Optimality of Network Coding with Buffers

Bernhard Haeupler
RLE, CSAIL
Massachusetts Institute of Technology
Email: haeupler@mit.edu

MinJi Kim
RLE
Massachusetts Institute of Technology
Email: minjikim@mit.edu

Muriel Médard
RLE
Massachusetts Institute of Technology
Email: medard@mit.edu

***Abstract*—We analyze distributed and packetized implementations of random linear network coding (PNC) with buffers. In these protocols, nodes store received packets to later produce coded packets that reflect this information.**

We show the optimality of PNC for any buffer size; i.e., we show that PNC performs at least as good as *any* protocols with the same buffer size. In other words, a multicast task completes at *exactly* the first time in which *in hindsight* it was possible to route information from the sources to each receiver individually given the buffer constraint, i.e., that the buffer used at each node never exceeds its buffer size.

This shows that PNC, even without any feedback or explicit buffer management, allows to keep minimal buffer sizes while maintaining its optimal performance.

I. Introduction

It is a classical result [1] that linear network coding is optimal for multicast in any (acyclic) network and furthermore that even choosing a random linear code suffices with high probability [2], [3]. This rateless and self-adaptive nature of random linear network coding has been shown particularly beneficial in distributed settings with unstable or time-varying network topologies. For these settings, a distributed and packetized network coding (PNC) implementation has been proposed [4], [5] in which nodes buffer all received packets and forward random linear combinations of these packets whenever they send a packet. The performance of PNC has since been intensely studied [5]–[13].

More recently, PNC variants with *finite* buffers [14]–[16] have been studied. Instead of storing and performing each coding operation over all received packets, these variants only buffer a small number of packets. This significantly lowers the memory required and the computational complexity. Nevertheless, the authors could show [16] that in many settings the same order-optimal performance guarantees as for the PNC protocol with infinite buffers are obtained.

This paper strengthens these results and shows that, for *any* buffer size and on *any* network, the PNC protocol performs optimally, i.e., as good as any other protocol with the same amount of buffer can possibly perform. For the case of finite buffers this shows that PNC, even without any feedback or explicit buffer management, allows to keep minimal buffer sizes without any loss in performance. In the case of the original PNC protocol with infinite buffers, the optimality does not come as a surprise and has been stated before. Nevertheless, to our knowledge, this is the first formal proof for its validity.

This material is based upon work supported by the Defense Advanced Research Project Agency (DARPA) and Space and Naval Warfare Systems Center Pacific under Contract No. N66001-11-C-4003. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Project Agency and Space and Naval Warfare Systems Center Pacific.

II. Related Work

For PNC with infinite buffers, [9]–[12] give bounds on the performance of (gossip) protocols in various network settings. In some cases these upper bounds come within constants to the lower bounds; thus, showing the order optimality of PNC in expectation. References [5]–[8] show more strongly that, in any static network with random losses, the performance of PNC asymptotically matches the expected capacity up to an arbitrarily small factor.

The pointwise and exact optimality of PNC in any network was stated in [17] and in [18, p. 475] with a reference to [2] but without a formal proof. The rather subtle problem in applying [2] is that it only applies to acyclic networks with stateless and memoryless nodes. To obtain an acyclic graph one can consider the time-expanded graph of the node transmissions/interactions and add extra infinite capacity edges between each node and its next copy in time. Unfortunately, while these edges seem to represent the fact that the nodes store all received information over time, they do not have the same operational meaning as edges in the acyclic networks of [2] and the min-cut result of [2] does not directly extend to networks with these infinite capacity edges. While it may be plausible to alter the proof of [2] to take this concern into consideration, we take a different approach. Our proofs consist of a simple hypergraph transformation that exactly captures how the nodes use their buffers. This reduction allows us to apply the results of [2] in a simple black-box manner. Furthermore, our reduction naturally extends to the PNC protocols with finite buffers, which are the main focus of this paper.

