An Analysis of Finite-Memory Random Linear Coding on Packet Streams

Desmond S. Lun*, Payam Pakzad†, Christina Fragouli†, Muriel Médard*, and Ralf Koetter‡
*Laboratory for Information and Decision Systems
Massachusetts Institute of Technology, Cambridge, MA 02139, USA
E-mail: {dslun, medard}@mit.edu
†Laboratoire d'algorithmique
Ecole Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland
E-mail: {payam.pakzad, christina.fragouli}@epfl.ch
‡Coordinated Science Laboratory
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
E-mail: koetter@uiuc.edu

***Abstract*— We consider the following packet coding scheme: The coding node has a fixed, finite memory in which it stores packets formed from an incoming packet stream, and it sends packets formed from random linear combinations of its memory contents. We analyze the scheme in two settings: as a self-contained component in a network providing reliability on a single link, and as a component employed at intermediate nodes in a block-coded end-to-end connection. We believe that the scheme is a good alternative to automatic repeat request when feedback is too slow, too unreliable, or too difficult to implement.**

I. Introduction

The recent advent of linear-complexity erasure-correcting codes in [1], [2] has made the use of coding, often called forward error correction (FEC), much more attractive as a means of providing reliable packet transmission. Indeed, when the physical medium makes establishing good feedback links difficult—as is often the case in wireless and satellite networks—or when the application is simply very demanding—as is often the case in real-time applications—using such codes is the clear strategy of choice over using automatic repeat request (ARQ).

The schemes in [1], [2] are suitable for end-to-end coding and thus only achieve the end-to-end capacity. More recently, it was shown that the min-cut capacity can in fact be achieved if we allow intermediate nodes to send out random linear combinations of all previously received packets [3], [4], [5], [6]. The rate benefits come, however, at the cost of memory usage: intermediate nodes need the capability to store all received packets that originate from the same coding block at the source and, though some compression is possible, the memory usage of intermediate nodes nevertheless grows with the size of the coding block.

This work was supported by the National Science Foundation under grant nos. CCR-0093349, CCR-0325673, CCR-0325951, and CNS-0434974; by the Office of Naval Research under grant no. N00014-05-1-0197; by the Air Force Office of Scientific Research under grant no. F49620-01-1-0365; and by Fonds National Suisse under award no. 200021-103836/1.

In this paper, we consider fixing the memory of intermediate nodes at a constant size. We establish the trade-off between memory usage and achievable rate. We show that, by varying the memory m at intermediate nodes, we can achieve all points of the gap between the min-cut and the end-to-end capacity. Interestingly, we can achieve a significant percentage of this gap for a small value of m. We thus argue that, in a practical system, where intermediate nodes have restricted resources to be possibly shared by several connections, a coding scheme that utilizes a small amount of memory can still offer significant rate benefits.

The proposed encoder functions much like a convolutional encoder at each intermediates node: it employs a fixed memory to accommodate an arriving stream of packets of indeterminate length. There is, however, one very important difference between the coding we consider and convolutional coding on packets [7]: Our coding scheme is rateless in the sense that the coded packet stream can maintain any ratio that we desire to the rate of the message packet stream and can be freely adjusted in real-time. By contrast, packet-level convolutional coding can adapt its rate only by puncturing the code, which allows variation only within a limited range. This ability to freely adapt the rate of the code allows the scheme to vary its erasure-correcting capability to meet time-varying conditions, which are common in packet networks.

We consider the use of our coding scheme in two settings: first, as a self-contained component in a network providing reliability on a single link; and, second, as a component employed at intermediate nodes in a block-coded end-to-end connection. We shall see that, by explicitly recognizing that the scheme is being employed as a component at intermediate nodes in a block-coded end-to-end connection, better performance can be achieved than if we did not.

We begin by describing the coding scheme in the following section. We consider its use as a self-contained component in Section III and as a component in a block-coded end-to-end connection in Section IV.

