Large File Transmission in Network-Coded Networks with Packet Loss – A Performance Perspective

Shenghao Yang
Institute of Network Coding
The Chinese University of Hong Kong
shyang@inc.cuhk.edu.hk

Raymond W. Yeung*
Institute of Network Coding &
Department of Information Engineering
The Chinese University of Hong Kong
whyeung@ie.cuhk.edu.hk

ABSTRACT

Network coding can significantly improve the transmission rate of communication networks with packet loss compared with routing. But using network coding usually involves more computational and storage costs in network devices and terminals. We discuss some recent schemes for file transmission in networks employing coding and compare the computational and storage costs of these schemes. The up-to-date state is that network coding is nearly ready for practical applications in large file transmission.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Network Communications*

General Terms

Algorithms

Keywords

Network coding, network communications

1. INTRODUCTION

Data packets transmitted in a communication network can be erased due to channel noise, congestion, faulty network hardware, and so on. In a routing network, where intermediate nodes only forward received packets, the transmission from a source node to a destination node can naturally be modeled as an erasure channel. Hence, various erasure correction techniques have been proposed and used in existing communication networks. When feedback is assumed from the destinations, the source node can just retransmit the erased packets. Retransmission can achieve the maximum rate of an erasure channel and is used in network protocols, for example TCP, to provide reliable end-to-end transmission.

When there is no feedback, it is also possible to achieve reliable end-to-end transmission using properly designed erasure codes. There exist erasure codes achieving the maximum rate of an erasure channel, for example. Reed-Solomon codes. Using erasure codes usually increases the computational cost of the source and destination nodes. Several class of efficient erasure codes were proposed under the fountain codes framework [2], including LT codes [11], Raptor codes [16], and online codes [13]. For example, Raptor codes are capacity achieving and have linear encoding and decoding complexities in terms of the number of packets for transmission.

Routing, however, is not an optimal operation at the intermediate nodes from the throughput point of view. For example, the routing capacity of the network in Fig. 1 is 0.64 packet per use. If we allow decoding and encoding operations at the intermediate node and treat the network as a concatenation of two erasure channels, we can achieve the rate 0.8 packet per use by using erasure codes on both links. For a general network with packet loss, the maximum multicast rate from a source node to a set of destination nodes is achieved by using erasure codes link-by-link and *network coding* in the network layer [1]. Network coding allows an intermediate node generating and transmitting new packets using the packets it has received. Linear network coding [9] was proved to be sufficient for multicast communications and can be realized distributedly by random linear network coding [8].

When using network coding, the link layer erasure codes are not necessary. The following network coding method has been proved to achieve the multicast capacity for networks with packet loss in a wide range of scenarios [12]. The source node transmits random linear combinations of the input packets and an intermediate node transmits random linear combinations of the packets it has received. Note that no erasure codes are required for each link though packet loss is allowed. Network coding itself plays the role of end-to-end erasure codes. A destination node can decode the input packets when it receives enough coded packets with linearly independent coding vectors.

However, the above scheme, referred to as the ordinary random linear network coding scheme, has computational and storage complexities that are not suitable for practical applications. Consider transmitting K packets and each packet consists of T symbols in a finite field. The computational complexity of encoding in the source node is $\mathcal{O}(TK)$

*Corresponding author.

ISABEL '11 Barcelona, Catalonia Spain

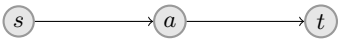

Figure 1: In this network, s is the source node, a is the relay node, and t is the destination node. Both links are can transmit one packet per use with a packet loss rate 0.2**.**

per packet. An intermediate node needs to buffer all the packets it has received for network coding, so in the worst case, the storage cost is K packets, and the computational complexity of encoding is $\mathcal{O}(TK)$ per packet. Decoding using Gaussian elimination has complexity $\mathcal{O}(K^3 + TK^2)$, or $\mathcal{O}(K^2 + TK)$ per packet. Though these complexities are polynomials in K, the ordinary random linear network coding scheme is still difficult to implement for large K.