Using random linear network coding with limited buffers was first considered in [14], albeit only for communication in a two-hop network. In [15] this analysis was extended to line-networks and an approximation scheme for determining throughput and delay was given. In [16] the authors introduce the use of network coding with finite buffers for general topologies and shows that, in many settings, buffering only one packet already leads to the same order optimal stopping times for multicast as that of the PNC variant with infinite buffer. Several other works propose schemes to reduce memory

requirements and the amount of data over which coding is performed; [19], [20] show that buffers can be reduced by deleting information that is already common to all receivers. Unfortunately, this approach can not work in unstable or dynamic networks without a non-trivial feedback mechanism. Furthermore, the reduced buffer size achieved using this method is often far from the theoretically minimal buffer required. Spatial buffer multiplexing is another method to reduce buffer sizes. Reference [21] shows that large networks act as a shared buffer under the strong assumption that the length of flow paths and the number of flows through each node both grow polynomially.

Lastly, Theorem 3.9. in [13] states a result similar to our main theorem for PNC but with a weaker bound. As here adversarial schedules are used to prove bounds that hold pointwise in any oblivious network setting. Instead of proving PNC to be exactly capacity achieving with failure probability $\epsilon = 1/poly(n)$, their result requires at least $k+p\cdot l_{\max}\cdot(\log k+\log \epsilon^{-1})$ capacity to guarantee the delivery of k degrees of freedom, where p is the number of flow path and $l_{\max}$ is the maximum length of one such path. In general, p and $l_{\max}$ can be of the order of the capacity k itself or even larger; making this bound quite loose.

III. Network Coding Review

In this section, we briefly review the classical network coding results and describe its adoption to a packetized distributed setting, the PNC protocol.

A. Memoryless Network Coding Setting

In the memoryless network coding setting [1]–[3], a directed acyclic circuit processes messages from a finite field $\mathbf{F}_q$ (or alternatively $\mathbf{F}_q^l$). A circuit is a directed acyclic hypergraph $\mathcal{C} = (V, A)$. For each node $v \in V$, we denote $\Gamma^+(v)$ as the incoming hyperedges, and $\Gamma^-(v)$ as the outgoing hyperedges. For each $e \in \Gamma^-(v)$, v contains a coding vector $c_e \in \mathbf{F}_q^{\Gamma^+(v)}$. We assume that there is only one node with exclusively outgoing hyperedges, the source node $s \in V$. Assuming an assignment of a message $val(e) \in \mathbf{F}_q$ to each hyperedge $e \in \Gamma^-(s)$, the circuit $\mathcal{C}$ processes information as follows. Each hyperedge can inductively be assigned a message in $\mathbf{F}_q$ by using the rule that the vector associated with an outgoing hyperedge e of v is $c_e \cdot val(\Gamma^+(v))$. In this way, $\mathcal{C}$ defines a linear transform $T(\Gamma^-(s), E') \in \mathbf{F}_q^{\Gamma^-(s)\times E'}$ between the messages $val(\Gamma^-(s))$ and the messages assigned to any subset of hyperedges $E' \subseteq E$.

Reference [1] shows that if the field size q is large enough, one can choose the c_e such that the rank of $T(\Gamma^-(s), E')$ is equal to the min-cut between s and E' in $\mathcal{C}$. Then, any node v with a min-cut of at least $|\Gamma^-(s)|$ can solve the linear system described by $T(\Gamma^-(s), \Gamma^+(v))$ and decode all messages. Furthermore, [2], [3] show that, with high probability, this remains true even if the coding coefficients are chosen uniformly at random. These are the classical results on (random linear) network coding that started this line of research.

Note that, in this model, timing is irrelevant and each node processes each message only once. References [1], [3] show that this setting can be extended to non-acyclic circuits with delays. Nonetheless, nodes remain stateless and memoryless, which is why we refer to these networks as circuits.

B. PNC: Distributed Packetized Network Coding

We introduce the PNC protocol [4], [5] in which, in contrast to the memoryless setting, nodes buffer received information to later produce coded packets reflecting this information.

Assume that there are k messages from $\mathbf{F}_q^l$ distributed to the nodes. If the PNC protocol is used in a network, any node u communicates by sending packets that contain vectors from $\mathbf{F}_q^{k+l}$ and maintains a subset $S_u \subset \mathbf{F}_q^{k+l}$ of received packets. Initially, S_u is empty for all nodes u. When node u initially knows the i^{th} message $s_i \in \mathbf{F}_q^l$, it adds the vector (e_i, s_i) to S_u, where e_i is the i^{th} unit vector in $\mathbf{F}_q^k$. If node u is requested to send a packet it sends a random vector from the span of S_u. Note that this description is completely independent of any assumption on the network.