II. Code design

The design of the coding scheme is simple. We assume that packets are vectors of length L over the finite field $\mathbb{F}_q$. Hence, if the packet length is b bits, then $L = \lceil b/\log_2 q \rceil$. The encoder has a memory capable of storing m packets, and it uses this memory in one of two ways:

1) as a shift register: arriving packets are stored in memory and, if the memory is already full, the oldest packet in the memory is discarded; or
2) as an accumulator: arriving packets are multiplied by a random vector chosen uniformly over $\mathbb{F}_q^m$, and the product is added to the m memory slots.

To form coded packets, the encoder simply takes a random linear combination (in $\mathbb{F}_q$) of its memory contents, with the vector of coefficients of the combination drawn uniformly from $\mathbb{F}_q^m \setminus \{0\}$. This can be done as often or as seldom as we wish; hence the ratelessness of the code—the output of coded packets does not need to be synchronized to the arrival of packets in any way and, in particular, does not need to be related to it according to some fixed coding rate.

The coding scheme we propose results essentially from taking the coding scheme described in [3], [4], [5] for a single intermediate node and limiting the encoder's memory to a size that is fixed with respect to the input. Related random linear coding schemes are described in [8], [9] for the application of multicast over lossless networks, in [10] for data dissemination, and in [11] for data storage.

III. Use as a self-contained component

When used as a self-contained component, the encoder takes an incoming stream of message packets, $u_1, u_2, \ldots$, and forms a coded stream of packets that is placed on its lossy outgoing link and decoded on reception. The decoder, we assume, knows the linear transformations that the packets it receives are of the message packets. This information can be communicated to the decoder by a variety of means, which include placing it into the header of each packet (which is certainly viable when the memory is used as a shift register—the overhead is $m \log_2 q$ bits plus that of a sequence number), and initializing the random number generators at the encoder and decoder with the same seed.

The task of decoding, then, equates to matrix inversion in $\mathbb{F}_q$, which can be done straightforwardly by applying Gaussian elimination to each packet as it is received. This procedure produces an approximately-steady stream of decoded packets with an expected delay that is constant in the length of the input stream. Moreover, if the memory is used as a shift register, then the complexity of this decoding procedure is also constant with the length of the input stream and, on average, is $O(m^2)$ per packet.

A. Model

We discretize the time axis into epochs that correspond to the transmission of an outgoing packet. Thus, in each epoch, an outgoing packet is transmitted, which may be lost, and one or more incoming packets are received. If transmission is to be reliable, then the average number of incoming packets received in each epoch must be at most one.

We adopt the following simple model of incoming packet arrivals and outgoing packet losses, with the understanding that generalizations are certainly possible. We assume that, in an epoch, a single packet arrives independently with probability r and no packets arrive otherwise, and the transmitted outgoing packet is lost independently with probability ε and is received otherwise. This model is appropriate when losses and arrivals are steady—and not bursty.

B. Analysis

The following analysis is in the limit of $q \to \infty$, i.e. the limit of infinite field size. We later discuss how the analysis may be adapted for finite q, and, in Section III-C, we quantify by simulation the difference between the performance in the case of finite q and that of infinite q in some particular instances.

We begin by considering the difference between the number of packets received by the encoder and the number of packets transmitted and not lost. This quantity, we see, evolves according to the infinite-state Markov chain shown in Figure 1, where $\alpha = r\varepsilon$, $\beta = (1-r)(1-\varepsilon)$, and $\gamma = r(1-\varepsilon)+(1-r)\varepsilon$.

At epoch 0, the memory of the encoder is empty and we are in state 0. We continue to remain in state 0 in subsequent epochs at least until the first packet u_1 arrives. Suppose the next outgoing packet is not lost. Then we remain in state 0, and the decoder receives a packet that is a random linear combination of u_1, i.e. a random scalar multiple of u_1, and, since q is infinitely large by assumption, this scalar multiple is non-zero with probability 1; so the decoder can recover u_1 from the packet that it receives.

