INVITED PAPER

Random Network Coding in Peer-to-Peer Networks: From Theory to Practice

The authors of this paper believe that applications such as file sharing and video streaming that involve peer-to-peer networks may be the most promising area for using network coding.

By Baochun Li and Di Niu

ABSTRACT | With random network coding, network nodes between the source and receivers are able to not only relay and replicate data packets, but also code them using randomly generated coding coefficients. From a theoretical perspective, it has been recognized that network coding maximizes the network flow rates in multicast sessions in directed acyclic network graphs. To date, random network coding has seen practical and real-world applications in peer-to-peer (P2P) networks, in which overlay network topologies are formed among participating end hosts, called "peers." Due to uncertainties and dynamics involved with peer arrivals and departures, these network topologies are usually randomly generated in practice, and are referred to as "random mesh" topologies. Unlike structured topologies such as trees, random mesh topologies are practical to be implemented, and are resilient to the level of volatility typically experienced in peer-to-peer networks. It has been shown, from both theoretical and practical perspectives, that random network coding leads to performance benefits in these peer-to-peer networks with random mesh topologies. This paper presents a survey of existing results with respect to practical applications of random network coding in peer-to-peer networks. We focus on bulk content distribution and media streaming systems, as well as the computational overhead introduced by random network coding in modern off-the-shelf servers and mobile devices. Throughout the paper, we also show theoretical insights on why random network coding may become beneficial in practice.

KEYWORDS | Content distribution; media streaming; peer-to-peer (P2P) networks; random gossip; random network coding; randomized algorithms

I. INTRODUCTION

Network coding, since its inception in information theory [1], has attracted a substantial amount of research attention in the networking community. It has been widely known that, with network coding, the cut-set bound of information flow rates in multicast communication sessions can be achieved. The essence of network coding is a paradigm shift to allow coding at network nodes between the source and receivers in a communication session.

With the ability to code at intermediate network nodes in a communication session, we may forward, replicate, and code incoming packets. This capability is in sharp contrast to traditional commodity flows, where only forwarding is allowed. The seminal works by Ahlswede *et al.* [1] and Koetter *et al.* [2] have shown that, in a directed acyclic network with network coding, a multicast rate is feasible if and only if it is feasible for a unicast from the sender to each receiver. Li *et al.* [3] have further shown that linear coding suffices to achieve the maximum rate.

One may naturally wonder whether theoretical benefits of network coding may become useful in practical real-world networking systems. To practically implement network coding, one needs to address the challenges of computing *coding coefficients* to be used by each of the intermediate nodes in a session, so that coded packets at the receivers are guaranteed to be decoded—a process

Manuscript received November 17, 2009; revised October 19, 2010; accepted October 21, 2010. Date of publication January 6, 2011; date of current version February 18, 2011.
The authors are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M55 3G4 Canada (e-mail: bli@eecg.toronto.edu; dniu@eecg.toronto.edu).
Digital Object Identifier: 10.1109/JPROC.2010.2091930

called *code assignment*. Although deterministic code assignment algorithms have been proposed and shown to be polynomial time algorithms (e.g., [4]), they require expensive exchanges of control packets. Ho *et al.* [5] have proposed the concept of *random network coding*, where a network node transmits on each of its outgoing links a linear combination of incoming packets over a finite field, with randomly chosen coding coefficients.

With random network coding, Chou *et al.* [6] have first conceived that random network coding can be applied in real-world networks, by dividing a stream of information into *generations*, and by performing random linear coding within each generation. Coding coefficients may be carried by packets themselves before transmission. It has been concluded that random network coding is robust to packet loss, delay, as well as variations in the network topology and capacity, and that sessions with random network coding can achieve close to the theoretically optimal performance.

Intuitively, an excellent scenario where network coding may be applied in practice is peer-to-peer (P2P) networks. In P2P networks, end hosts (from servers to smartphones), called "peers," organize themselves in *overlay* topologies, in which packet transmission on each of the overlay links is free of errors, thanks to transport protocols used in the Internet, such as transmission control protocol (TCP). Peers are also computing devices that are potentially capable of coding packets by implementing network coding in software, without the need of revising existing switches and routers in the Internet.

P2P bulk content distribution systems, such as BitTorrent, allow peers to collaborate with one another so that large files can be distributed from one peer to a large number of subscribing receivers, without the aid of dedicated servers. P2P bulk content distribution systems adopt a simple design philosophy: a file is divided into *blocks*, and peers connect with one another in a random mesh topology, exchanging these blocks with random "gossiping." Just as its name suggests, in gossiping each peer transmits a subset of the blocks it has obtained to a subset of its neighbors that are selected using randomized algorithms. Such random gossiping on random mesh topologies is simple to implement, resilient to the level of volatility caused by peer arrivals and departures, and has been shown to achieve good performance from a theoretical perspective. For these reasons, real-world P2P media streaming systems, currently used by millions of users, are also designed by following such a design philosophy.

It is natural to see a potential link between random network coding and random gossiping, in random mesh topologies. Due to their common randomized nature, one wonders if they can be used in an integrated fashion in P2P networks. In essence, random gossiping studies the spread of a single block or multiple blocks across a group of participating peers, and random network coding, by mixing the blocks each peer transmits, is shown to maximize the information diffusion efficiency given limited bandwidth in network topologies. In this paper, we start by introducing random gossiping algorithms in the context of P2P networks in general, and bulk content distribution systems in particular. We then explore the potential benefits of using random network coding in both P2P content distribution and P2P live media streaming systems. In order to bring random network coding to practical use, we show the performance of heavily optimized implementations of network coding on a variety of modern off-the-shelf computing hardware, including multicore processors, graphics processing units (GPUs), and mobile phones. Finally, we show the design of the first real-world deployment of an on-demand streaming system that uses random network coding, and is now operational with millions of Internet users.