If enough communication takes place among nodes for the system to "mix", then for each node u the subspace spanned by S_u will converge to the k dimensional subspace of $\mathbf{F}_q^{k+l}$ given by the k input vectors. Each node can then use Gaussian elimination to recover the input messages.

References [9]–[12] provide upper bounds on how quickly this "mixing" happens for specific (stochastic) communication models. In this work, we prove a stronger statement that the mixing happens with high probability in optimal time for *any* communication history.

IV. Network Model: Time Expanded Hypergraphs

We consider discrete or continuous time dynamic network topologies where communication links are established synchronously and/or asynchronously. Nodes can potentially send data at different and highly non-regular rates. Links are assumed to have varying delays. We also incorporate broadcast constraints that arise in wireless settings. Our model applies to any static or stochastic model, including arbitrary stochastic link failures, and to adversarial worst-case communication schedules chosen by an oblivious adversary. All these models specify a (distribution over) communication schedules that is independent from the randomness in the coding coefficients. We shall prove a pointwise optimality, i.e., for any instance of a communication schedule, PNC achieves optimal performance. Therefore, throughout the rest of the paper, we assume that there is a specific given communication schedule on which we have to give an optimality proof.

Each communication schedule can be specified as a sequence of *events*, where a node sends or receives packets. We assume that, at each time, a node either transmits or receives a packet. We capture these events using the following definition of a time expanded communication hypergraph. This notion of time expanded hypergraph has been previously used under different names, e.g., continuous trellis [8] or adversarial schedule [13].

Definition IV.1 (Time Expanded Hypergraph)**.** *Consider a network with n nodes, and denote this set of nodes as V. A communication schedule from time 0 to t among nodes in V is captured by the following time expanded hypergraph $G = (V, V', A)$. Let $v \in V$ be a node in the network. We create a vertex copy $v_{t'} \in V'$ for every time $t' \in [0, t]$ when the node v receives or sends at least one packet. If v is transmitting at time t' to nodes $u^1, u^2, ..., u^b$ with associated delay $\Delta_1, \Delta_2, ..., \Delta_b$ respectively, we create a single hyperedge $(v_{t'}, \{u^1_{t'+\Delta_1}, u^2_{t'+\Delta_2}, ..., u^b_{t'+\Delta_b}\}) \in A$.*

Given a network, we consider the following (distributed) many-to-many multicast problem. Messages are generated at nodes in the network. A message can be generated at (multiple) different times at multiple nodes. The goal is to disseminate all the messages to all nodes (or a subset of *destination nodes* $D \subseteq V$) as fast as possible. One example of an application of this problem could be a source distributing a large file (which is divided into small parts) to many receivers. Another application is in sensor networks, where each sensor transmits its measurements at different times.

To formalize this problem, we assume that there are exactly k messages that are vectors of $\mathbf{F}_q^l$. We assume that the nodes employ the PNC protocol of Section III-B. Note that this requires each message to have a unique identifier that is known to every node at which the messages is generated. We incorporate the message generation in our network model using the following additional definition.

Definition IV.2. *Let $G = (V, V', A)$ be a communication schedule of a network in which k messages $m_1, \ldots, m_k \in \mathbf{F}_q^l$ are generated. We alter G by adding a supersource node s to V'. Furthermore for each message m_i that is generated by nodes $u^1, u^2, ...$ at time $t_1, t_2, ...$ we add a hyperedge $(s, \{u^1_{t_1}, u^2_{t_2}, ...\})$ to A.*

V. Our Results

Given an adversarial schedule and an initial message distribution, the network capacity between the source and any node at any time can be determined. To do so, one enriches the time expanded hypergraph by memory-edges, which capture the possibility that nodes *store* knowledge over time. This is achieved by connecting each node v_t in the time expanded hypergraph to its next copy in time $v_{t'}$ with an edge with capacity equal to the amount of information that v can store, i.e., its buffer size μ (in packets). We assume for simplicity that all nodes have the same amount of memory μ. If all nodes have unlimited buffers, we follow [8] and set $\mu = \infty$. We call this enriched time expanded hypergraph the *(natural) information flow graph* and denote it by G_μ. The next lemma confirms the intuition that the information flow graph indeed represents an upper-bound on the amount of information that can be transmitted by *any* protocol.