Now suppose the next outgoing packet is lost; so we move to state 1. If an outgoing packet is transmitted and not lost before the next packet arrives, then we again receive a random scalar multiple of u_1 and return to state 0. So suppose u_2 arrives. Then, the next outgoing packet is a random linear combination of u_1 and u_2. Suppose further that this packet is received by the decoder, so we are again in state 1. This packet, currently, is more or less useless to the decoder; it represents a mixture between u_1 and u_2 and does not allow us to determine either. Nevertheless, it gives the decoder some information that it did not previously know, namely, that u_1 and u_2 lie in a particular linear subspace of $\mathbb{F}_q^2$. Any subsequent packet received by the decoder, then, also gives the decoder information that it did not previously know, provided that the linear combination that it is formed from is linearly independent of the one already received. We call such an informative packet *innovative.*

What we see is that, provided that packets arrive only in states $0, 1, \ldots, m-1$, then every packet that is transmitted from a non-zero state is innovative at the decoder if it is not lost, and, at every return to state 0, the decoder is able to recover one or more packets. If a packet arrives in state m, then losses start to occur. Information in the encoder's memory is overwritten or corrupted, and will never be recovered. The current contents of the encoder's memory, however, can still be recovered and, from the point of view of recovering these

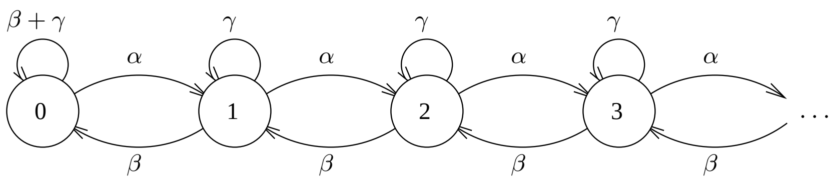

Fig. 1. Markov chain modeling the evolution of the difference between the number of packets received by the encoder and the number of packets transmitted and not lost.

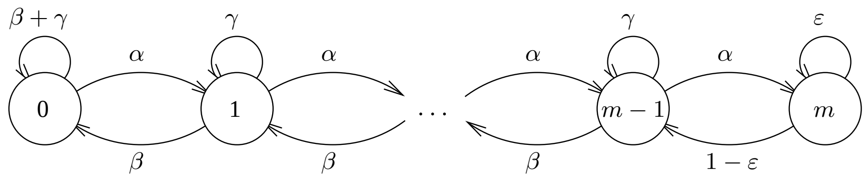

Fig. 2. Markov chain modeling the behavior of the coding scheme in the limit of $q \to \infty$.

contents, the coding system behaves as though we were in state m. Hence, to analyze the performance of the coding scheme, we modify the Markov chain shown in Figure 1 to that in Figure 2. Let x_t be the state of this Markov chain at time t. We can interpret x_t as the number of innovative packets the encoder has to send at time t.

We now proceed to derive some quantities that are useful for designing the parameters of the coding scheme. We begin with the steady-state probabilities $\pi_i := \lim_{t\to\infty} \Pr(x_t = i)$. Since $\{x_t\}$ is a birth-death process, its steady-state probabilities are readily obtained. We obtain

$$\pi_i = \frac{\rho^i(1-\rho)}{1-\sigma\rho^m} \qquad (1)$$

for $i = 0, 1, \dots, m-1$, and

$$\pi_m = \frac{\varepsilon\sigma\rho^{m-1}(1-\rho)}{1-\sigma\rho^m}, \qquad (2)$$

where $\rho := \alpha/\beta = r\varepsilon/(1-r)(1-\varepsilon)$ and $\sigma := r/(1-\varepsilon)$. We assume $\rho < 1$, which is equivalent to $r < 1-\varepsilon$, for, if not, the capacity of the outgoing link is exceeded, and we cannot hope for the coding scheme to be effective.