In practice, we hope to have generic network coding enabled devices with limited storage and computational capabilities. Accordingly, a network coding scheme is hoped to have i) low encoding complexity in the source node, ii) low decoding complexity in destination nodes, iii) constant computational complexity of encoding a packet in an intermediate node, and iv) constant storage requirement in an intermediate node.

In the rest of this paper, we review some recent attempts for designing efficient file transmission schemes in networks with coding at the intermediate nodes. There are roughly two lines of works. The first line of works tries to extend fountain codes to networks with coding at the intermediate nodes. Solutions exist for special network topologies (e.g., line networks) and special communication scenarios (e.g., peer-to-peer file sharing), but those solutions cannot meet our requirement in a general network setting. The second line of works try to simplify the complexities of linear network coding using chunks. Most recent developments of chunk based codes target the application of peer-to-peer file sharing, where the storage cost at an intermediate node is not considered as an issue. At the end, we show that a good scheme exists in the place where the two lines intersect. By extending the idea of LT/Raptor codes from erasure channels to networks employing linear network coding, a new class of fountain codes, called batched sparse (BATS) codes, are proposed [19]. BATS codes work naturally with linear network coding and have all the desired properties.

2. FOUNTAIN CODES WITH CODING IN NETWORKS

2.1 Fountain Codes

LT codes are a class of fountain codes introduced by Luby for erasure channels [11]. An LT encoder generates *output packets* from K *input packets* using a degree distribution $\Psi = (\Psi_0, \Psi_1, \cdots, \Psi_K)$. Every time an output packet is generated, the degree distribution is sampled and an integer value d is returned with probability Ψ_d. Then d distinct input packets are chosen randomly and they are added together to yield the output packet.

An LT decoder uses any n output packets to recover the original K input packets. A decoding graph is a bipartite graph with K nodes on one side and n nodes on the other side, which correspond to the input packets and output packets, respectively. There is an edge between an input packet and an output packet if the input packet contributes to the value of the output packet. At each step, the decoder identifies an output packet of degree one, which is just the value of its unique neighbor among the input packets. Then the value of the decoded input packet is substituted into its neighboring output packets. Luby proposed a robust soliton distribution for Ψ that guarantees successful decoding with n slightly larger than K.

Raptor codes [16] further reduce the complexity of LT codes by precoding. The input packets are first encoded by an erasure code, the output of which is called the intermediate packets, and then the intermediate packets are encoded by a variation of LT code. This variation of LT code only recover a constant fraction of the intermediate packets and the erasure code is capable of recovering all the input packets in face of a fixed fraction of erasures.

2.2 Link-by-Link Fountain Codes

A line network is a network with a sequence of nodes connected consecutively, where the first node is the source and the last node is the destination. Figure 1 is an example of a line network. One coding scheme for a line network is to apply fountain codes link by link (see the discussion in [15]). An intermediate node must completely decode and re-encode all the input packets. Each intermediate node requires a storage to buffer at least K packets. Moreover, the transmission suffers a delay of $\mathcal{O}(K)$ due to the decoding and encoding operations at each intermediate node.

To reduce the storage cost at the intermediate nodes, Pakzad et al. [15] proposed several schemes using systematic codes. In their schemes, however, the storage required depends on the erasure probabilities of the links. Though the storage cost can be reduced to a fraction of K, it still increases linearly with K.

The delay for applying fountain codes link by link can be reduced by applying fountain codes in a stack manner [5]: An intermediate node buffers the packets it receives and re-encodes them to approximate a fountain code. The difficulty of this method is to approximate a foundation code by using a subset of the input packets. While this method moves all the decoding operations to the destinations nodes, it cannot reduce the storage cost at the intermediate nodes.

2.3 End-to-End Fountain Codes

The high decoding complexity of the ordinary random linear network coding scheme comes from the use of Gaussian elimination to solve a system of linear equations. One way to resolve the complexity issue is to solve the system of linear equations by using belief propagation, like the decoding of an LT code. Champel et al. [3] proposed this for a peer-to-peer sharing model where all network nodes require the file. The input packets are encoded not only at the source node but also at the intermediate nodes in a decentralized manner so that the resulting receiving packets resemble an LT code. Their methods require special intermediate operations, and an intermediate node is required to buffer all the packets it receives. Designing "sparse" network coding to approximate fountain codes may be possible for special communication scenarios, e.g., [3], but in general it is difficult to guarantee that the degree of the received packets follows a particular distribution.