II. RANDOM NETWORK CODING IN P2P CONTENT DISTRIBUTION

P2P network topologies are formed by end hosts who connect with one another using overlay links, and these topologies are formed for a particular objective. One of the objectives is to distribute some bulk content, such as a large file, from one peer to other subscribing peers in a topology. More formally, consider a network with n peers who wish to receive a copy of a large file to be disseminated. The file is divided into k blocks. For simplicity, let us also assume a synchronized model in which time is measured in "rounds." Each peer uploads to "neighbors" that are selected randomly. How many rounds are needed for all peers to receive a copy of the file?

Conceptually, this objective is similar to the classical problem of random gossiping, which considers the problem of n people spreading a rumor initially held by only one person. How many rounds are needed for everyone to receive the rumor? Pittel [7] has shown that, if each person who has already received the rumor communicates it to a person chosen at random and independently of all other past and present choices, it takes $\log_2 n + \log n + O(1)$ rounds for the rumor to reach all n persons. Such a random target selection protocol is also called the "random phone call" model. The difference between P2P content distribution and the problem of random gossiping is that multiple blocks need to be distributed, rather than a single rumor.

Sanghavi *et al.* [8] are among the first to study the time needed to spread multiple blocks. They extend the "random phone call" model to incorporate multiple blocks, where in each round, each peer communicates with another target peer chosen uniformly at random from the entire network, and each peer can upload at most one of the blocks it possesses. Within such a model, even if centralized block scheduling is allowed, at least $k + \log_2 n$ rounds are needed to disseminate all k blocks from a single source to all n peers [9], [10]. The intuition is that it takes at least k rounds for the source to issue the last block, and a further $\log_2 n$ rounds for that block to reach all n peers.

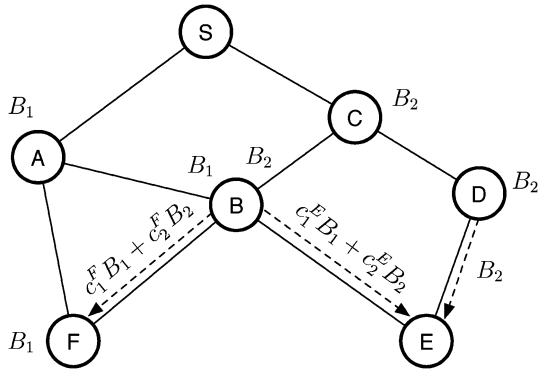

Fig. 1. *An example of distributing two blocks B_1 and B_2 with network coding. S is the source peer.*

Sanghavi *et al.* [8] show that with a decentralized block selection protocol, one can finish distributing k blocks from a single source to n peers in $9(k + \log n)$ time, with high probability for a large number of peers.

A. Using Random Network Coding

We now turn to a natural question that arises when we consider network coding. Will the use of random network coding reduce the number of rounds needed to distribute k blocks with random gossiping?

Gkantsidis *et al.* [11] was the first to consider random network coding as a substitute for P2P dissemination based on exchanges of individual blocks (e.g., BitTorrent) that can be executed in a decentralized fashion. It was argued that, if peers can linearly combine all the blocks it has already received so far using randomly generated coding coefficients, and then transmit such coded blocks to other peers, the amount of time required to distribute a large file to all the peers in the network may be reduced.

A simple example of distributing two blocks is given in Fig. 1. Suppose that peer B has received blocks B_1 and B_2, and peer D has received block B_2. Although both peers B and D can serve peer E at this point, they may end up transmitting the same block B_2 to peer E, since there is no connection between peers B and D, and their transmission can hardly be coordinated. In this case, the upload bandwidth of peer B is wasted. With the use of network coding, however, peer B can transmit to peer E a coded block $c_1^E B_1 + c_2^E B_2$, with coefficients c_1^E and c_2^E randomly chosen. Peer E can solve for B_1 using the received blocks B_2 and $c_1^E B_1 + c_2^E B_2$.

Similarly, without coding, peer B in Fig. 1 may transmit to peer F block B_1 that is already possessed by peer F, unless accurate and frequent buffer comparison is performed. With network coding, peer B can instead transmit to peer F another randomly encoded block $c_1^F B_1 + c_2^F B_2$, which is always useful to peer F if c_1^F and c_2^F are appropriately chosen. Intuitively, the use of random network coding has increased the diversity of blocks being transmitted.

In practice, however, the computational complexity of network coding escalates with an increasing number of blocks. To manage such complexity, it has been proposed [6] that blocks in a file be divided into multiple *generations*, and network coding is only performed within the same generation. To be more specific, the original file with F bytes is divided into G generations, each of which is further divided into m blocks, referred to as the *generation size*. There are a total of $M = G \cdot m$ original blocks, each with a size of $k = F/M$ bytes.

When the file is to be distributed, random network coding is applied across the blocks within a generation, say generation i, containing m original blocks $\mathbf{B}^{(\mathbf{i})} = [B_1^i, B_2^i, \ldots, B_m^i]$. On the source, a coded block b from this generation is a linear combination of these original blocks in the Galois field $\mathrm{GF}(2^q)$. Network coding is, of course, not limited to the source: if a peer (including the source) possesses l ($l \leq m$) coded blocks $[b_1^i, b_2^i, \ldots, b_l^i]$ of generation i, when the need arises to serve a new coded block to a neighbor p, it independently and randomly chooses a set of coding coefficients $[c_1^p, c_2^p, \ldots, c_l^p]$ in the Galois field $\mathrm{GF}(2^q)$, and encodes all the blocks of generation i it possesses to produce one coded block x of k bytes: $x = \sum_{j=1}^{l} c_j^p \cdot b_j^i$.

When random network coding is applied, for the benefit of successful decoding, a coded block x is self-contained, in that coding coefficients used to encode *original blocks* to x are embedded in its header. As soon as a peer has received a total of m coded blocks from generation i that are linearly independent, $\mathbf{x} = [x_1^i, x_2^i, \ldots, x_m^i]$, it will be able to recover all original blocks in this generation with Gaussian elimination, taking advantage of the coding coefficients embedded in each of the m coded blocks received.