We now derive the probability of packet loss, p_l. Evaluating p_l is not straightforward because, since coded packets depend on each other, the loss of a packet owing to the encoder exceeding its memory is usually accompanied by other packet losses. We derive an upper bound on the probability of loss.

A packet is successfully recovered by the decoder if the ensuing path taken in the Markov chain in Figure 2 returns to state 0 without a packet arrival occurring in state m. Let q_i be the probability that a path, originating in state i, reaches state 0 without a packet arrival occurring in state m. Our problem is very similar to a random walk, or ruin, problem (see, for example, [12, Chapter XIV]). We obtain

$$q_i = \frac{1-\sigma\rho^{m-i}}{1-\sigma\rho^m}$$

for $i = 0, 1, \dots, m$.

Now, after the coding scheme has been running for some time, a random arriving packet finds the scheme in state i with probability π_i and, with probability $1-\varepsilon$, the scheme returns to state i after the next packet transmission or, with probability ε, it moves to state $i+1$. Hence

$$\begin{aligned} 1 - p_l &\geq \sum_{i=0}^{m-1} \{(1-\varepsilon)q_i + \varepsilon q_{i+1}\}\pi_i \\ &= \frac{1}{(1-\sigma\rho^m)^2}\{1 - \rho^m - (1-2\varepsilon)m\sigma\rho^m \\ &\qquad - \varepsilon m\sigma\rho^{m-1} + (1-\varepsilon)m\sigma\rho^{m+1}\}. \end{aligned}$$

From which we obtain

$$\begin{aligned} p_l \leq \frac{\rho^{m-1}}{(1-\sigma\rho^m)^2}\{&\varepsilon m\sigma + (1-2\sigma+m\sigma-2\varepsilon m\sigma)\rho \\ &- (1-\varepsilon)m\sigma\rho^2 + \sigma^2\rho^{m+1}\}. \end{aligned} \qquad (3)$$

We have thus far looked at the limit of $q \to \infty$, while, in reality, q must be finite. There are two effects of having finite q: The first is that, while the encoder may have innovative information to send to the decoder (i.e. $x_t > 0$), it fails to do so because the linear combination it chooses is not linearly independent of the combinations already received by the decoder. For analysis, we can consider such non-innovative packets to be equivalent to erasures, and we find that the effective erasure rate is $\varepsilon(1-q^{-x_t})$. The Markov chain in Figure 2 can certainly be modified to account for this effective erasure rate, but doing so makes analysis much more tedious.

The second of the effects is that, when a new packet arrives, it may not increase the level of innovation at the encoder. When the memory is used as a shift register, this event arises because a packet is overwritten before it has participated as a linear factor in any successfully received packets, i.e. all successfully received packets have had a coefficient of zero for that packet. When the memory is used as an accumulator, this event arises because the random vector chosen to multiply the new packet results in the dimension of the encoder's memory

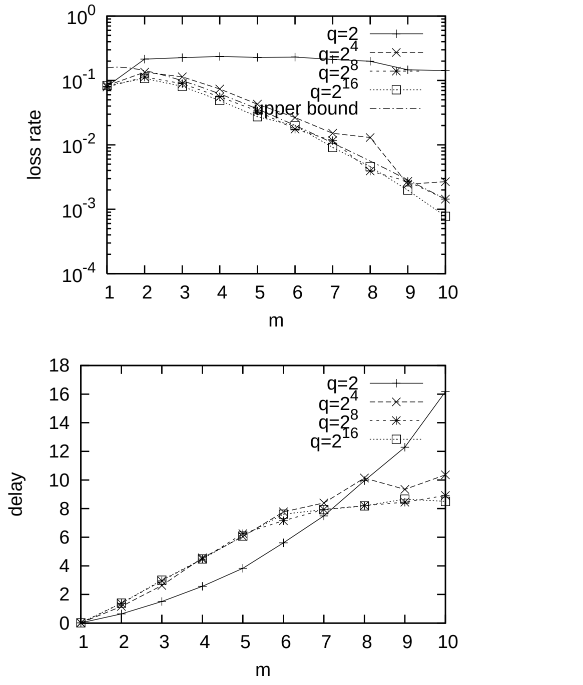
Fig. 3. Average loss (top) and delay (bottom) for 200,000 packets as a function of memory size m with $r = 0.8$, $\varepsilon = 0.1$, and various coding field sizes q. The upper bound on the probability of loss for $q \to \infty$ is also drawn.