3. CHUNK BASED NETWORK CODING

3.1 Using Chunks to Reduce Complexity

A chunk (also called generation or class) is a subset of the K packets. One practical method to simplify the complexities of network coding is to group the input packets into disjoint chunks [4]. Encoding at the source node, network coding at the intermediate nodes, and decoding at a destination node are all performed for packets belong to the same chunk. Using disjoint chunks reduces the encoding and decoding complexities to $\mathcal{O}(TKL)$ and $\mathcal{O}(KL^2 + TKL)$, respectively, when all the chunks have the same size L.

In practice, for the sake of complexity, the chunk size L cannot be too large. For example, $L = 100$ is used in [4] and $L = 16$ is used in [6]. However, a file of 100 MB may have 10^5 UDP packets and hence may have 1000 chunks when $L = 100$. Therefore for large files, the practical number of chunks is also large.

Using disjoint chunks decomposes the task of transmitting a large file into subtasks of transmitting small files. This can increase the protocol overhead significantly when the number of chunks is large. In the following two sections, we discuss two approaches to scheduling the transmission of the chunks and their respective issues.

3.2 Sequential Scheduling of Chunks

If feedback from the destinations is allowed, sequential scheduling of chunks can be used. In this approach, the source node keeps transmitting packets belonging to one chunk until all the destination nodes have successfully decoded the chunk. The source node needs positive feedbacks from all the destination nodes before moving on to the next chunk. The number of feedbacks required for each chunk increase with the number of destination nodes.

Feedback generally degrades the system performance due to the round-trip delay. This degradation can be severe if the round-trip delay is long and/or some feedbacks are lost along the way. If a link in the network is half-duplex, the transmission rate needs to be reduced in order to accommodates the feedbacks.

Besides the protocol overhead, sequential scheduling is also not scalable for multicast, because when there is a large number of destination nodes, it will take a long time for all of them to successfully decode a chunk before the source node can move on to the next chunk.

3.3 Random Scheduling of Chunks

To resolve the issues in sequential scheduling, chunks can be scheduled randomly [14]. When the source node or an intermediate node needs to generate a packet for transmission, the node randomly picks a chunk and produces a new packets using that chunk. (The chunks can also be scheduled by round robin [4]. However, we will not discuss this scheduling particularly, because it shares the same properties with random scheduling.)

In [14], an adversarial schedule is used as the network model and random scheduling is shown to asymptotically achieving the capacity of the adversarial schedule when the number of packets goes to infinity and the chunk size is bounded below by an increasing function of the number of packets. For practical chunk sizes, performance of random scheduling of disjoint chunks has been observed to be suboptimal [17, 6].

Random scheduling becomes less efficient as the fraction of decoded chunks increases. For example, when more than half of the chunks have already been decoded by a destination node, more than half of the transmissions from the source node are useless for this destination node. As an alleviation of this problem, precoding like Raptor codes are introduced [14]. The input packets are first encoded by an erasure code, and then the intermediate packets are transmitted in chunks. Precoding allows the input packets to be recovered when only a fraction of the set of chunks have been successfully decoded. However, to have high coding rate, the rate of the erasure code (precode) should be high and a large fraction of chunks are required to be decoded individually. Thus using precoding cannot completely eliminate the issue.

Moreover, for random scheduling the intermediate nodes are required to buffer all the chunks. This may not be an issue for peer-to-peer file distribution, but it is not suitable for networks with intermediate nodes that do not require the file. The schemes discussed in the next section [17, 6, 10], which address the issue in random scheduling discussed in the last paragraph by employing overlapped chunks, share the same storage cost at the intermediate nodes as random scheduling discussed in this section.

3.4 Overlapped Chunks

Two groups [17, 6] have independently shown by simulation that using overlapped chunks can improve the throughput of random scheduling for practical chunk sizes. Intuitively, the advantage of overlapped chunks is to use the already decoded chunks to help the decoding of the other chunks.