Lemma V.1. *Let G be the time expanded hypergraph for a communication schedule and let G_μ be its natural information flow graph. The min-cut between the supersource s and a node v_t in G_μ is an upper bound on the amount of information that* any *protocol can transmit from the sources to node v by time t if all nodes have an active memory of at most μ.*

Given this simple min-cut bound on the achievable point-to-point capacity the interesting question is which protocols achieve it. It is tempting to apply the results form the memoryless setting [2] to conclude that PNC and its finite buffer variants achieve this bound. Unfortunately this is not valid. It is, e.g., not difficult to find protocols that do not achieve this capacity, e.g., the shift-register finite memory network coding protocol in [14]. In the what follows we provide a correct, general and simple approach to base the optimality of protocols on [2]:

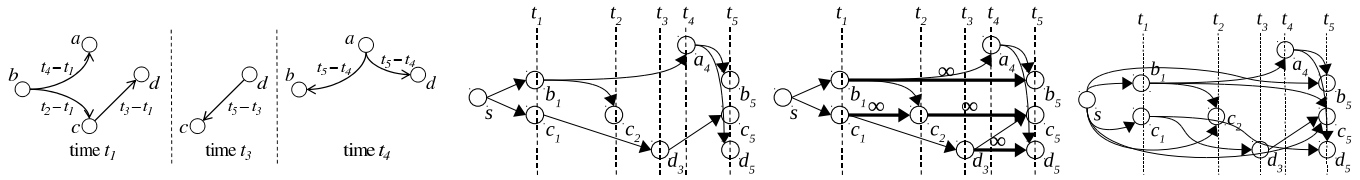
A. General Approach

We show that, for many network coding protocols, it is possible to systematically *transform* the time expanded hypergraph into a circuit that *exactly captures* how the protocol uses the buffers. Given a protocol, a communication schedule, and the corresponding circuit, we prove optimality in three steps. We first show that the circuit indeed simulates the execution of protocol; then apply the results from [2] for memoryless circuits to show that the protocol achieves the min-cut of this circuit with high probability; and finally show that the min-cut of the circuit is the same as the min-cut in G_μ.

To describe our transforms, we note that many network coding protocol proposed so far [4], [14], [16] are composed of two elementary operations: 1) coding packets together by taking a random linear combination of them, and 2) buffering packets. While the coding operation is already naturally captured by the memoryless circuits we show that the storing operation can be simulated by extending a hyperedge (representing a transmission) to all future versions of the recipient(s). Using this observation, we define a hypergraph transformation $(.)_X$ for any given protocol implementation X. This transformation takes a time expanded hypergraph G and transforms it to the hypergraph G_X that exactly captures the execution of protocol X on the communication schedule G. Note that the hypergraph transformation $(.)_X$ does not just depend on the size of the buffer X uses but has to be carefully designed to match the implementation details of protocol X.

B. Protocols and their Transformations

In this section, we describe the transforms for several protocols. We start with the PNC-protocol from Section III-B and then cover two network coding protocols described in [16]: the μ-recombinator and the μ-accumulator protocols. Both protocols are highly efficient variants of PNC, for which any node only stores μ packets in its buffer. Besides reducing the required memory resources, this also improves the computational cost of network coding, because of the reduced amount of information each coding operation is performed over. The two protocols differ in the way the new set of μ packets is obtained after a reception of a new packet (and/or generation of a new packet). The μ-recombinator simply picks μ random packets from the span on the stored packets and



(a) Network transactions over time (b) Time expanded hypergraph (c) Information flow hypergraph (d) PNC Transform

Fig. 1: An example network G with $V = \{a, b, c, d\}$. In Figure 1a, the network communication history is shown in sequence. The link delays are shown on the edges. In Figure 1b, the time expanded hypergraph of the network in Figure 1a is shown assuming that node b and c start with a message at time t_1. In Figure 1c, we show the corresponding natural information flow graph G_∞. In Figure 1d shows the corresponding PNC transform G_{PNC}.

the received packets while the more efficient μ-accumulator randomly combines the incoming packet with each of the μ stored packet individually. The next two definitions present the transformations for the PNC protocol and the μ-recombinator protocol.