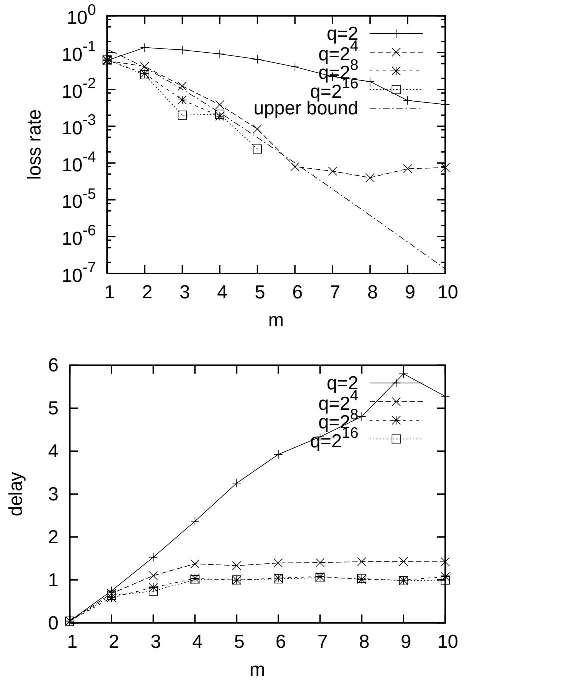
Fig. 4. Average loss (top) and delay (bottom) for 200,000 packets as a function of memory size m with $r = 0.6$, $\varepsilon = 0.1$, and various coding field sizes q. The upper bound on the probability of loss for $q \to \infty$ is also drawn. Average losses of zero are not shown.

not being increased. The event of the level of innovation not being increased by a new packet can be quite disastrous, because it is effectively equivalent to the encoder exceeding its memory. Fortunately, the event seems rare; in the accumulator case, we can quantify the probability of the event exactly as $1 - q^{x_t - m}$.

C. Simulation results

We chose $\varepsilon = 0.1$ and simulated the performance of our coding scheme for 200,000 packets with various choices of the parameters r, q, and m (see Figures 3 and 4). We decoded using Gaussian elimination on packets as they were received and used the encoder's memory as a shift register to keep decoding complexity constant with the length of the packet stream. Delay was evaluated as the number of epochs between a packet's arrival at the encoder and it being decoded, neglecting transmission delay. We see that a field size $q \geq 2^8$ (perhaps even $q \geq 2^4$) is adequate for attaining loss rates close to the upper bound for infinite field size.

IV. Use in a block-coded end-to-end connection

When our coding scheme is used as a self-contained component, packets are sometimes lost because the decoder receives linear combinations that, although innovative, are not decodable. For example, suppose the decoder receives $u_1 + u_2$, but is neither able to recover u_1 nor u_2 from other packets. This packet, u_1+u_2, definitely gives the decoder some information, but, without either u_1 or u_2, the packet must be discarded. This would not be the case, however, if u_1 and u_2 were themselves coded packets—a trivial example, assuming that we are not coding over $\mathbb{F}_2$, is if $u_1 = u_2 = w_1$, where w_1 is a message packet for an outer code.

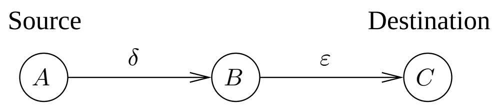

Fig. 5. A two-link tandem network.

In such a case, we can use the encoder described in Section II, but we should not attempt to recover the stream fed to the encoder with the procedure detailed in Section III, or rather, if we do, then the decoder should preserve any packets that are linearly independent after Gaussian elimination, not only those that are copies of packets in the encoder's input stream.