B. Random Gossiping With Network Coding

With the use of random network coding, it is intuitive to see that coded blocks are equally useful at a receiver, as the receiver only needs to *accumulate* a sufficient number of coded blocks, which is easier than *collecting* a number of distinct original blocks from its neighbors. When original blocks are collected by a receiver, it may occur that a *rare* block is harder to run into using random gossiping, which leads to a longer time to finish collecting all the blocks in a file. In contrast, with random network coding, the receiver only needs to "hold a bucket" until it is "full," which is only constrained by its available download bandwidth. Random network coding simplifies the design of block selection protocols, as it enables the "blind" transmission of coded blocks, without the need for any reconciliation of blocks between a pair of peers.

From a more theoretical perspective, Deb *et al.* [12] are the first to analyze the performance of using random network coding with random gossiping. The "random phone call" model has also been considered, where each peer chooses a random target from the entire network to upload to, obeying the constraint that only one block can be

transmitted in a round. It is assumed that each peer has one of the k blocks initially.

Deb *et al.* [12] have shown that, if each peer linearly combines all the blocks it has already received using random coefficients and transmits this coded block to its target, the time for all n peers to receive all k blocks is $ck + O(\sqrt{k}\log k \log n)$ rounds, where c is a constant around 3–6. This essentially shows that by "blindly" transmitting coded blocks and selecting communication partners randomly without any form of reconciliation between peers, the time to disseminate all k blocks to all the peers is linear in terms of k.

As each peer chooses random targets from the entire network to communicate with, it is implicitly assumed in [12] that the P2P overlay topology—defined by the neighboring relationships—is a complete graph. Performance bounds on the time to disseminate all the blocks using random gossiping with random network coding in arbitrary networks, where each peer can only communicate with its neighbors defined by the graph, have been given by Mosk-Aoyama *et al.* [13], Vasudevan *et al.* [14], and Borokhovich *et al.* [15].

C. Optimality From a Trellis Graph Perspective

Random network coding is originally introduced to P2P content distribution systems in order to reduce the time required to successfully distribute a file to all participating peers, yet with a simplified design of the protocol involved. As a matter of fact, by unfolding the evolution of network states over a number of rounds to a discrete-time *trellis graph*, it has been proved by Yeung [16] that, with the use of network coding, optimal broadcast times are achieved, regardless of the network topology and transmission schedule.

In the spirit of explaining why network coding leads to the shortest broadcast times possible, we explain the gist of the proof by showing the trellis graph in the context of an example involving four peers. As shown in Fig. 2, peer 1 is the source of the content distribution session in our simple example. In the trellis graph, the vertex i_t is used to represent the state of peer i at time t. If l ($l \leq m$) blocks within the same generation are transmitted from peer i to j starting at time t and ending at time t' ($t' > t$), there is a directed edge in the trellis graph with capacity l from the vertex i_t to $j_{t'}$. Naturally, all blocks received by peer i at time t, represented by the vertex i_t, are still available at peer i in its buffer at time $t' > t$. This fact is represented by a directed edge with infinite capacity from the vertex i_t to i_{t+1}, shown by dashed lines in the trellis graph. Let $\text{maxflow}(i_t)$ denote the maximum flow from the vertex 1_{t_1} to vertex i_t in the trellis graph. We can see that $\text{maxflow}(2_{t_2}) = 2 + 2 = 4$, shown by the thicker edges.

We can now apply the theorem that network coding maximizes the multicast flow rate in acyclic graphs [1] to such a trellis graph. Given a set of participating peers U_t at time t, if $\text{maxflow}(U_t) \geq m$, then all the peers in U are

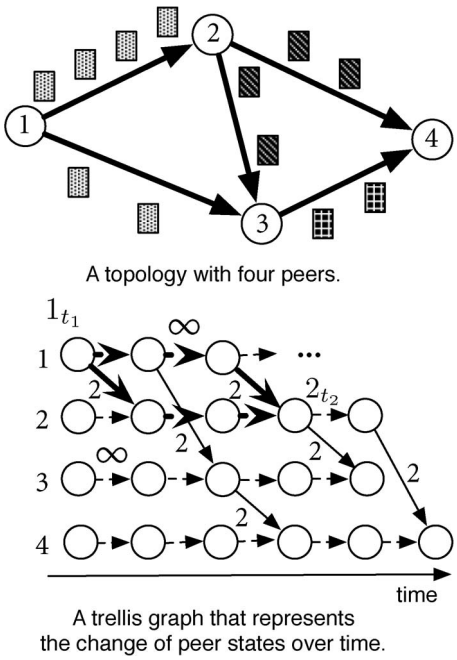

Fig. 2. *A discrete-time trellis graph representing a simple example with four peers.*

able to receive m blocks in its buffer at time t with high probability, when random network coding is applied and the size of the finite field is sufficiently large. In other words, the minimum length of time $t_{\text{opt}}(U)$ required for a set of peers U to receive all m blocks in the same generation is the minimum t that satisfies $\text{maxflow}(U_t) \geq m$, i.e.,

$$t_{\text{opt}}(U) = \inf\{t : \text{maxflow}(U_t) \geq m\}.$$

In our example, the minimum time $t_{\text{opt}}(\{2, 3, 4\})$ that peers 2, 3, and 4 will take to receive all four blocks is five rounds. Without taking into account the small probability of linear dependence, this is the best possible performance one can expect from such a system represented by the trellis graph, because if at a given time t, $\text{maxflow}(U_t) < m$, it is *impossible* for peers in U to recover all m blocks in the generation regardless of whether network coding is used, even if the peers are allowed to exchange information among themselves.

D. Practical Limitations of Network Coding

There are, however, limitations to the benefits that random network coding may bring to the design of P2P content distribution protocols. Since random network coding can only be performed on blocks within the same generation, reconciliation between a pair of neighboring peers may still be necessary across the boundary of generations. In other words, rather than collecting fine-grained

blocks, now a receiver only needs to collect all the distinct coarse-grained generations that constitute the file to be distributed. It mitigates the problem of locating rare portions of a file that may be less available in the entire P2P network. In practice, such a need for reconciliation, even at the coarser granularity of generations, may negatively affect the advantage of random network coding.

