theoretical results on P2P content distribution with network coding

Let us begin with the system model for P2P content distribution. The overlay network formed by the single server and participating peers can be modeled by a directed graph $G = (V, E)$, where $s \in V$ denotes the server, a vertex $v \in V - \{s\}$ corresponds to a participating peer, and every edge $e \in E$ corresponds to an overlay connection from one peer to another.[1]

This model is well suited for *static* P2P content distribution systems without peer arrivals and departures. However, peers may join and leave the system at any time in the file distribution process. To model such peer dynamics as well as block transmissions, the *trellis graph technique* [Y07, W06] can be applied as follows. For a directed graph $G = (V, E)$, we construct a new trellis graph $G^* = (V^*, E^*)$ with the node set

$$V^* = \big\{(i,t) : i \in V \text{ and } t \in \{t_0, t_1, t_2, \ldots\}\big\},$$

where the node $(i,t) \in V^*$ corresponds to the node $i \in V$ at time t, and the set $\{t_0, t_1, t_2, \ldots\}$ denotes all starting times and ending times for events of peer dynamics and block transmissions. The edge set E^* is determined by the strategy adopted by the server as well as by all the other nodes in V to request and upload coded blocks. Specifically, there are two types of edges in the trellis:

1. An edge $e \in E^*$ is added in the trellis graph from node $(i, t_k) \in V^*$ to node $(j, t_l) \in V^*$, where $(i,j) \in E$ and $t_k < t_l$, if a coded block is uploaded from node i to node j, starting at time t_k and ending at time t_l. This type of edges represents transmissions of coded blocks between neighboring peers.
2. Assume that node i joins the system at time t_k and leaves the system at time t_l. Then there is an edge with infinity capacity from node (i, t_m) to node (i, t_{m+1}), for all m satisfying $k \leq m \leq l-1$. This type of edges represents the accumulation of received information at node i over time.

The trellis graph model presented here is a generalization of that used in [Y07, W06]. An illustration of the trellis graph G^* up to $t \leq t_8$ is given in Figure 1.1, where the edges with capacity ∞ are lightened. Note that the trellis graph G^* is

[1] If peer i maintains a TCP connection with another peer j, then we say there is an edge from peer i to peer j. Note that an edge in G is not a physical communication link; instead, it is an abstract link that consists of a path of underlying physical links.

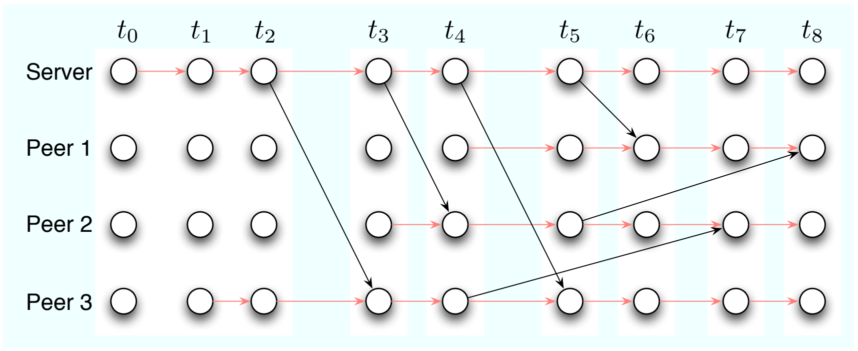

Figure 1.1 An illustration of the trellis graph G^*.

always acyclic regardless of whether the directed graph G is acyclic or not. The analysis in the sequel is a direct extension of that in [Y07].

At time t_0, the server s is ready to distribute the file consisting of r data blocks via swarming techniques with network coding as described in Section 1.1.1. This can be exactly modeled as the node $(s, t_0) \in V^*$ multicasting r data blocks to all other nodes in G^* via random linear network coding.

We are now in a position to determine the downloading time for a particular node $i \in V$. Consider the maximum flow from $(s, t_0) \in V^*$ to a node $(i, t_l) \in V^*$ in the trellis G^*, denoted by $f_i(t_l)$. When $f_i(t_l) \geq r$, the node (i, t_l) can recover the whole file with probability close to 1 under random linear network coding, provided that the field size q is sufficiently large (please refer to the discussion in Chapter 1 for details). Let $m(i)$ be the minimum value of l such that $f_i(t_l) \geq r$, that is, $m(i) = \min\{l : f_i(t_l) \geq r\}$. Then with high probability, the downloading time for node $i \in V$ is given by $t_{m(i)}$. Note that $t_{m(i)}$ is also a lower bound on the downloading time based on the information theoretic cut-set bound. This implies that the use of random linear network coding can achieve the minimum downloading time for this system. Even if the rare event indeed happens that node $i \in V$ cannot reconstruct the original file at time $t_{m(i)}$ due to linear dependence, it can eventually recover the file after receiving a few additional coded blocks. This can be proved by using a simple probabilistic argument.

The above analysis demonstrates the first major advantage offered by network coding — the peers do not need to decide which data blocks to be downloaded, while still achieve the minimum downloading time for any given strategy of requesting and uploading coded blocks. We next turn to the second major advantage of network coding — robustness to peer departures.