In this section, we consider using our coding scheme in the simplest network with end-to-end coding and intermediate node coding: a two-link tandem network with coding at the intermediate node (see Figure 5). This simple two-link tandem network that we analyze serves as a basis for longer tandem networks and more general network topologies.

The source A has K message packets $w_1, w_2, \ldots, w_K$ that it encodes with an erasure code into N packets $u_1, u_2, \ldots, u_N$. These packets are sent over link AB with loss rate δ to the intermediate node B. Node B uses our coding scheme to encode the packets that it receives, and sends coded packets over link BC with loss rate ε to the destination C. Node C, now, cares only about receiving K packets that are linearly-independent transformations of $u_1, u_2, \ldots, u_N$ because, from these, it has an invertible linear transformation of the original message packets $w_1, w_2, \ldots, w_K$. Recovering $w_1, w_2, \ldots, w_K$ at C is done by Gaussian elimination.

A. Model

We again adopt a discrete-time model. This model, which is identical to that used in [6], supposes that packets are transmitted on both links AB and BC at each epoch and that packets transmitted on AB and BC are lost independently with probability δ and ε, respectively. Although actual networks may not have transmissions that are synchronized in this way, the synchronicity assumption may be relaxed to an extent by accounting for differences in the packet injection rates on links AB and BC using the loss rates δ and ε.

B. Analysis

We again conduct our analysis in the limit of infinite field size. The considerations for finite field size are the same as those mentioned in Section III-B.

Let x_t denote the number of innovative packets (relative to $u_1, u_2, \ldots, u_N$) node B has to send at time t, and let y_t denote the number of innovative packets received by node C at time t. By the arguments of Section III-B, the following principles govern the evolution of x_t and y_t over time:

- As long as $x_t < m$, i.e. the memory is not fully innovative, node B increases the innovation contents of its memory by 1 upon successful reception of a packet over link AB.
- As long as $x_t > 0$, i.e. the memory is not completely redundant, the output of B is innovative, so y_t will increase by 1 provided that transmission over BC is successful.

Let $\alpha := (1-\delta)\varepsilon$, $\beta := \delta(1-\varepsilon)$, and $\lambda := (1-\delta)(1-\varepsilon)$. Then the evolution of x_t and y_t is modeled by the Markov chain shown in Figure 6, where the horizontal coordinate of a state indicates x_t, and the vertical coordinate corresponds to the variable y_t.

We see that $\{x_t\}$ evolves as in Section III-B, so its steady-state probabilities are given by (1) and (2) with $r = 1-\delta$. Hence, once the system is sufficiently mixed, the probability that y_t increases at time t is given by

$$\begin{aligned}\lambda\pi_0 + (1-\varepsilon)\pi_1 + \cdots + (1-\varepsilon)\pi_m &= (1-\varepsilon)(1-\delta\pi_0) \\ &= (1-\delta)(1-\pi_m).\end{aligned}$$

Therefore the system can operate at rate

$$R = (1-\delta)(1-\pi_m) \tag{4}$$

with high probability of success.

Suppose, without loss of generality, that $\delta > \varepsilon$, so $\rho < 1$. Let R^* be the min-cut capacity, or maximum rate, of the system, which, in this case, is $1-\delta$. Then the relative rate loss with respect to the min-cut rate is

$$1 - \frac{R}{R^*} = \pi_m. \tag{5}$$

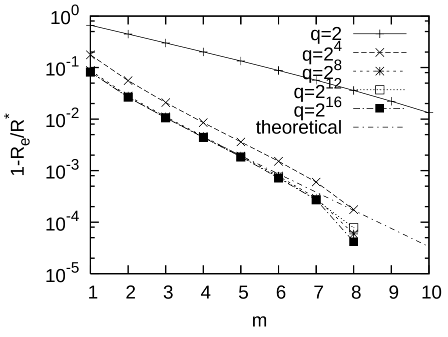

Fig. 7. Relative rate loss with respect to min-cut rate as a function of memory size m for $\delta = 0.2$, $\varepsilon = 0.1$, and various coding field sizes q.