From an analytical perspective, Niu *et al.* [17] have analyzed how the generation size m used for network coding can affect the block diversity in a P2P network with peer dynamics. They have quantitatively identified an inversely proportional correlation between the block diversity and the size of the generation m with the use of random network coding: the larger the generation size, the more diverse blocks are in the system, and the faster downloads can be completed.

Theoretical analysis using the discrete-time trellis graph model shows that protocols using random network coding are able to achieve the best possible performance, but the performance gap between the use of network coding and a well-designed block selection protocol is not clear. Such a performance gap may be quite small, as practical block selection protocols without coding, such as BitTorrent [18], are usually designed to download the rarest blocks first (with the aid of frequent exchanges of buffer states among peers), in the hope of mitigating some of the adverse effects of locating rare blocks as the file download approaches completion. In contrast, random network coding uses additional computational power on *every* peer, which may not be well justified if the performance advantage of network coding is small in large-scale P2P networks.

Empirical studies [19] have attempted to show that the use of random network coding is practical in P2P content distribution systems, leading to smooth and fast downloading sessions, and without much additional computational overhead. However, it was not clear how distinct generations should be selected and reconciled between peers, and there were no comparisons with any of the operational P2P content distribution systems that do not use network coding, especially at a large scale.

To date, though random network coding appears promising in theoretical settings, its benefits in real-world content distribution systems have not yet been quantitatively evaluated with respect to a number of important performance metrics, such as download completion times and resilience to peer dynamics. There have not been practical bulk content distribution systems—in operation with real-world users—that are designed and implemented with the use of random network coding.

III. RANDOM NETWORK CODING IN P2P STREAMING

Unlike bulk content distribution, live and on-demand streaming systems require the timely delivery and playback of time-sensitive data streams, typically video streams. The motivation of using the P2P paradigm is to conserve the bandwidth consumed at dedicated streaming servers, since peers are able to contribute their uplink bandwidth to the system by uploading media streams to one another.

It turns out that, despite the emphasis on the timing of block delivery, the design philosophy of random gossiping in practical P2P content distribution systems is equally useful in P2P streaming systems. In P2P live streaming systems, where media channels are broadcast live to participating peers, the only fundamental difference is that a dynamic *sliding window* of blocks over time needs to be distributed in a streaming fashion, rather than a fixed number of blocks in a static file. The design philosophy is first applied in P2P live streaming systems by Zhang *et al.* [20], who first proposed the live *CoolStreaming* system using random gossiping, and evaluated its performance [21]. Its design organizes peers into a random mesh topology, requiring them to "pull" data blocks from each other, after exchanging block availability information.

We briefly explain common design elements of P2P live streaming systems that use random gossiping in random mesh topologies. As a peer joins the streaming session, it connects to a list of randomly selected peers as its neighbors, and starts to exchange data blocks in a sliding window with them. As blocks accumulate in the sliding window (residing in the main memory), media playback begins shortly afterwards, as a certain criterion about the number of blocks accumulated is met, or after a predetermined length of time.

During playback, any block arriving later than the time it is due for playback will be skipped, causing a pause or quality degradation in the playback process. In live streaming systems, blocks will be discarded after playback, and the sliding window advances itself. The sliding window has a predetermined size, usually spanning several minutes during playback. This implies that if the playback buffer of a peer is about to exceed the size of the sliding window, no more blocks will be "pulled" from other peers until some of the existing blocks are played back.

Since all participating peers are roughly synchronized with respect to their points of playback, they are able to periodically exchange the states of their respective buffers in the sliding window, usually called "buffer maps." Based on the knowledge of block availability in each other's sliding windows, a peer sends requests to its neighbors in order to "pull" blocks it has yet to receive.

A. Design Objectives

Since the design philosophy of random gossiping in random mesh topologies has recently been adopted by a number of operational P2P streaming systems, such as PPLive [22], it has performed quite well. Tens of thousands of video channels have been successfully served to millions of users in systems of such a kind. It is simple to

implement such a design philosophy, which is robust to peer arrivals and departures. The important question, however, is whether *random network coding* is a suitable choice to be integrated in such a design philosophy.

Before we explore the feasibility and potential benefits of using network coding, we would first like to present a few performance metrics that are important in P2P streaming systems. First, since the *initial buffering delay* must be experienced by a user when it joins a streaming session, a shorter delay improves the user experience. Second, since peer upload capacities may not be sufficient to sustain the entire streaming session for all participating peers, dedicated streaming servers provide additional "supply" of bandwidth. As the operational costs of these dedicated servers depend on the bandwidth consumed, it is critical to minimize *server bandwidth costs*. Finally, with an increasing number of peers in a session and a fixed amount of bandwidth available at dedicated streaming servers, the saturation point will eventually be reached, where the aggregate bandwidth supply barely exceeds the demand. We wish to maintain *smooth playback* as much as possible in such challenging situations.

B. Random Network Coding in P2P Live Streaming

We now turn our attention to the potential benefits of using random network coding in P2P live streaming systems. Live streaming systems using the random gossiping philosophy are practical to be implemented, but they do have an "Achilles' heel": *communication overhead*. Intuitively, since the sliding window at a peer advances itself over time, buffer availability maps need to be exchanged periodically, which may lead to a substantial amount of overhead.

As an example, let us consider a streaming system that uses 600 blocks in its sliding window, each block lasting one second of playback at a typical streaming bit rate of 45 kB/s. It requires 75 B to represent a buffer availability map. With a conservative estimate of 60 neighbors, if a peer sends these buffer maps to all the neighbors every second, it amounts to an overhead of 4.5 kB/s at each peer, which is 10% of the overhead as a ratio of the streaming bit rate. A higher communication overhead leads to less efficient utilization of peer upload bandwidth, and indirectly results in higher bandwidth costs at dedicated streaming servers. To mitigate such an overhead, most practical live streaming systems choose to exchange buffer maps less frequently. An analytical study [23], however, has attributed the performance gap between practical systems and their theoretically optimal performance to the lack of timely exchanges of buffer availability maps.