With some peers or even the server leaving the system, it is natural to ask whether the remaining peers are able to reconstruct the original file and finish their downloads. This important question can again be answered by using the trellis graph techniques as above. For any given time t_l, assume that we are interested in whether the remaining peers in the system can recover the original file, provided that no more peers leave the system after time t_l. Toward this end, let $R(t_l) = \{(i, t_l) : i \in V\}$ and let $f_R(t_l)$ denote the maximum flow from the node $(s, t_0) \in V^*$ to the set of nodes $R(t_l)$ in the trellis graph. Then the remaining peers can recover the whole file with high probability, provided that $f_R(t_l) \geq r$ [HKMESK06]. Notice that when $f_R(t_l) < r$, it is impossible for the remaining peers to recover the whole file even with the help of global information, due to the information theoretic cut-set bound. This implies that the use of network coding indeed provides the best possible robustness. The robustness issue also appears in peer-to-peer storage systems which provide reliable access to data through unreliable peers. The use of network coding offers similar benefits as shown in [DGWWR10, ADMK05].

Thus we see that the use of network coding in P2P content distribution achieves both minimum downloading time and maximum robustness with respect to the strategy of requesting and uploading coded blocks. For a given strategy, this is almost the best possible performance one can expect. However, one still needs to design a good strategy for peers to request and upload coded blocks. In fact, this question is closely related to fairness issues in real-world P2P content distribution, since a peer naturally prefers to help those neighbors that provide the best download rates. As such, a "tit-for-tat" strategy is used in Avalanche [GMR06] for requesting and uploading coded blocks, which aims to encourage cooperation and reduce free-riding.

On the other hand, when all participating peers are operated by the same company, fairness is not a major concern, since peers are willing to cooperate with each other even without incentive. Can network coding help in this case? Deb *et al.* [DMC06] have shown that with network coding, even a naive strategy of requesting and uploading coded blocks can achieve a shorter downloading time. More specifically, they consider a system model consisting of n participating peers and there are r data blocks to start with. Initially, every peer has only one out of the r data blocks and each data block is equally spread in the system. The goal is to distribute all r data blocks among all the peers as fast as possible. For simplicity, they assume that time is divided into slots and slots are synchronized at the various peers. During each time slot, each peer requests coded blocks from all the neighboring peers. Then each peer chooses a neighbor uniformly at random from those who sent requests and uploads a coded block using random linear network coding. In other words, a peer simply employs a random strategy to upload a coded block per time slot. The following theorem characterizes the performance of this strategy with network coding.

Theorem 1.1 ([DMC06]). *Suppose the underlying overlay network is fully connected. Suppose the field size $q \geq \max\{r, \ln(n)\}$. Let T_b be the random variable denoting the broadcasting time (the time required by all the peers to download all* *data blocks) under the system model and the strategy described as above. Then,*

$$T_b \leq 5.96r + O\left(\sqrt{r\ln(r)}\ln(n)\right), \text{ with probability } 1 - O\left(\frac{1}{n}\right).$$

Further, let T_i be the downloading time for peer i, then

$$E[T_i] \leq 5.96r + O\left(\sqrt{r\ln(r)}\ln(n)\right).$$

Observe that when $r = \Theta(n)$, it takes at least $\Theta(n)$ time slots for all the peers to download all data blocks. Thus, Theorem 1.1 implies that this simple strategy with network coding is order-optimal for $r = \Theta(n)$. They also prove that without network coding, the simple strategy performs strictly worse in terms of the broadcasting time. In other words, the use of network coding has the potential to improve system performance in an environment with full cooperation when $r = \Theta(n)$. Furthermore, their simulation results demonstrate that the benefits of network coding carry over to the case when r is small.

Previous theoretical results suggest that the use of network coding helps to reduce the broadcasting time, which corresponds to the maximum downloading time among all the peers. It might be natural to ask: what about other functions of downloading time, for example, the average downloading time? In fact, it is shown in [WHLC09] that, given an order at which the peers finish downloading, the use of network coding achieves any point in the optimal delay region and in particular the average downloading time. This result is partially extended to dynamically changing network scenarios [CHEML10] in which network coding is shown to provide a robust solution that outperforms routing.

We have presented many advantages of network coding in P2P content distribution. However, these advantages do not come without a price. In fact, additional computational resources are required compared with traditional approaches. The next question naturally arises: can we reduce the computational cost without significant performance loss?

In [CWJ03], Chou *et al.* has proposed the concept of *group network coding*, in which a file is divided into equal-sized *segments* (also referred to as *generations*) with each segment further divided into equal-sized data blocks. The coding operation is performed on the blocks within the same segment, but not across different segments. Though group network coding has a potential to reduce computational complexity to a large degree, its effects on the file downloading time and robustness to peer departures are not clear at first glance.

It has been shown in [MHL06] that the use of group network coding is still able to achieve minimum file downloading time with high probability for any given strategy of requesting and downloading coded blocks. Moreover, the computational cost for decoding group network codes can be further reduced with a precoding technique, similar to the one used in Raptor codes (see [MHL06] for details). In other words, a significant number of computational operations can be saved without noticeable loss in downloading time.

We next turn to the effect of group network coding on robustness to peer departures. Intuitively, the robustness degrades as the number of segments increases, since each segment contains fewer data blocks and it is likely that all of them are lost with peer departures. On the other hand, a small number of segments brings a high degree of robustness, but little saving of computational cost. In other words, there exists a fundamental robustness-complexity tradeoff of network coding in P2P content distribution. It would be best if one can operate at a "sweet spot" to enjoy most of robustness with reasonable computational cost. In [NL07], this robustness-complexity tradeoff is quantitatively characterized, providing a theoretical guideline for choosing such a "sweet spot".