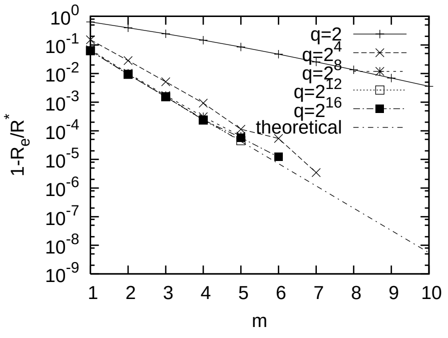

Fig. 8. Relative rate loss with respect to min-cut rate as a function of memory size m for $\delta = 0.4$, $\varepsilon = 0.1$, and various coding field sizes q.

C. Simulation results

As discussed before, the analysis of Section IV assumes forming linear combinations over an infinitely large field, resulting in a Markov chain model with transition probabilities given in Figure 6. If on the other hand the field size is finite, we can still find new expressions for the transition probabilities, although the complete analysis becomes very complex. In particular, assume that the memory is used as an accumulator, so that the contents of the memory at each time are uniformly random linear combinations, over $\mathbb{F}_q$, of the received packets at B by that time. Then, as we have mentioned, if the innovation content of the memory is x and a new packet arrives at B, the probability that B can increase the innovation of its memory by 1 is $(1 - q^{x-m})$, independently from all other past events. Similarly, the probability that the output of B is innovative is $(1 - q^{-x})$.

To quantify the effect of operations over a finite field, we simulated the evolution of this Markov chain for two combinations of δ and ϵ values that were also considered in Section III (see Figures 7 and 8). The effective rate is considered to be $R_e := y_N/N$, where N is the number of packet transmissions at A, and as before, y_N is the number of innovative packets received at C by time N. We simulated this process for $N = 10^9$ packets. For different field sizes, we plot the relative rate loss with respect to the min-cut rate—i.e. $1 - R_e/R^*$—as a function of the memory size. Also plotted

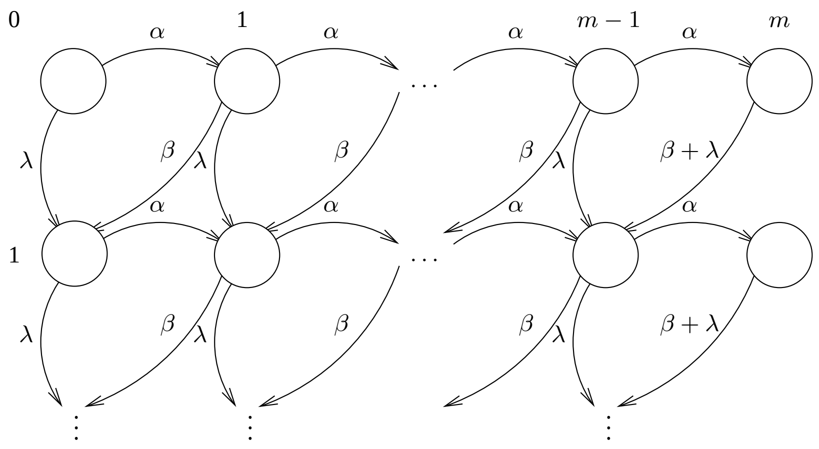

Fig. 6. Markov chain modeling the evolution of x_t and y_t. To simplify the diagram, we do not show self-transitions.

is the theoretical result from (5).