Would the use of random network coding be able to mitigate the problem of communication overhead? Wang *et al.* [24] have raised such a question, and presented a new set of design principles, referred to as R^2, that takes advantage of random network coding to substantially improve the performance of live streaming systems, addressing concerns of playback quality, server bandwidth costs, and initial buffering delays.

R^2 stands for random push with random network coding. Similar to the use of network coding in content distribution systems, it divides the content of the media stream in a sliding window into generations, each of which is further divided into m blocks. With the introduction of generations in R^2, we can afford to design parameter settings so that a block is much smaller than its counterpart in traditional live streaming systems based on random gossiping. This is due to the fact that buffer availability maps only need to stay at the granularity of a generation (one bit to represent each generation), rather than a block. With the same amount of communication overhead to exchange buffer availability maps, the size of a single block can be much smaller in R^2.

When a peer serves a generation s to its downstream target peer p, it linearly encodes all the blocks it has received so far from s using random coefficients in $\mathrm{GF}(2^8)$, and then transmits the coded block to p. Since each peer buffers coded blocks it has received so far, it is able to linearly combine them using random coefficients, much like how random network coding is used in P2P content distribution systems. Only blocks from the same generation are allowed to be coded, in order to reduce the computational complexity of network coding.

In response to unique requirements imposed by live streaming systems, in particular the timeliness of playback, R^2 proposes a set of design principles that facilitate the use of generation-based network coding. It advocates the use of *random push* instead of "pull" to transmit data: each peer randomly selects a small number of downstream peers based on certain criteria. When serving a chosen downstream peer, it then randomly selects a generation to code within, among those that the downstream peer has not yet completely received. If generations very closely ahead of the point of playback have not been completely received, they are given a higher priority as they are more urgent.

There are a number of clear advantages brought forth by the design principles in R^2. First, it greatly simplifies protocol design. Since coded blocks within a generation are useful with high probability during random gossiping, when serving its downstream peer, a peer only needs to blindly push coded blocks in a generation, until the downstream peer has obtained a sufficient number of coded blocks to decode and recover this generation. This eliminates the need of sending explicit requests to "pull" missing blocks, as well as the communication overhead associated with these requests.

Second, R^2 induces much less overhead involved in buffer map exchanges, due to a smaller number of generations in the sliding window. Consider the previous example of a live streaming system, in which a playback buffer has 600 blocks representing 600 s of playback, with each block corresponding to one second of playback at a 45 kB/s streaming bit rate. With R^2, one can divide the

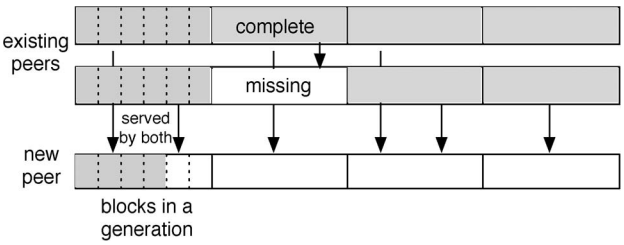

Fig. 3. *P2P live streaming with the use of random network coding: multiple existing peers are able to collaborate and serve coded blocks within the same generation to a new peer joining the session, minimizing its initial buffer delay.*

sliding window into 60 generations, each with 450 blocks of 1 kB in size. This requires only 8 B to represent the sliding window in each buffer availability map. A generation is played back and removed from the sliding window every 10 s, and a new generation is completely received every 10 s in a steady state. This implies that a peer only needs to send at most two buffer maps to each neighboring peer every 10 s, on average. With 60 neighbors, it amounts to a negligible overhead of only 48 B/s.

Finally, R^2 makes it possible for an incoming peer to start its playback with the shortest initial buffering delay. Illustrated in Fig. 3, as a new peer joins the session, it establishes a playback point that it intends to start playing in the future, and its empty sliding window starts from this playback point. *Multiple* existing peers are able to collaborate and push fresh coded blocks in the first one or two generations after the playback point, so that the rate of accumulating blocks in these generations is only limited by the new peer's available downlink bandwidth. Once the playback point is reached, it is likely that the new peer will have received at least the first generation, if its download bandwidth is higher than the playback bit rate of the stream.

The advantage of such "perfect collaboration" among multiple upstream peers when serving coded blocks is not limited to shorter initial buffering delays. It also allows the streaming protocol to be more resilient to peer dynamics, since the downloading process of a particular generation is not adversely affected by the departure of any of its serving peers. Without the use of random network coding, a coordination protocol across multiple serving peers is needed to avoid sending duplicates to the new peer. Such a protocol is not scalable to the number of serving peers. It also incurs communication overhead with the use of state update messages, and leads to longer latencies waiting for these messages to arrive. This is another example where network coding simplifies protocol design, and as a result it not only reduces the amount of overhead due to the transmission of protocol-related packets, but also becomes more resilient to packet losses, excessive delays, and peer departures.

Using both theoretical analysis and simulations, Feng *et al.* [25] have shown that the design principles in R^2 have led to much better performance than live streaming protocols without network coding, especially in extreme scenarios such as "flash crowd" sessions where a large number of peers arrive in a short period of time. It has been analytically shown that the maximum streaming bit rate that R^2 is able to support is within a factor of $1+\epsilon$ of the optimal achievable rate, where ϵ is a small constant that depends upon the number of blocks in a generation, as well as the ratio of server bandwidth over the total uplink bandwidth in the system. This positively reflects the ability of random network coding to support a smooth playback experience when the "supply" of bandwidth has been saturated by the "demand," which is one of the most challenging scenarios when P2P live streaming systems scale up to millions of users.

IV. IMPLEMENTING RANDOM NETWORK CODING IN MODERN HARDWARE

Conceptually, random network coding may be beneficial in both P2P content distribution and live streaming systems. In practice, however, it pays the price of higher computational complexity. Will modern off-the-shelf computer hardware without customization—including dedicated streaming servers, notebook computers, and mobile phones—be able to perform a software implementation of random network coding?