Definition V.2 (PNC transform)**.** *The PNC-transform G_{PNC} of a time expanded hypergraph $G = (V, V', A)$ is formed by replacing every hyperedge $e \in A$ by its memory closure $\overline{e}$. Here the memory-closure of a hyperedge $e = (v_t, R_e) = (v_t, \{u^1_{t_1}, u^2_{t_2}, ...u^b_{t_b}\}) \in A$ is defined as $\overline{e} = (v_t, \overline{R_e})$ where $\overline{R_e} = \{u_{t'} \mid \exists u, t : u_t \in R_e$ and $t' \geq t\}$. In other words, we extend every hyperedge e to include all future copies of the recipients.*

Definition V.3 (μ-recombinator transform)**.** *The μ-recombinator transform $G_{\mu\text{-recombinator}}$ of a time expanded hypergraph $G = (V, V', A)$ is formed by adding μ edges from every vertex $v_t \in V'$ to its next copy in time $v_{t'}$ where t' is the smallest $t'' > t$ with $v_{t''} \in V'$.*

Note that the two transforms, G_{PNC} and $G_{\mu-\text{recombinator}}$, have an intuitive structure. Extending a hyperedge in G_{PNC} can be interpreted as changing the storage operation of nodes to requesting/receiving the exact same packet again whenever the "stored" packet is used. For $G_{\mu-\text{recombinator}}$, the μ memory-edges represent that the μ buffered packets are used to generate the next μ random packets to be kept.

Note that, in general, the network transforms are not necessarily as natural and straight-forward as suggested by Definitions V.2 and V.3. One has to be very careful to specify and map all implementation details. Indeed, the transform presented in Definition V.3 does not exactly capture the protocol described in [16] but instead also recombines its stored packets whenever a packet is send. For simplicity, we consider this variant of the recombinator protocol here. As a final example for a slightly more complicated transformation, we pictorially describe the μ-accumulator transform. We consider the implementation described in [14], [16] in which a random multiple of the received packet(s) is added to each stored packet. Its network transform $G_{\mu\text{-accumulator}}$ is formed by first taking the G_{PNC} and then replacing each node according to the template in Figure 2.

C. Simulation and Optimality Proofs

Showing that a protocol implementation and its induced hypergraph transformation match follows by a straight forward

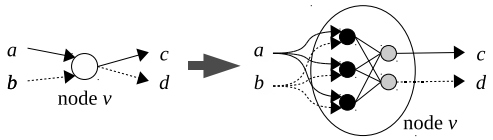


Fig. 2: Template for $(.)_{\mu\text{-accumulator}}$ with $\mu = 3$: The μ black nodes represent the buffer and the gray nodes represent transmissions.

induction. We give such a proof for the PNC protocol; the proofs for the μ-accumulator or μ-recombinator are similar.

Theorem V.4. *Consider a network using the PNC protocol, and let G be the corresponding time-expanded hypergraph with supersource s. Consider the PNC transform G_{PNC} as a circuit as in Section III-A. If the coding vectors for this circuit are selected independently and uniformly from F_q then this simulates the behavior of the PNC protocol. The message associated with each circuit hyperedge $\overline{e} = (v_t, \overline{R_e})$ in G_{PNC} is the message sent by node v at time t. Furthermore, the messages on the incoming hyperedges of v_t in G_{PNC} correspond to the messages buffered by node v at time t.*

Proof: For sake of space only sketch the proof: In order to prove that the circuit G_{PNC} simulates the execution of the PNC protocol, we need to specify carefully how the randomness is used on both sides. For the PNC protocol we assume that a node keeps all received packets (and does not, e.g., keep only innovative packets) and creates any coded packet by drawing random coding coefficient for the packets in the order they were received. We similarly fix the process of choosing the random coding vectors for the circuit to make it match with the PNC protocol.