Silva et al. [17] proposed two designs of chunked codes, rectangular grid codes and diagonal grid codes. For example, a rectangular grid code is constructed by placing the input packets in a rectangular grid and forming chunks using the packets in the same rows or columns. Heidarzadeh and Banihashemi [6] used a head-and-toe overlapping pattern for chunked codes and analyzed its asymptotic performance over a line network when the number of packets goes to infinity. Theoretical comparison of disjoint chunks and overlapped chunks for line networks can be found in [7]. Li et al. [10] proposed random annex codes which are formed by adding a number of randomly chosen packets to disjoint chunks. Random annex codes are analyzed heuristically using a coupon collection model. Note that the same probability for the chunks with the same size is used in almost all analysis and simulations in [17, 6, 10].

Overlapped chunks have some properties similar to fountain codes, but compared with fountain codes, sophisticated designs and complete performance analysis are still lacking.

4. BATCHED SPARSE CODES

To address the issues of the schemes discussed in Sections 2 and 3, a class of generalized LT/Raptor codes, called batched sparse (BATS) codes, are proposed [19]. BATS codes are designed to be used with networks employing linear network coding, while preserving the properties of fountain codes such as ratelessness and low encoding/decoding complexity.

4.1 Encoding of Batches

A *batch* is a set of M coded packets generated from a subset of the input packets. The encoding of BATS codes is similar to LT codes, except that batches instead of coded packets are generated. When $M = 1$, BATS codes become

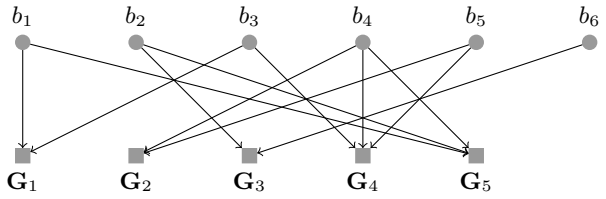

Figure 2: The Tanner graph for encoding the first five batches. Here, b_1, $i = 1, 2, \cdots$ are the input packets, and $\mathbf{G}_j$, $j = 1, 2, \cdots$ are the generator matrices.

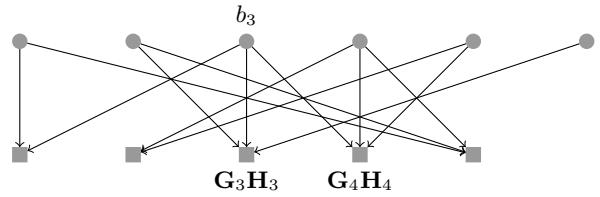

Figure 3: Assume that the third batch is decodable. Then the contributors of the third batch are recovered by solving a linear system of equations. Here, b_3 is contributor of the third batch. Since b_3 is also a contributor of the fourth batch, besides removing the node b_3, the row in $\mathbf{G}_4\mathbf{H}_4$ corresponding to b_3 and hence the edge from node b_3 to the fourth batch are removed.

LT codes.

Specifically, a batch is generated using the following procedure. First, a degree distribution $\Psi = (\Psi_0, \Psi_1, \cdots, \Psi_K)$ is independently sampled and a degree d is returned with probability Ψ_d. Second, uniformly at random pick d distinct input packets, which are called the contributors of the batch. Last, the M coded packets of the batch are generated using possibly different linear combinations of the contributors. The last step can be equivalently specified by a generator matrix with dimension $d \times M$. There are various options for designing the generator matrices. For example, the generator matrices can be deterministic and pre-designed, or they can be generated on the fly. See Figure 2 for the Tanner graph describing the encoding of a BATS code.

To transmit a batch, the source node transmits the packets in the batch. No feedback is required to stop the transmission of each batch. Transmission of the batches at the source node and the intermediate nodes can be scheduled in different ways. For example, for file transmission in a line network, sequential scheduling of the batches can minimize the buffer requirement at the intermediate nodes. For a general network, how to schedule the transmission of batches at the intermediate nodes is an open research problem.