A. Network Coding With Multicore Processors

Shojania *et al.* [26] have presented the first attempt towards a high-performance implementation of random network coding. Progressive decoding with Gauss–Jordan elimination has been implemented on both ×86 and PowerPC processor families, such that coded blocks can be decoded progressively as they are received. The implementation features two highlights. First, the implementation is accelerated with SSE2 and AltiVec SIMD vector instructions on ×86 and PowerPC processors, respectively. Second, a careful threading design is implemented to take advantage of symmetric multiprocessor (SMP) systems and multicore processors. The objective of this work is to explore the computational limits of random network coding in practice using current-generation off-the-shelf processors, and to provide a solid reference implementation to facilitate commercial deployment of network coding.

It has been shown that, with an Intel Core Duo processor running at 1.83 GHz (a state-of-the-art dual core processor for mainstream desktop computers at the time of the work), SSE2-accelerated and multithreaded network coding can process around 5 MB/s, with 128 blocks of 4 kB each in a generation. The result was encouraging, in that a software implementation of network coding on

off-the-shelf desktop processors is able to saturate a typical DSL uplink for home Internet users.

B. Network Coding in Dedicated Servers

But how about dedicated streaming servers? With dual Gigabit Ethernet interfaces, their uplink capacity can reach 2000 Mb/s—or 250 MB/s. If random network coding is to be used in a P2P streaming system, it needs to be implemented on *all* participating peers, including dedicated servers. In the *Nuclei* project, Shojania *et al.* [27] have presented a first attempt towards the implementation of random network coding on GPUs. With a single NVIDIA GeForce 8800 GT, a mainstream GPU at the time of the work, GPU-accelerated random network coding developed using the CUDA platform [28] is able to process 66 MB/s, with 128 blocks of 4 kB each in a generation. This is a substantial performance improvement over Intel dual-core CPUs, thanks to the superb ability of the GPU hardware to launch thousands of parallel computation-bound threads with zero context switching overhead, as well as the embarrassingly parallel nature of random network coding as a computational task.

Although the performance of network coding with GPU acceleration has been much higher than that with multicore CPUs, it is still not able to saturate the uplink bandwidth of a typical server. Moore's law, it turns out, comes to our rescue, combined with even more aggressive optimization with respect to the GPU implementation of network coding. In [29], Shojania *et al.* have pushed the performance envelope to the extremes, by showing that a heavily optimized coding implementation on a single NVIDIA GeForce GTX 280 is able to code at a rate of 294 MB/s, with 128 blocks of 4 kB each in a generation. This level of coding performance is able to saturate two Gigabit Ethernet interfaces, so that random network coding can be performed at *line speed* on streaming servers, pushing fresh coded blocks to downstream peers in a streaming network.

Though the implementation of network coding is still in software, the NVIDIA GeForce GTX 280, the fastest GPU in 2008, featured 240 cores and a peak performance of 360 giga instructions per second (GIPS). When network coding is executed on the GPU, over 90% of the peak computational power provided by the GPU hardware is utilized. The surprising performance of random network coding can be attributed to the fact that such a level of computational power can be available as off-the-shelf commodity hardware, and can be utilized with a software implementation.

C. Network Coding in Mobile Phones

Finally, recent work has shown that random network coding may not be out of the question on computing platforms optimized for power efficiency, such as mobile phones. Shojania *et al.* [30] have shown the surprising result that, by optimizing its implementation to the ARM Cortex-A8 CPU architecture using hand-optimized assembly, an iPhone 3GS with its CPU operated at 600 MHz is able to achieve 1 MB/s with 128 blocks in a generation. Such a high coding rate may open up new opportunities to bring streaming systems with network coding to mobile smartphones. Decoding a high-quality video stream at a streaming bit rate of 768 kb/s, for example, will increase CPU usage by no more than 10%.

V. OPERATIONAL ON-DEMAND STREAMING WITH RANDOM NETWORK CODING

With the application of random network coding in P2P content distribution and live streaming systems that adopt the random gossiping philosophy in random mesh topologies, one may wonder if it can be similarly applied to P2P on-demand streaming systems, also called video-on-demand (VoD) systems. With high-performance implementations of random network coding in commodity hardware from dedicated servers to mobile devices, one may also wonder if network coding can be deployed in large-scale operational P2P streaming systems.

Annapureddy *et al.* [31] have been the first to study the feasibility of applying random network coding in P2P on-demand streaming systems. A P2P on-demand streaming system is conceptually more challenging to design and implement, in that it requires not only smooth sequential playback, but also the shortest possible "restarting" latency, after an interactive *random seek* request is received from the user to relocate the playback point. In the work by Annapureddy *et al.*, similar to live streaming systems, the on-demand media stream is divided into generations, which implies that the initial buffering delay is at least the length of one generation in terms of playback time at the streaming bit rate. In both a simulation and a small prototype, it shows that the throughput achieved with random network coding is higher, compared to a block selection algorithm that downloads globally rarest blocks in the generation first.

Similar to the scenario of bulk content distribution systems, once again, it was not clear whether benefits of random network coding in simulations and small prototypes can easily be carried over to real-world on-demand streaming systems operating at a large scale. Overall, in both content distribution and streaming systems, despite the foundation established by extensive research in the literature, random network coding has not been reported to be deployed in real-world operational systems at a large scale or in a production setting. What are the lingering challenges that prevent such a deployment?

Liu *et al.* [32] have applied random network coding in an operational on-demand streaming system called UUSee, and have presented the first successful attempt that applies the theory of random network coding in large-scale real-world systems. The design objectives were similar to those

of live streaming systems, in that random seek latencies and server bandwidth costs should be minimized, while a consistent and smooth playback quality should be maintained. We now introduce the practical challenges and their corresponding design choices, learned from the experiences in such an operational system.