Now, using an inductive proof over the time (or the topological depth of the nodes in $\overline{G}$), we can show that G_{PNC} simulates the PNC protocol. Firstly, the messages associated with the outgoing hyperedges of the supersource s are by definition the messages generated by the sources. Now consider a node v at time t. We assume, without loss of generality, that no node sends a packet when it has not received or generated a message. Thus, v_t has at least one incoming hyperedge from another node $u_{t'}$ where $t' < t$. By construction of G_{PNC}, the incoming hyperedges to v_t are from all nodes that have sent a packet to v before time t. By induction hypothesis, the incoming hyperedges of v correspond to the messages stored in v in the PNC protocol at time t. Since both the circuit G_{PNC} and the PNC protocol linearly combine packets using

the same random coefficients, the hypothesis holds for the packets created at node v at time t. ∎

Given G_X as a representation of the execution of X on the communication schedule G it is easy to state and proof an equivalent of Lemma V.1: The amount of source information transmitted from s to v at time t via protocol X is at most the (s, v_t)-min-cut in G_X. More interestingly, since G_X is memoryless, we can directly apply the results of [2] to show the converse:

Lemma V.5. *Let G be the time expanded hypergraph for a communication schedule and let G_X be its transform for the network coding protocol X. With probability $1-\epsilon$, the amount of information transmitted from the sources to node v by time t is exactly the min-cut between the supersource s and a node v_t in G_X. Here $\epsilon = O(1/\operatorname{poly}(n))$ is an arbitrarily small inverse polynomial probability given that the coefficient size $\log q$ used in X is $\Theta(\log n)$.*

All that is left to check is that for the protocols presented here this min-cut is indeed the same as the information theoretical optimum as given by G_μ in Lemma V.1:

Lemma V.6. *Let G be any time expanded hypergraph with supersource s. The min-cut between the supersource s and any node v_t is the same in G_∞ and G_{PNC}. Furthermore, the same is true for G_μ, $G_{\mu\text{-recombinator}}$, and $G_{\mu\text{-accumulator}}$.*

Proof: We begin with considering the min-cuts of G_∞ and G_{PNC}. For this we transform any integral flow in G_∞ to a valid flow in G_{PNC} and vice versa. Then, we use the min-cut max-flow theorem. The transformation operates on each path in a flow decomposition separately and repeatedly removes flow from ∞-edges. Consider a flow-carrying unit-capacity hyperedge $(u_t, w_{t'})$ with an ∞-capacity memory-edge $(w_{t'}, w_{t''})$ immediately following it ($t < t' < t''$). We eliminate such ∞-edges one-by-one by rerouting the flow directly through u_t to $w_{t''}$ using the extended hyperedges in G_{PNC}. This process is flow preserving, respects capacities, and eliminates all ∞-edges since every flow path starts with an unit-capacity outgoing hyperedge of s. It can be verified that this transformation is also reversible; thus, gives a bijection between integral (s, v_t)-flows in G_∞ and integral (s, v_t)-flows in G_{PNC}. This finishes the proof for G_{PNC}.

For $G_{\mu\text{-recombinator}}$, one can use the same strategy, and re-route the flow over the μ-capacity memory-edges in G_μ to the μ unit-capacity edges in $G_{\mu\text{-recombinator}}$.

Similarly, for $G_{\mu\text{-accumulator}}$, we first re-route the flow over the μ-capacity memory-edges in G_μ via the extended hyperedges in $G_{\mu\text{-accumulator}}$ created by the PNC transformation. After the PNC transformation, $G_{\mu\text{-accumulator}}$ is formed by replacing each node according to the template in Figure 2. In $G_{\mu\text{-accumulator}}$, we can re-route the flows of each replaced node since each node v_t in G_μ carries at most μ-units of flow. This is true by construction: if a node v is receiving at time t, node v_t has one out-going memory-edge with capacity μ; if a node v is transmitting at time t, then node v_t has only one in-coming memory-edge with capacity μ. ∎

Putting everything together finishes our main theorem:

Theorem V.7. *Assume a network and communication model in which the random coding coefficients are independent from the communication schedule. With high probability, the μ-recombinator, and the μ-accumulator protocols disseminate exactly the maximum amount of information from the sources to every node that* any *protocol using μ memory could have disseminated. The same holds for PNC and algorithms with unlimited memory.*

REFERENCES

[1] S. Li, R. Yeung, and N. Cai, "Linear network coding," *Transactions on Information Theory (TransInf)*, vol. 49, no. 2, pp. 371–381, 2003.