By comparing Figures 7 and 8 with Figures 3 and 4, respectively, we see the advantage that comes from explicitly recognizing that the coding scheme is being employed at the intermediate node of a block-coded end-to-end connection. The loss rate in the latter plots essentially equates to the factor $1 - R_e/R^*$ in the former. Thus, in the limit of infinite q, we compare the probability of loss p_l upper bounded by equation (3) and the expression for $1 - R/R^*$ given by equation (5). We note that, in both cases, the decay as $m \to \infty$ is as ρ^m. Moreover, it follows from our discussion that $1 - R/R^*$ must be a lower bound for p_l, hence p_l itself decays as ρ^m as $m \to \infty$.

V. Conclusion and discussion

We analyzed the performance of finite-memory random linear coding and saw that it shows promise as a means of providing reliability in packet networks. This task is performed in many current packet networks using ARQ, which relies on feedback. But feedback is sometimes either not available or not of high enough quality; and, in this case, feedforward coding schemes such as ours become useful. We therefore believe that our coding scheme is a good alternative to ARQ when feedback is too slow, too unreliable, or too difficult to implement. (Indeed, the Markov chain in Figure 2 also describes the behavior of stop-and-wait ARQ with delay-free feedback.)

If some feedback is available, however, it can be used to assist our coding scheme. For example, feedback messages can be sent from the decoder when the state of the coding scheme approaches m, which the encoder should respond to either by decreasing the code rate, by increasing m, or both. This strategy requires some level of feedback that is less than that required by an ARQ scheme and, indeed, by varying the threshold state above which feedback messages are sent, it is possible to obtain a full trade off of coding with feedback.

Looking more broadly, we see from this and other work that random linear coding, as a general technique, is a promising way of providing reliability and enabling multicast in packet networks and that, while various implementation issues have been discussed, no clear picture guiding the path toward a protocol has yet been offered. This picture, we believe, should include the finite-memory considerations of this paper as well as the use of feedback. It may even allows us to view ARQ as a special case.

Acknowledgments

The authors would like to thank Amin Shokrollahi for helpful discussions and suggestions.

References

[1] P. Maymounkov, “Online codes,” NYU, Technical Report TR2002-833, Nov. 2002.
[2] A. Shokrollahi, “Raptor codes,” Jan. 2004, preprint.
[3] D. S. Lun, M. Médard, R. Koetter, and M. Effros, “On coding for reliable communication over packet networks,” submitted to *IEEE Trans. Inform. Theory*.
[4] ——, “Further results on coding for reliable communication over packet networks,” in *Proc. 2005 IEEE International Symposium on Information Theory (ISIT 2005)*, Sept. 2005, pp. 1848–1852.
[5] D. S. Lun, M. Médard, and M. Effros, “On coding for reliable communication over packet networks,” in *Proc. 42nd Annual Allerton Conference on Communication, Control, and Computing*, Sept.–Oct. 2004, invited paper.
[6] P. Pakzad, C. Fragouli, and A. Shokrollahi, “Coding schemes for line networks,” in *Proc. 2005 IEEE International Symposium on Information Theory (ISIT 2005)*, Sept. 2005.
[7] M. Arai, S. Fukumoto, and K. Iwasaki, “Reliability analysis of a convolutional-code-based packet level FEC under limited buffer size,” *IEICE Trans. Fundamentals*, vol. E88-A, no. 4, pp. 1047–1054, Apr. 2005.
[8] P. A. Chou, Y. Wu, and K. Jain, “Practical network coding,” in *Proc. 41st Annual Allerton Conference on Communication, Control, and Computing*, Oct. 2003.
[9] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, “Toward a random operation of networks,” submitted to *IEEE Trans. Inform. Theory*.
[10] S. Deb and M. Médard, “Algebraic gossip: A network coding approach to optimal multiple rumor mongering,” submitted to *IEEE Trans. Inform. Theory*.
[11] S. Acedański, S. Deb, M. Médard, and R. Koetter, “How good is random linear coding based distributed networked storage?” in *Proc. WINMEE, RAWNET and NETCOD 2005 Workshops*, Apr. 2005.
[12] W. Feller, *An Introduction to Probability Theory and its Applications*, 3rd ed. New York, NY: John Wiley & Sons, 1968, vol. 1.