4.2 Belief Propagation Decoding

When applying linear network coding, an intermediate node encodes the received packets of a batch into new packets using linear combinations. These new packets are regarded as belonging to the same batch. The packets of a batch generated in the source node will be thereafter referred to as the original packets of the batch. The rule is that the packets in different batches are not mixed inside the network.

Following this rule, the received packets of a batch are linear combinations of the original packets of the batch. Let G be the generator matrix of a batch and H be the transfer matrix of the network from the original packets to the received packets. The overall transformation from the contributors of the batch to the received packets is GH, which is known by a destination node.

The decoding process can be described by the bipartite graph in Figure 3, which is the same as the encoding graph except that associated with the ith batch is the matrix $\mathbf{G}_i\mathbf{H}_i$, an instance of GH. The ith batch is called decodable if the rank of $\mathbf{G}_i\mathbf{H}_i$ is equal to the degree of the batch d_i. After decoding a batch, the decoded input packets are substituted in the undecoded batches. In the decoding graph, this is equivalent to remove the nodes corresponding to the decoded packets and the related edges. We repeat this decoding-substitution procedure on the new graph until no more batches are decodable.

The same technique of Raptor codes can be applied here to reduce the encoding/decoding complexity of BATS codes. The input packets are first encoded using an erasure code, and then encoded by a BATS code. The belief propagation decoding of the BATS code is required to recover a constant fraction of the intermediate packets (the output of the precoder). The erasure code can then decode the input packets in face of a fixed fraction of erasures of the intermediate packets.

4.3 Degree Distribution

The degree distribution of a BATS code needs to be chosen such that i) the belief propagation decoding succeeds with high probability, ii) the encoding/decoding complexity is low, and iii) the coding rate is large.

By analyzing the decoding process of BATS codes, a sufficient condition of a degree distribution such that the belief propagation decoding succeeds with high probability is obtained. This sufficient condition induces a linear programming for finding a degree distribution that maximizes the asymptotic achievable rate. The only channel knowledge required for the linear programming is the rank distribution of the network transfer matrix.

It is verified theoretically for certain cases and demonstrated numerically for the general cases that BATS codes achieve rates very close to the capacity of a linear operator channel, which models the transmission of a network employing linear network coding [18].

The degrees of the batches affect the encoding/decoding complexity. For an optimal degree distribution, the maximum degree with non-zero probability is bounded by $\mathcal{O}(M)$. Thus the encoding/decoding of a batch has complexity independent of K. The complexity comparison of BATS codes and chunk based codes are given in Table 1.

4.4 Batches Chunks

Even though both BATS codes and chunked codes require network coding to be applied within one batch or one chunk, batches and chunks are different in many aspects.

BATS codes are rateless code, i.e., the number of batches that can be transmitted is not fixed. By contrast, the number of chunks is fixed after encoding. The source node only transmits the original packets of a batch, so the transmission of a batch uses a fixed time. But the transmission of a chunk is in a rateless manner for both random and sequential

Table 1: Complexity comparison. The encoding/recoding/decoding complexity is in terms of per packet.

	source node encoding	intermediate node		destination node decoding
		recoding	storage	
routing	$\mathcal{O}(T)$	$\mathcal{O}(T)$	$\mathcal{O}(T)$	$\mathcal{O}(T)$
sequential scheduling of chunks	$\mathcal{O}(TL)$	$\mathcal{O}(TL)$	$\mathcal{O}(TL)$	$\mathcal{O}(L^2+TL)$
random scheduling of chunks	$\mathcal{O}(TL)$	$\mathcal{O}(TL)$	$\mathcal{O}(TK)$	$\mathcal{O}(L^2+TL)$
BATS code	$\mathcal{O}(TM)$	$\mathcal{O}(TM)$	$\mathcal{O}(TM)$	$\mathcal{O}(M^2+TM)$

scheduling.

Sequential scheduling of disjoint chunks requires feedback to signify that a chunk is successfully decoded by all the destination nodes, while BATS codes do not need feedback for this purpose. Instead, BATS codes only require a small fraction of the batches to be decodable to trigger the belief propagation decoding procedure. This small fraction of decodable batches can be guaranteed, for example, by using small enough degrees.