To summarize, network coding makes optimal use of available overlay connections, without the need for sophisticated block scheduling algorithms. At the same time, it provides best possible robustness, even if a number of peers leave the system suddenly. Moreover, if coding operation is applied within each segment rather than the whole file, the computational cost can be greatly saved without significant performance loss.

1.1.4 Practical aspects of P2P content distribution with network coding

The price of network coding is mostly the computational cost, which, however, may require prohibitive computational resources. As discussed in Section 1.1.3, the use of group network coding can reduce the computational cost without noticeable performance loss. However, there is a tradeoff between complexity and performance in general. So a question of practical interest may be: What is the maximum computational cost one may afford with modern processors? Or what are the possible configurations with respect to the number of blocks and the block size that are affordable in practice? This question has been addressed in [WL06] with a high-performance implementation of random linear network coding at the application layer. A number of important observations have been made in [WL06] with a brief summary here.

1. The number of data blocks in each segment should be less than few hundreds with block size less than few megabytes, in order to ensure affordable encoding and decoding.
2. The number of data blocks in each segment matters more than the block size. This suggests the use of a small number of data blocks in each segment (such as 100).
3. The optimal block size is around $2 - 32$ KB, which provides fastest encoding and decoding. This optimal value increases with the number of data blocks in each segment.

In fact, each segment in Avalanche contains 80 data blocks, agreeing with our second observation. Each data block in Avalanche has approximately 2.3 MB, which is reasonable as suggested in our first observation. Why doesn't Avalanche further reduce the block size? This is partially due to slow connections between participating peers. With slow connections, uploading and downloading speeds become the bottleneck rather than encoding and decoding speeds. In addition, a larger block size means less overhead in terms of the "header", which is also desirable in practice.

Another practical concern about network coding is the protection against malicious peers. In fact, even a single corrupted block — injected by a malicious peer — has the potential to pollute a large number of coded blocks and prevent participating peers from decoding. Can we detect or even correct corrupted coded blocks? In Avalanche, the use of *secure random checksums* [GMR06] provides an on-the-fly detection of corrupted coded blocks at a low computation cost. Another solution to address malicious attacks is to use network error-correcting codes [KK08], which itself becomes a new research area for network coding.

1.2 P2P Multimedia Streaming with Network Coding

Peer-to-peer (P2P) 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 P2P streaming is to dramatically increase the number of peers a streaming channel may sustain with 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" P2P streaming systems. Let us visit a few of them as examples. With respect to *playback quality*, if streaming content does not arrive in a timely fashion, it has to be skipped at playback, leading to a degraded playback quality. How do we consistently maintain a high playback quality at all participating peers? With respect to the *initial buffering delay*, which is experienced by a peer when it first joins or switches to a new streaming channel, how do we improve user experience with shortest possible buffering delay? With respect to *server bandwidth costs*, how do we minimize such costs by maximizing bandwidth contribution from participating peers? Last 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?

These performance metrics should be given priority when evaluating a protocol that is designed specifically for P2P multimedia streaming. The playback quality and the initial buffering delay matter most to the user experience, which determines the level of user satisfaction. The server bandwidth costs, however, matter most to the companies in operation, as they directly determine most of the ongoing operational costs. In the following, we will demonstrate how the use of network coding can help to design P2P multimedia streaming systems that enjoy good overall performance with respect to all of these performance metrics.

1.2.1 How can network coding be applied to P2P multimedia streaming?

With a large number of P2P multimedia streaming protocols proposed, they generally fall into two strategic categories. *Tree-based push* streaming strategies (*e.g.*, [VYF06]) organize participating peers into one or more multicast trees, and distribute streaming content along these trees. In contrast, *mesh-based pull* streaming strategies organize peers into a random mesh structure, with each peer having a random subset of participating peers as its neighbors. The streaming content is divided into a series of data blocks, each representing a short duration of playback. Each peer maintains a *playback buffer* that consists of data blocks to be played in the immediate future. Every peer *periodically* exchanges block availability information of playback buffers (often referred to as *buffer maps*) 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 in time are skipped during playback, leading to degraded quality.

Compared to tree-based push strategies, mesh-based pull strategies eliminate the need to construct and maintain multicast trees, which may be difficult in practice especially when peers join and leave the system frequently. This makes them enjoy the advantages of simplicity and robustness to peer dynamics, which are in fact inherited from the design philosophy of BitTorrent-like content distribution systems. For this reason, most real-world P2P multimedia streaming systems, such as CoolStreaming [XLKZ07] and PPLive [HFCLH08], are implemented using mesh-based pull strategies.

In such a streaming system, each participating peer has to decide *which data blocks to download from which neighbors*. This question is similar to the block scheduling problem in BitTorrent-like systems, but with quite a different purpose. Instead of minimizing the file downloading time, the main purpose here is to minimize the playback skips, thereby improving the playback quality. For a newly arrived peer, a short initial buffering delay is also an important concern in the design of block scheduling algorithms.