A. Practical Challenges

In real-world systems, any benefits are accompanied with costs or drawbacks. One seemingly naive question when network coding is to be used in the UUSee on-demand streaming system is how the size of a block should be determined. A coded block is the basic transmission unit used to serve a portion of a generation in the stream. Block-level granularity accommodates slower serving peers so that they can serve at least one block in a reasonable amount of time. It is intuitively preferable to use a smaller block size, such as 1 kB, such that each block directly corresponds to a user datagram protocol (UDP) packet in practice.

This seems to be a good choice if there were no overhead imposed by coding coefficients. Yet with random linear coding on $\mathrm{GF}(2^8)$, each coefficient occupies a byte, and such overhead depends on the number of blocks in a generation. It appears that in order to reduce the overhead due to coding coefficients, one should consider using a smaller number of blocks in each generation.

However, we cannot afford to use too few blocks in a generation, since a smaller number of blocks will lead to a different type of communication overhead, after a generation is completely received and decoded on the receiving peer. Illustrated in Fig. 4, "braking" acknowledgment packets need to be sent back to each of the serving peers, respectively, to stop them from sending more coded blocks. Due to the latency for these braking acknowledgments to be received by serving peers, additional redundant blocks may be received by the receiving peer after the generation is complete, and will need to be discarded. Apparently, as the number of blocks in a generation becomes smaller, the overhead caused by these redundant blocks will become more pronounced due to frequent "braking."

In addition, the need for exchanging buffer availability information between neighboring peers calls for a larger number of blocks in a generation, since a larger generation size reduces the number of bits required to represent buffer availability in an on-demand video stream. The good news is that, with highly optimized implementations of network coding on commodity CPUs, the UUSee system can afford to use a larger number of blocks if it needs to.

B. Design Choices

The primary design goal is to amplify conceptual and theoretical benefits of network coding to the maximum extent possible, and to mitigate its drawbacks in practical use. The following design choices have been made in the UUSee on-demand streaming system with network coding.

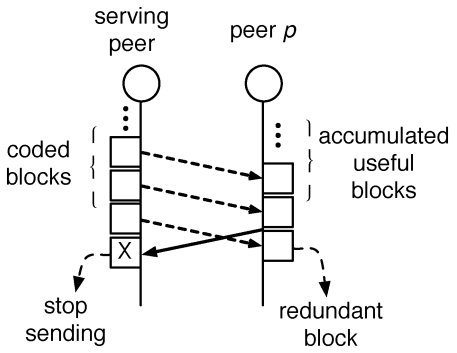

Fig. 4. *The communication overhead due to "braking."*

1) Overhead: Applying the *lesser of two evils* principle, the overhead caused by redundant blocks—received after the generation is completely downloaded by a receiving peer—is a more important concern. To mitigate such overhead, a larger number of blocks, 300 to 500, is used in a generation, with a smaller block size. In the UUSee system, 1 kB is used as the size of a block, corresponding to a single UDP packet. With such a design choice, if coding coefficients are embedded into coded blocks so that they are self-contained, up to 50% overhead is incurred due to the small size of each block, and the large number of blocks in a generation.

Instead of embedding coding coefficients in the coded blocks, UUSee chooses to embed the *PRNG seed* that is used to produce the sequence of random coefficients with a known pseudorandom number generator (PRNG). This effectively reduces the overhead to only 4 B, regardless of the number of blocks, m, in a generation. The only side effect is that peers are no longer able to serve coded blocks to others before completely receiving a generation. However, unlike live streaming, on-demand streaming systems do not have any requirements on the playback lag between live events and their corresponding times of playback. As a result, such a side effect is a nonissue in on-demand streaming.

2) The Push Protocol for Coordinating Multiple Serving Peers: Since random linear network coding is used to make such coordination simple, coded blocks from all serving peers are equally useful. Upon the explicit request from a downstream peer p for blocks in a particular generation, a serving peer starts to *push* freshly generated coded blocks consecutively to p as UDP packets (with flow control), until the "braking" acknowledgment packet arrives. The downstream peer p may send explicit requests to more than one serving peer, and simply sends one acknowledgment packet (as the "stop" signal) to each serving peer

after it has successfully decoded the generation progressively using Gauss–Jordan elimination.

C. Practical Experiences With Network Coding

Liu *et al.* [32] have designed and implemented the first operational on-demand streaming system with random network coding, and have observed an excellent level of real-world streaming performance, based on measurement studies using 200 GB worth of operational traces. It has become evident that the design objectives in on-demand streaming systems have been achieved: multiple serving peers are able to coordinate their actions serving a peer, leading to minimized buffering delays and bandwidth costs on servers. The playback quality has been satisfactory for normal-quality videos. For high-quality videos, the use of network coding has mitigated negative effects when the server bandwidth supply becomes tight in meeting the demand for bandwidth.

VI. CONCLUDING REMARKS

In retrospect, network coding research in the past decade has remained largely theoretical, with respect to the advantages of coding over routing in multicast sessions over directed graphs, as well as the conceptual benefits of using random network coding in P2P networks. With this paper, we wish to introduce important milestones towards bringing theoretical benefits of network coding to practical implementations.

We point out that P2P networks may be the most promising application scenario for network coding to be applied, only because peers are able to afford the increased computational complexity introduced by network coding. We note that the benefits of random network coding in P2P bulk content distribution systems have been extensively evaluated in both simulation and prototype implementation settings, but have not yet materialized in real-world systems.

In contrast, the use of network coding in P2P streaming systems is more realistic, with a recent report that network coding has been successfully applied in an operational on-demand streaming system, operated at a large scale that accommodates millions of Internet users. Thanks to Moore's law and optimized implementations of network coding in software, it is feasible to perform network coding on a wide range of commodity hardware, including servers, desktops, notebook computers, and even mobile phones. As long as the additional energy consumed to perform network coding can be well justified, we may continue to observe more practical systems that incorporate the benefits of network coding in the future. ■

REFERENCES

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

[2] R. Koetter and M. Medard, "An algebraic approach to network coding," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 782–795, Oct. 2003.

[3] S. Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. Inf. Theory*, vol. 49, no. 2, pp. 371–381, Feb. 2003.