[2] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *Transactions on Information Theory (TransInf)*, vol. 52, no. 10, pp. 4413–4430, 2006.

[3] R. Koetter and M. Médard, "An algebraic approach to network coding," *Transactions on Networking (TON)*, vol. 11, no. 5, pp. 782–795, 2003.

[4] P. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. of the 41st Allerton Conference on Communication Control and Computing*, vol. 41, 2003, pp. 40–49.

[5] D. Lun, M. Médard, R. Koetter, and M. Effros, "On coding for reliable communication over packet networks," *Physical Communication*, vol. 1, no. 1, pp. 3–20, 2008.

[6] ——, "Further results on coding for reliable communication over packet networks," in *Proc. of the International Symposium on Information Theory (ISIT)*, 2005, pp. 1848–1852.

[7] D. Lun, "Efficient operation of coded packet networks," Ph.D. dissertation, Massachusetts Institute of Technology, 2006.

[8] Y. Wu, "A trellis connectivity analysis of random linear network coding with buffering," in *Proc. of the International Symposium on Information Theory (ISIT)*, 2006, pp. 768–772.

[9] S. Deb, M. Médard, and C. Choute, "Algebraic gossip: a network coding approach to optimal multiple rumor mongering," *Transactions on Information Theory (TransInf)*, vol. 52, no. 6, pp. 2486 – 2507, 2006.

[10] D. Mosk-Aoyama and D. Shah, "Information dissemination via network coding," in *Proc. of the International Symposium on Information Theory (ISIT)*, 2006, pp. 1748–1752.

[11] M. Borokhovich, C. Avin, and Z. Lotker, "Tight bounds for algebraic gossip on graphs," in *Proc. of the International Symposium on Information Theory (ISIT)*, 2010, pp. 1758–1762.

[12] B. Haeupler, "Analyzing Network Coding Gossip Made Easy," *Proc. of the 43nd Symposium on Theory of Computing (STOC)*, 2011.

[13] P. Maymounkov, N. Harvey, and D. Lun, "Methods for efficient network coding," in *Proc. of the 44th Allerton Conference on Communication, Control, and Computing*, 2006.

[14] D. S. Lun, P. Pakzad, C. Fragouli, M. Médard, and R. Koetter, "An analysis of finite-memory random linear coding on packet streams," in *Proc. of the International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt)*, 2006, pp. 1–6.

[15] B. Vellambi, N. Torabkhani, and F. Fekri, "Throughput and latency in finite-buffer line networks," *Information Theory, IEEE Transactions on*, vol. 57, no. 6, pp. 3622–3643, 2011.

[16] B. Haeupler and M. Médard, "One Packet Suffices - Highly Efficient Packetized Network Coding With Finite Memory," in *Proc. of the International Symposium on Information Theory (ISIT)*, 2011.

[17] R. Yeung, "Avalanche: A Network Coding Analysis," *Communications in Information & Systems*, vol. 7, no. 4, pp. 353–358, 2007.

[18] ——, *Information theory and network coding*. Springer Verlag, 2008.

[19] J. Sundararajan, D. Shah, and M. Médard, "On queueing in coded networks-queue size follows degrees of freedom," in *IEEE Information Theory Workshop on Information Theory for Wireless Networks (ITW)*, 2007, pp. 1–6.

[20] K. Sundararajan, D. Shah, and M. Médard, "ARQ for network coding," in *Proc. of the IEEE International Symposium on Information Theory (ISIT)*, 2008, pp. 1651–1655.

[21] S. Bhadra and S. Shakkottai, "Looking at Large Networks: Coding vs. Queueing," in *Proc. of the 25th IEEE International Conference on Computer Communications (INFOCOM)*, 2007, pp. 1–12.