Inspired by the success of network coding in solving the block scheduling problem for content distribution systems, one may conjecture that a similar approach may work well for multimedia streaming systems as well. In fact, Wang and Li [WL07] have evaluated the effectiveness of applying network coding in P2P multimedia streaming, by replacing traditional block scheduling algorithms with a group network coding scheme in an experimental testbed. It has been discovered that network coding provides some marginal benefits when the overall bandwidth supply barely exceeds the demand, or when peers are volatile with respect to their arrivals and departures.

With such mildly negative results against the use of network coding, one may argue that the advantages of network coding may not be fully exploited by simply replacing block scheduling algorithms in traditional mesh-based pull protocols. This motivates a complete redesign of P2P multimedia streaming with network coding. Indeed, Wang and Li have proposed R^2 in [WLi07] — a new streaming algorithm designed from scratch to take full advantage of network coding. Here we present a brief overview of the design principles in R^2.

Random Push on a Random Mesh Structure

In traditional mesh-based pull streaming protocols (henceforth referred to as PULL for brevity), the streaming content to be served is divided into a sequence of equal-sized data blocks. In R^2, the streaming content is first divided into a sequence of equal-sized segments, and each segment is further divided into k equal-sized data blocks. The coding operation is only performed within each segment, but not across different segments. This is again for the purpose of reducing the computational cost.

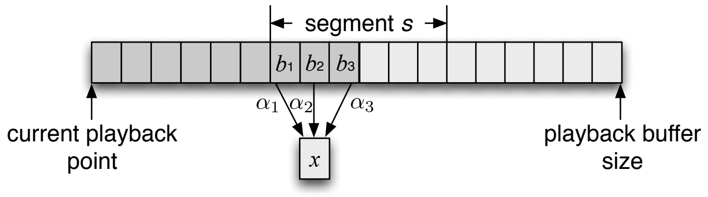

Figure 1.2 An example to illustrate the coding operation on peer p, where peer p has received 3 coded blocks within the segment s, and each segment consists of 6 blocks.

Suppose a peer p has received m ($m \leq k$) coded blocks in a segment s so far, denoted as $[\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_m]$. When peer p needs to serve segment s for its downstream peers, as shown in Fig. 1.2, it generates a coded block $\boldsymbol{x}$

$$\boldsymbol{x} = \alpha_1 \boldsymbol{b}_1 + \alpha_2 \boldsymbol{b}_2 + \cdots + \alpha_m \boldsymbol{b}_m,$$

with α_j $(1 \leq j \leq m)$ being chosen randomly from the finite field $\mathbb{F}_q$. As pointed out in Section 1.1.1, $\boldsymbol{x}$ is ultimately some random linear combination of the original blocks in segment s. Thus a downstream peer of p is able to decode segment s as long as it has received k linearly independent coded blocks.

In PULL, a missing block on a peer is requested and pulled from an appropriate neighbor. If this block has not been received within a certain time (due to peer departures, for example), it has to be requested and pulled again, under the risk of missing a deadline. With network coding in R^2, a missing segment on a peer is served by multiple neighbors simultaneously without any explicit coordination, as illustrated in Fig. 1.3. In this way, participating peers are able to perform *push* operations on a random mesh structure (*i.e.*, upstream peers decide which segments to serve without any requests from downstream peers), thereby fully utilizing available overlay connections. More importantly, even if a few neighbors leave the system suddenly, other neighbors serving the same segment are still at work, with little chance of missing a deadline.

Timely Feedback from Downstream Peers

Before pushing coded blocks, an upstream peer should obtain a precise knowledge of the missing segments on its downstream peers at any time. This requires

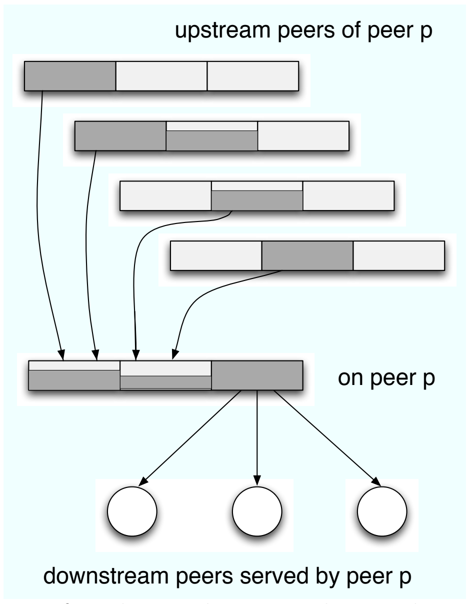

Figure 1.3 An illustration of random push on a random mesh structure. With network coding, multiple upstream peers are able to perform push operations on coded blocks within a segment without any explicit coordination.

participating peers in the system to exchange their buffer maps in a timely fashion. In PULL, buffer maps are exchanged periodically to avoid excessive overhead. While R^2 can afford "real time" exchanges of buffer maps — whenever a peer has played back or completely received a segment, it sends a new buffer map to all its neighbors.

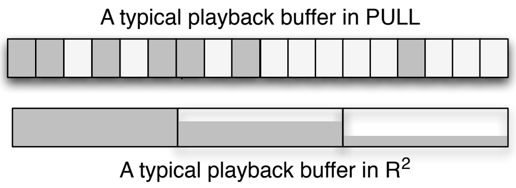
A typical playback buffer in PULL
A typical playback buffer in R^2

Figure 1.4 An illustrative comparison of playback buffers between R^2 and PULL.