[4] P. Sanders, S. Egner, and L. Tolhuizen, "Polynomial time algorithm for network information flow," in *Proc. 15th ACM Symp. Parallelism Algorithms Architectures*, Jun. 2003, pp. 286–294.

[5] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," in *Proc. Int. Symp. Inf. Theory*, 2003, DOI: 10.1109/ISIT.2003.1228459.

[6] P. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. Allerton Conf. Commun. Control Comput.*, Oct. 2003. [Online]. Available: http://sites.google.com/site/saloot2/practical-network-coding.pdf

[7] B. Pittel, "On spreading a rumor," *SIAM J. Appl. Math.*, vol. 47, no. 1, pp. 213–223, 1987.

[8] S. Sanghavi, B. Hajek, and L. Massoulie, "Gossiping with multiple messages," in *Proc. IEEE INFOCOM*, Anchorage, AK, 2007, pp. 2135–2143.

[9] A. Bar-Noy and S. Kipnis, "Broadcasting multiple messages in simultaneous send/receive systems," *Discrete Appl. Math.*, vol. 55, pp. 95–105, 1994.

[10] J. Mundinger, R. Weber, and G. Weiss, "Optimal scheduling of peer-to-peer file dissemination," *J. Scheduling*, pp. 105–120, 2007.

[11] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *Proc. IEEE INFOCOM*, Mar. 2005, vol. 4, pp. 2235–2245.

[12] S. Deb, M. Medard, and C. Choute, "Algebraic gossip: A network coding approach to optimal multiple rumor mongering," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2486–2507, Jun. 2006.

[13] D. Mosk-Aoyama and D. Shah, "Information dissemination via network coding," in *Proc. IEEE Int. Symp. Inf. Theory*, Oct. 2006, pp. 1748–1752.

[14] D. Vasudevan and S. Kudekar. (2009, Jan.). Algebraic gossip on arbitrary networks. [Online]. Available: http://arxiv.org/pdf/0901.1444

[15] M. Borokhovich, C. Avin, and Z. Lotker, "Tight bounds for algebraic gossip on graphs," in *Proc. IEEE Int. Symp. Inf. Theory*, 2010, pp. 1758–1762.

[16] R. W. Yeung, "Avalanche: A network coding analysis," *Commun. Inf. Syst.*, vol. 7, no. 4, pp. 353–358, 2007.

[17] D. Niu and B. Li, "On the resilience-complexity tradeoff of network coding in dynamic P2P networks," in *Proc. 15th IEEE Int. Workshop Quality of Service*, Evanston, IL, Jun. 2007, pp. 38–46.

[18] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, analysis, and modeling of BitTorrent-like systems," in *Proc. ACM Internet Meas. Conf.*, Oct. 2005, pp. 35–48.

[19] C. Gkantsidis, J. Miller, and P. Rodriguez, "Comprehensive view of a live network coding P2P system," in *Proc. 6th ACM Internet Meas. Conf.*, 2006, pp. 177–188.

[20] 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. IEEE INFOCOM*, 2005, vol. 3, pp. 2102–2111.

[21] B. Li, S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, and X. Zhang, "Inside the new coolstreaming: Principles, measurements and performance implications," in *Proc. IEEE INFOCOM*, 2008, pp. 1031–1039.

[22] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang, "Challenges, design and analysis of a large-scale P2P-VoD system," in *Proc. ACM SIGCOMM*, 2008, pp. 375–388.

[23] C. Feng, B. Li, and B. Li, "Understanding the performance gap between pull-based mesh streaming protocols and fundamental limits," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 891–899.

[24] M. Wang and B. Li, "R^2: Random push with random network coding in live peer-to-peer streaming," *IEEE J. Sel. Areas Commun.*, vol. 25, no. 9, pp. 1655–1666, Dec. 2007.

[25] C. Feng and B. Li, "On large-scale peer-to-peer streaming systems with network coding," in *Proc. ACM Multimedia*, Oct. 2008, pp. 269–278.

[26] H. Shojania and B. Li, "Parallelized progressive network coding with hardware acceleration," in *Proc. 15th Int. Workshop Quality of Service*, 2007, pp. 47–55.

[27] H. Shojania, B. Li, and X. Wang, "Nuclei: GPU-accelerated many-core network coding," in *Proc. IEEE INFOCOM*, 2009, pp. 459–467.

[28] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *ACM Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[29] H. Shojania and B. Li, "Pushing the envelope: Extreme network coding on the GPU," in *Proc. 29th Int. Conf. Distrib. Comput. Syst.*, 2009, pp. 490–499.

[30] H. Shojania and B. Li, "Tenor: Making coding practical from servers to smartphones," in *Proc. ACM Multimedia*, Oct. 2010, pp. 45–54.

[31] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez, "Is high-quality VoD feasible using P2P swarming?" in *Proc. 16th Int. World Wide Web Conf.*, 2007, pp. 903–912.

[32] Z. Liu, C. Wu, B. Li, and S. Zhao, "UUSee: Large-scale operational on-demand streaming with random network coding," in *Proc. IEEE INFOCOM*, 2010, DOI: 10.1109/INFCOM.2010.5462030.

ABOUT THE AUTHORS

Baochun Li received the B.Engr. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995 and the M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000, respectively.

Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, Toronto, ON, Canada, where he is currently a Professor. He holds the Nortel Networks Junior Chair in Network Architecture and Services from October 2003 to June 2005, and the Bell University Laboratories Endowed Chair in Computer Engineering since August 2005. His research interests include large-scale multimedia systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks.

Dr. Li was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the IEEE Communications Society, and a recipient of the University of Toronto McLean Award.

Di Niu received the B.Engr. degree from the Department of Electronics and Communication Engineering, Sun Yat-sen (Zhongshan) University, Guangzhou, Guangdong, China, in 2005 and the M.A.Sc. degree from the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada, in 2009, where he is currently working towards the Ph.D. degree at the Department of Electrical and Computer Engineering.

His research interests include measurement, data mining and implementation of large-scale multimedia systems, peer-to-peer networks, and applications of network coding.