When using sequential scheduling, BATS codes do not require an intermediate node buffering all batches for acyclic networks. Since packets of a same batch are transmitted consecutively, no new packets of a batch will come after certain time of the first packet of that batch is received. So there will be no harm to delete a batch after buffering it for enough long time. The overlapped chunked schemes [17, 6, 10] are designed to solve the issue of random scheduling scheme in [14], and hence assume random scheduling. How to sequentially schedule the chunks without feedbacks is not clear. One method is to use a round-robin order to scheduling the chunks. However, this method, similar to random scheduling, requires that an intermediate node buffers all the chunks. Since the packets of all chucks can come in the future, buffering a fraction of all the chunks will harm the benefits of network coding.

5. ACKNOWLEDGMENTS

This work was partially supported by a grant from the University Grants Committee of the Hong Kong Special Administrative Region, China (Project No. AoE/E-02/08).

6. REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. Inform. Theory*, 46(4):1204–1216, July 2000.

[2] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of the ACM SIGCOMM '98*, pages 56–67, New York, NY, USA, 1998.

[3] M.-L. Champel, K. Huguenin, A.-M. Kermarrec, and N. L. Scouarnec. LT network codes. Research Report RR-7035, INRIA, 2009.

[4] P. A. Chou, Y. Wu, and K. Jain. Practical network coding. In *Proc. Allerton Conf. con Comm., Control, and Computing*, Oct. 2003.

[5] R. Gummadi and R. Sreenivas. Relaying a fountain code across multiple nodes. In *Proc. ITW '08*, pages 149–153, 2008.

[6] A. Heidarzadeh and A. H. Banihashemi. Overlapped chunked network coding. In *Proc. ITW '10*, pages 1–5, 2010.

[7] A. Heidarzadeh and A. H. Banihashemi. Analysis of overlapped chunked codes with small chunks over line networks. In *Proc. IEEE Inte. Symp. on Information Theory ISIT '11*, Saint Petersburg, Russia, 2011.

[8] T. Ho, B. Leong, M. Medard, R. Koetter, Y. Chang, and M. Effros. The benefits of coding over routing in a randomized setting. In *Proc. IEEE ISIT'03*, June 2003.

[9] S.-Y. R. Li, R. W. Yeung, and N. Cai. Linear network coding. *IEEE Trans. Inform. Theory*, 49(2):371–381, Feb. 2003.

[10] Y. Li, E. Soljanin, and P. Spasojevic. Effects of the generation size and overlap on throughput and complexity in randomized linear network coding. *Information Theory, IEEE Transactions on*, 57(2):1111 –1123, feb. 2011.

[11] M. Luby. LT codes. In *Proc. 43rd Ann. IEEE Symp. on Foundations of Computer Science*, pages 271–282, Nov. 2002.

[12] D. S. Lun, M. Médard, R. Koetter, and M. Effros. On coding for reliable communication over packet networks. *Physical Communication*, 1(1):3–20, 2008.

[13] P. Maymounkov. Online codes. Technical report, NYU, Nov. 2002.

[14] P. Maymounkov, N. J. A. Harvey, and D. S. Lun. Methods for efficient network coding. In *Proc. Allerton Conf. Comm., Control, and Computing*, Sept. 2006.

[15] P. Pakzad, C. Fragouli, and A. Shokrollahi. Coding schemes for line networks. In *Proc. Int. Symp. Information Theory ISIT 2005*, pages 1853–1857, 2005.

[16] A. Shokrollahi. Raptor codes. *IEEE Trans. Inform. Theory*, 52(6):2551–2567, Jun. 2006.

[17] D. Silva, W. Zeng, and F. R. Kschischang. Sparse network coding with overlapping classes. In *Proc. Workshop Network Coding, Theory, and Applications NetCod '09*, pages 74–79, 2009.

[18] S. Yang, S.-W. Ho, J. Meng, and E. hui Yang. Linear operator channels over finite fields, 2010.

[19] S. Yang and R. W. Yeung. Coding for a network coded fountain. In *Proc. IEEE Inte. Symp. on Information Theory ISIT '11*, Saint Petersburg, Russia, 2011.