Why is R^2 able to perform such "real time" exchanges of buffer maps? Since R^2 uses large segments instead of small data blocks, buffer maps that indicate segment availability information — as opposed to block availability information — can be an order of magnitude smaller. In addition, with larger segments, it takes much longer to playback or finish downloading a segment, leading to a slower update of buffer maps. As such, with reasonable overhead, a new buffer map can be sent to all neighbors as soon as its status has been updated.

Synchronized Playback and Initial Buffering Delays

The use of much larger segments also makes it easier to synchronize playback buffers on different participating peers, so that all peers are playing the same segment at approximately the same time. R^2 features *synchronized playback* as follows. When a peer joins or switches to a new streaming channel, it first retrieves buffer maps from its neighbors, along with the information of current segment being played back. To synchronize the playback buffer, the new peer skips a few segments and *only* retrieves segments that are D seconds after current playback point, where D corresponds to the *initial buffering delay*, which depends on the number of segments to be skipped and the current playback point (see Fig. 1.5). The peer then starts playback after precisely D seconds elapsed in real time, regardless of the status of the playback buffer.

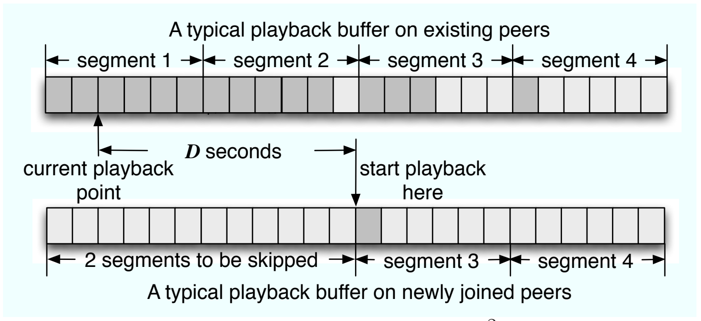

Figure 1.5 An illustration of initial buffering delays in R^2, which shows that the initial buffering delay on a newly joined peer is determined by the number of segments to be skipped and the current point of playback.

Under synchronized playback, peers are able to help each other more effectively, since their playback buffers overlap as much as possible. This desirable property is of particular importance during a flash crowd scenario, when a large number of peers join the streaming channel around the same time. As these newly arrived peers request almost same segments, they are able to help one another as soon as they receive a small number of coded blocks. This leads to a better utilization of their upload bandwidth, which in turn improves scalability of the streaming system.

Finally, we remark here R^2 represents a simple design philosophy, rather than a strict protocol design. The design space of R^2 is flexible to accommodate more elaborate protocols designed for different purposes. For example, a peer in R^2 has the freedom to decide which segments to be pushed to which neighbors. Referred to as a *push strategy*, this decision may be made based on timing requirement to ensure smooth playback of urgent segments, on fairness issues to encourage cooperation and reduce free-riding, or on geography consideration to reduce traffic across different ISPs.

1.2.2 Why is network coding helpful in multimedia streaming?

The strict timing requirement of multimedia streaming applications has marked a significant departure from applications in content distribution. Due to such a constraint, the advantages of network coding are less obvious. In fact, it has been shown in [WL07] 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 P2P streaming algorithms is indeed required. This motivated the design and implementation of R^2, as described before. In this section, we intuitively explain why the use of network coding in R^2 is able to provide good *overall* performance for streaming systems.

First, with network coding, R^2 is able to use much larger segments compared to typical data blocks in PULL. The use of larger segments leads to "real time" exchanges of buffer maps without additional overhead (or even with less overhead!). With up-to-date buffer maps in R^2, participating peers are able to serve each other better. While in PULL, buffer maps are exchanged in a periodic fashion. As shown in [ZZSY07, FLL09], the lack of timely exchanges of buffer maps may be a major factor that separates the actual performance of PULL from optimality.

Second, with network coding, peers in R^2 perform push operations rather than pull operations, thereby making a better use of their upload bandwidth resources. More importantly, even slow overlay connections may be utilized in R^2, which is generally impossible in PULL [ZZSY07]. In short, all these factors contribute to a better utilization of peer bandwidth resources, leading to a higher playback quality and reduced server bandwidth costs.

Third, with network coding, robustness to peer departures in R^2 can be significantly enhanced. Since multiple upstream peers are serving a 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 suddenly, the downstream peer has to find it out 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.

Last but not least, with network coding, R^2 scales well to accommodate a large flash crowd. Recall that the use of network coding enables synchronized playback in R^2. With playback buffers overlapping as much as possible, newly arrived peers during a flash crowd are able to help one another immediately after they have received a few coded blocks. This allows full utilization of upload bandwidth from new peers, which in turn greatly improves scalability of streaming systems.

1.2.3 Theoretical results on P2P multimedia streaming with network coding

In this section, we present a number of analytical results on the performance of P2P streaming systems with network coding, with a focus on the fundamental limits and achievable performances of R^2. For the sake of mathematical tractability, we make a few assumptions in the system model. The key notations involved is summarized in Table 1.1 for easy reference.

Table 1.1. 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 (the number of participating peers in the system).
k	Number of data blocks in each segment.
δ	Server strength ($= \frac{U_s}{NU_p}$).

First, in accordance with measurement studies of existing P2P systems (*e.g.*, [GCXTDZ03]), 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 [Kumar07], in which peers in the system are broadly classified into two classes, with each class having approximately the same upload capacity[2]. We use U_p to denote the *average* upload capacity of participating peers and U_s to denote the upload capacity of a dedicated streaming server (if multiple streaming servers exist, they can be regarded as a virtual "super-server").

We are now in a position to discuss several fundamental performance limits for P2P streaming systems with network coding. First of all, we observe that *the total bandwidth consumption should be no greater than the total bandwidth supply.* This leads to the following theorem, which has been proved in [Kumar07].

Theorem 1.2 ([Kumar07])**.** *The maximum streaming rate $R_{\max}$ is given by*

$$R_{\max} = U_p + \frac{U_s}{N},$$

where N is the number of participating peers in the system.

Second, let us consider the buffering delay for all participating peers in the system to buffer a segment. On the one hand, since a segment consists of k data blocks, at least kN block transmissions are needed for N peers to buffer a segment. On the other hand, the aggregate upload rate is upper bounded by

[2] 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 only two classes are assumed here, the analysis is easily extended to the case of multiple classes in the system.

$U_s + NU_p$, since a peer cannot serve a segment unless it has received at least one coded block in this segment. Thus we have the following limit on the buffering delay

Theorem 1.3. *Let D_s be the random variable denoting the buffering delay for a segment (the time required by all the peers in the system to receive the segment) under the system model described as above. Then for any given push strategy,*

$$E[D_s] \geq \frac{kN}{U_s + NU_p},$$

where N is the number of participating peers in the system, and k is the number of data blocks in each segment.

Note that at least one segment should be buffered to ensure smooth playback. By Theorem 1.3, it takes at least $E[D_s]$ seconds in expectation for participating peers to achieve so. In other words, Theorem 1.3 provides a lower bound for the shortest initial buffering delay during a flash crowd.

Given the above performance limits, we shall answer the following two questions:

- ▷ What is the sufficient condition for R^2 to achieve good overall performance?
- ▷ How far from optimality is the performance of R^2?

These questions are crucial for understanding the fundamental properties and limitations of R^2. We mainly focus on the flash crowd scenario here, since it poses unique challenges in the streaming system design, as observed in various measurement studies [Ross06, XLKZ07]. We refer readers to [FL08] for analysis under other peer dynamic patterns.

During a flash crowd, a large number of peers join the system within a short period of time, just after a new live event has been released. To model a flash crowd event, we assume time is *slotted* in the sense that it takes one time slot to playback a segment. We further assume that all participating peers join the system within one time slot. We emphasize here these assumptions are not necessary, and can easily be relaxed in the analysis. We now introduce the following definitions.

Definition 1.1. *The scale of a flash crowd, denoted by N, is defined as the maximum number of peers joining the system during a flash crowd event.*

Definition 1.2. *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.

Before we venture into theoretical analysis of R^2, we shall specify all design choices, since R^2 is not a strict protocol design, as discussed in Section 1.2.1. Indeed, we adopt a simple random push strategy in the analysis. More specifically, whenever a peer has a chance to serve, it chooses a partner uniformly at random from its neighbors, and uploads a coded block in the *most urgent* segment on that partner, that is, the segment closest to the playback point yet not completely received.

The following theorem establishes a sufficient condition on smooth playback at a streaming rate R during any flash crowd with scale N.

Theorem 1.4 ([FL08]). *Suppose the underlying overlay network is fully connected. Assume that the following condition holds:*

$$U_s + NU_p \geq (1 + \varepsilon)NR, \tag{1.3}$$

where ε *is given by*

$$\varepsilon = \gamma(q) + \frac{\ln(1+\delta) - \ln \delta}{k}, \tag{1.4}$$

and $\gamma(q)$ *is a system-wide parameter denoting the fraction of linearly dependent coded blocks induced by random linear network coding (which depends on the field size* q*). Then* R^2 *is able to achieve smooth playback at a streaming rate* R *under a simple random push strategy described as above, when the scale* N *of the flash crowd is sufficiently large.*

Combining Theorem 1.4 with Theorem 1.2 and 1.3, we are able to characterize the performance gap between R^2 and the optimal streaming scheme with regard to the sustainable streaming rate and initial buffering delay.

Corollary 1.1. *Suppose the underlying overlay network is fully connected. Then* R^2 *achieves a factor of* $1 + \varepsilon$ *of the maximum streaming rate* $R_{\max}$ *under a simple random push strategy, where* ε *is given in (1.4).*

Corollary 1.2. *Suppose the underlying overlay network is fully connected. Then* R^2 *achieves a factor of* $2(1 + \varepsilon)$ *of the minimum initial buffering delay under a simple random push strategy, where* ε *is given in (1.4).*

Corollary 1.1 and 1.2 demonstrate that R^2 is able to support a near-optimal streaming rate with short initial buffering delays during a flash crowd, even with a simple random push strategy. Here we present a concrete numerical example to better illustrate this point. As shown in [FL08], the parameter $\gamma(q)$ is typically in the order of 0.1% for large q ($q \geq 64$). Hence we set $\gamma(q) = 0.1\%$ in our example. We next set the server strength δ to 0.001, and the number of data blocks in each segment k to 100 in our example. Then the sustainable streaming rate R

satisfies $R \geq R_{\max}/1.07$, with initial buffering delays within a factor of 2.14 of the limit.

Notice that in above theorems, the overlay network is represented by a complete graph. However, in practical steaming systems, each participating peer only maintains a limited number of neighbors. Thus it is of great interest to investigate the impact of restricted neighborhood. Simulation results in [FL08] demonstrate that a small size of neighborhood (such as 50) is good enough to enjoy good overall performance in R^2.

1.2.4 Practical aspects of P2P multimedia streaming with network coding

The design philosophy of R^2 has been applied and implemented in UUSee — a large-scale operational streaming system operated by UUSee Inc. (one of the leading peer-assisted media content providers in China). With 200 Gigabytes worth of real-world traces collected and analyzed, it has been reported in [LWLZ10] that the theoretical benefit of using network coding can be achieved in practice: multiple upstream peers are allowed to collaboratively serve a downstream peer, leading to minimized buffering delays and serve bandwidth costs. In particular, the overall performance has been satisfactory for normal-quality videos in UUSee. For high-quality videos, the buffering delay could be larger due to the high bandwidth demand. Nevertheless, the UUSee measurements suggest that the delay is in general below a reasonable 20 seconds.

Any advantages may come with tradeoffs. We shall now look at the flip side of the coin — some practical issues in R^2 — to get a complete picture. Similar to content distribution applications, the computational cost of network coding is again a major concern for multimedia streaming applications. Even with modern processors, it may not be computationally feasible to support more than a few hundred data blocks in each segment. On the other hand, the use of large segments and small blocks is a key to the success of R^2, as explained before. Therefore, one shall maximize the number of data blocks in each segment and minimize the block size, subject to practical constraints.

For example, each segment in R^2 is divided into 128 data blocks of 2 KB each. With this design choice, good overall performance has been observed, but at the cost of sustained high CPU usage. In other words, we cannot afford a larger number of data blocks in each segment due to CPU constraint. Shall we choose a smaller block size in R^2? Note that the overhead in terms of the "header" is around 6% for this design choice, when the field size $q = 256$. Thus a smaller block size may lead to excessive overhead. To summarize, there is a trade-off between performance, computational cost and overhead. One shall find the right compromise for any particular streaming system.

Another practical concern in R^2 is the *braking redundancy*. Recall that a downstream peer sends a new buffer map to all its neighbors immediately after it has completely received a segment. This buffer map is also used as a signal to stop upstream peers from serving the segment. As it takes time for the braking signal to reach upstream peers, a downstream peer may receive additional redundant blocks after a segment is complete. How shall we minimize this braking redundancy? One engineering approach is to allow an "early braking" mechanism, which encourages a downstream peer to stop a subset of its upstream peers even *before* a segment is completely received. However, the design of such a "early braking" algorithm still remains a challenge.

Finally, a possible drawback of R^2 is that the time between the occurrence of a live event and its playback is the same across the board in all participating peers due to synchronized playback. Though harmful to some applications, this may be an advantage to those involving live interaction (such as live voting with SMS): all participating peers will view the same content at the same time, such that interactive behavior starts to occur at the same time as well.

1.3 Conclusion

The main objective of this chapter is to explore the potential benefits that network coding may offer in P2P networks. In particular, we have addressed two major applications: content distribution and multimedia streaming. To achieve this goal, we first describe how network coding can be successfully used in each application. We then provide a number of key insights on the advantages offered by network coding.

- ▷ In P2P content distribution, we have shown that the use of network coding solves the block scheduling problem in a surprisingly simple and effective way, leading to a shorter file downloading time and better robustness to peer departures.
- ▷ In P2P multimedia streaming, we have shown that a complete redesign of streaming protocols is required in order to take full advantages of network coding. In particular, we have presented R^2 — a new streaming system design with network coding — and explained why the use of network coding in R^2 is able to fully utilize available bandwidth resources, thereby improving the overall system performance.

To deepen understanding of these claimed advantages, we further provide a number of selected theoretical results. Finally, we present several practical issues that deserve special attention in real-world system design. We believe such an exploration could shed light on future applications of network coding in P2P networks.

References

[C03] B. Cohen. Incentives Build Robustness in BitTorrent. *1st Workshop on Economics of Peer-to-Peer Systems*, (Berkeley, CA), June 5-6, 2003.

[PGES05] J. Pouwelse, P. Garbacki, D. Epema and H. Sips. The Bittorrent P2P File-sharing System: Measurements and Analysis. *Proc. of 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, (Ithaca, New York), Feb. 24-25, 2005.

[GR05] C. Gkantsidis and P. Rodriguez. Network Coding for Large Scale Content Distribution. *Proc. of IEEE INFOCOM 2005*, (Miami, FL), March 13-17, 2005.

[GMR06] C. Gkantsidis, J. Miller and P. Rodriguez. Anatomy of a P2P Content Distribution System with Network Coding. *Proc. of 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, (Santa Barbara, CA), Feb. 27-28, 2006.

[Y07] R. W. Yeung. Avalanche: A Network Coding Analysis. *Communications in Information and Systems*, vol. 7, pp. 353-358, 2007.

[W06] Yunnan Wu. A Trellis Connectivity Analysis of Random Linear Network Coding with Buffering. *Proc. of International Symposium on Information Theory (ISIT)*, (Nice, France), June 24-29, 2007.

[HKMESK06] 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*, vol. 52, pp. 4413-4430, October 2006.

[DMC06] S. Deb, M. Medard and C. Choute. Algebraic Gossip: A Network Coding Approach to Optimal Multiple Rumor Mongering. *IEEE Transactions on Information Theory*, vol. 52, pp. 2486-2507, June 2006.

[CWJ03] P. Chou, Y. Wu and K. Jain. Practical Network Coding. *Proc. of Allerton Conference on Communication, Control, and Computing*, (Monticello, Illinois), Oct. 2003.

[MHL06] P. Maymounkov, N. J. A. Harvey and D. S. Lun. Methods for Efficient Network Coding. *Proc. of Allerton Conference on Communication, Control, and Computing*, (Monticello, Illinois), Oct. 1-3, 2006.

[NL07] D. Niu and B. Li. On the Resilience-Complexity Tradeoff of Network Coding in Dynamic P2P Networks. *Proc. of 15th IEEE International Workshop on Quality of Service (IWQoS)*, (Evanston, IL), June 2007.

[WL06] M. Wang and B. Li. How Practical is Network Coding?. *Proc. of 14th IEEE International Workshop on Quality of Service (IWQoS)*, (New Haven, CT), June 19-21, 2006.

[VYF06] V. Venkataraman, K. Yoshida and P. Francis. Chunkyspread: Heterogeneous Unstructured Tree-based Peer-to-Peer Multicast. *Proc. of 14th IEEE International Conference on Network Protocols (ICNP)*, (Santa Barbara, CA), Nov. 12-15, 2006.

[XLKZ07] S. Xie, B. Li, G.-Y. Keung, and X. Zhang. Coolstreaming: Design, Theory, and Practice. *IEEE Transactions on Multimedia*, vol. 9, pp. 1661-1671, December 2007.

[HFCLH08] Y. Huang, Z. J. Fu, D. M. Chiu, C.S. Lui and C. Huang. Challenges, Design and Analysis of a Large-scale P2P VoD System. *Proc. of ACM Sigcomm*, (Seattle, WA), Aug. 17-22, 2008.

[WL07] M. Wang and B. Li. Lava: A Reality Check of Network Coding in Peer-to-Peer Live Streaming. *Proc. of IEEE INFOCOM*, (Anchorage, Alaska), May 6-12, 2007.

[WLi07] 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*, vol. 25, pp. 1655-1666, Dec. 2007.

[ZZSY07] 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*, vol. 25, pp. 1678-1694, Dec. 2007.

[FLL09] C. Feng, B. Li and B. Li. Understanding the Performance Gap between Pull-based Mesh Streaming Protocols and Fundamental Limits. *Proc. of IEEE INFOCOM*, (Rio de Janeiro, Brazil), April 19-25, 2009.

[GCXTDZ03] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding and X. Zhang. Measurements, Analysis, and Modeling of BitTorrent-like Systems. *Proc. of Internet Measurement Conference (IMC)*, (Berkeley, CA), Oct. 19-21, 2005.

[Kumar07] R. Kumar, Y. Liu and K. W. Ross. Stochastic Fluid Theory for P2P Streaming Systems. *Proc. of IEEE INFOCOM*, (Anchorage, Alaska), May 6-12, 2007.

[LWLZ10] Zimu Liu, Chuan Wu, Baochun Li and Shuqiao Zhao. UUSee: Large-Scale Operational On-Demand Streaming with Random Network Coding. *Proc. of IEEE INFOCOM*, (San Diego, California), March 15-19, 2010.

[Ross06] X. Hei, C. Liang, J. Liang, Y. Liu and K. W. Ross. A Measurement Study of a Large-Scale P2P IPTV System. *IEEE Transactions on Multimedia*, vol. 9, pp. 1672-1687, Dec. 2007.

[FL08] C. Feng and B. Li. On Large-Scale Peer-to-Peer Streaming Systems with Network Coding. *Proc. of ACM Multimedia*, (Vancouver, BC), Oct. 27 - Nov. 1, 2008.

[KK08] R. Koetter and F. R. Kschischang. Coding for errors and erasures in random network coding. *IEEE Trans. on Information Theory*, vol. 54, pp. 3579-3591, Aug. 2008.

[WHLC09] Yunnan Wu, Y. C. Hu, J. Li and P. A. Chou. The Delay Region for P2P File Transfer. *Proc. of International Symposium on Information Theory (ISIT)*, (Coex, Seoul, Korea), June 28-July 3, 2009.

[CHEML10] C. S. Chang, T. Ho, M. Effros, M. Medard and B. Leong. Issues in Peer-to-Peer Networking: a Coding Optimization Approach. *Proc. of the 2010 IEEE International Symposium on Network Coding (NetCod)*, (Toronto, Canada), June, 2010.

[DGWWR10] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright and K. Ramchandran. Network Coding for Distributed Storage Systems. *IEEE Trans. on Information Theory*, vol. 56, pp. 4539-4551, Sep. 2010.

[ADMK05] S. Acedanski, S. Deb, M. Mdard and R. Koetter. How Good is Random Linear Coding Based Distributed Networked Storage? *First Workshop on Network Coding, Theory, and Applications (NetCod)*, (Riva del Garda, Italy), April 2005.