Practical Network Coding

Philip A. Chou*, Yunnan Wu†, and Kamal Jain*

*Microsoft Corporation, One Microsoft Way, Redmond, WA 98052-6399 USA
†Dept. of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA
pachou@microsoft.com, yunnanwu@ee.princeton.edu, kamalj@microsoft.com

Abstract

We propose a distributed scheme for practical network coding that obviates the need for centralized knowledge of the graph topology, the encoding functions, and the decoding functions, and furthermore obviates the need for information to be communicated synchronously through the network. The result is a practical system for network coding that is robust to random packet loss and delay as well as robust to any changes in the network topology or capacity due to joins, leaves, node or link failures, congestion, and so on. We simulate such a practical network coding system using the network topologies of several commercial Internet Service Providers, and demonstrate that it can achieve close to the theoretically optimal performance.

1 Introduction

In their pioneering theoretical work on network coding, in which the network is modeled by a directed graph (V, E) with edge capacities, Alswede et al. [1] showed that a sender $s \in V$ can communicate common information to a set of receivers $T \subseteq V$ at a rate achieving the broadcast capacity h (the value of the minimum cut between s and any $t \in T$) provided one allows network coding, i.e., encoding at the interior nodes of the network. Conversely, it is generally not possible to achieve this communication rate if one allows only routing or copying messages at the interior nodes of the network. Shortly afterwards, Li, Yeung, and Cai [2] showed that it is sufficient for the encoding functions at the interior nodes to be linear. Koetter and Médard [3] showed how to find the coefficients of the linear encoding and decoding functions by finding values for the indeterminates of a polynomial for which the polynomial is non-zero. They also showed that such values can always be found in a field of size $h|T|$, where $|T|$ is the number of receivers. Jaggi, Sanders, et al. [4, 5, 6] showed for acyclic networks how to find the encoding and decoding coefficients in polynomial time, and showed (as did [7]) that field size $|T|$ suffices. They also showed that the linear encoding functions can be designed randomly, and that if the field size is at least $|E|/\delta$, the encoding will be invertible at any given receiver with probability at least $1 - \delta$, while if the field size is at least $|E||T|/\delta$, then the encoding will be invertible simultaneously at all receivers with probability at least $1 - \delta$. Other researchers, e.g., Ho et al. [8], provided a very similar result for random coding.

Network coding is presumably highly applicable to communication through real networks, the primary example being the Internet, both at the IP layer (e.g., in routers) and at the application layer (e.g., in peer-to-peer networks, content distribution networks, and

other overlay networks). Other examples include ATM networks, ad hoc wireless radio networks, and so forth, all of which are packet networks. However, there are significant gaps between the previous theoretical work on network coding and the practical network coding needed for communication through real networks.

Previous theoretical work in network coding has often assumed that symbols flow synchronously throughout the entire network, and (to facilitate this model) that edges have integer or unit capacities. In real networks, however, information travels asynchronously in packets, packets are subject to random delays and losses on every edge, and edges have essentially unknown capacities, which vary as competing communication processes begin and end. Previous theoretical work has also assumed at least some centralized knowledge of the network topology for the purposes of computing the broadcast capacity h and/or computing the coding functions. In real networks, however, it may be difficult either to obtain centralized knowledge, or to arrange reliable broadcast of that knowledge to the nodes across the very communication network that is being established. Previous theoretical work has given some consideration to designing encoding functions for a class of non-ergodic failure patterns not reducing the capacity below a certain amount [3, 6]. However, in these works the decoders still need to know the failure pattern in order to compute and apply the proper linear decoding function. Unfortunately, communicating the failure pattern to the decoders needs to be done reliably, which is again problematic. In previous theoretical work, graphs with cycles have generally presented difficulties, with results holding only in the limit of large delay, for example. However, in real networks, cycles abound; indeed most edges are bi-directional. Finally, previous theoretical work has generally ignored the problem of heterogeneous receivers, targeting the sending rate to the capacity of the worst-case receiver. In real networks, the worst-case receiver may not be known. Moreover, if an important link to a receiver fails, bringing its throughput below the nominal broadcast capacity, the other receivers should not experience the same worst-case throughput.

Our work on practical network coding addresses real packet networks, where information is delivered in packets subject to random delays and losses, where edges have variable capacities due to congestion or other cross traffic, where node and link failures as well as additions and deletions are common (e.g., in peer-to-peer or ad hoc networks), where cycles are everywhere, where the actual broadcast capacity is unknown, and where receivers have heterogeneous capacities. We require no centralized knowledge of the graph topology or the encoding or decoding functions, and we use simple techniques that are applicable in practice.

2 Packet Format

In this section, we propose a packet format that removes the need for any centralized knowledge of the graph topology or the encoding or decoding functions. This is the foundation of our practical network coding scheme.

We start our discussion in the standard framework: with an acyclic graph (V, E) having unit capacity edges, a sender $s \in V$, and a set of receivers $T \subseteq E$. The broadcast capacity h is the minimum number of edges in any cut between the sender and a receiver. Each edge $e \in E$ emanating from a node $v = in(e)$ carries a symbol $y(e)$ that is a linear combination of the symbols $y(e')$ on the edges e' entering v, namely, $y(e) = \sum_{e':out(e')=v} m_e(e')y(e')$. The *local encoding vector* $\mathbf{m}(e) = [m_e(e')]_{e':out(e')=v}$ represents the encoding function at node v along edge e. If v is the sender s, then to

maintain uniformity of notation we introduce artificial edges $e'_1, \ldots, e'_h$ entering s, carrying the h source symbols $y(e'_i) = x_i$, $i = 1, \ldots, h$. Thus by induction $y(e)$ on any edge $e \in E$ is a linear combination $y(e) = \sum_{i=1}^{h} g_i(e) x_i$ of the source symbols, where the h-dimensional vector of coefficients $\mathbf{g}(e) = [g_1(e), \ldots, g_h(e)]$ can be determined recursively by $\mathbf{g}(e) = \sum_{e':out(e')=v} m_e(e')\mathbf{g}(e')$, where $\mathbf{g}(e'_i)$ on the artificial edge e'_i is initialized to the ith unit vector. The vector $\mathbf{g}(e)$ is known as the *global encoding vector* along edge e. Any receiver t receiving along its h (or more) incoming edges $e_1, \ldots, e_h$ the symbols

$$\begin{bmatrix} y(e_1) \\ \vdots \\ y(e_h) \end{bmatrix} = \begin{bmatrix} g_1(e_1) & \cdots & g_h(e_1) \\ \vdots & \ddots & \vdots \\ g_1(e_h) & \cdots & g_h(e_h) \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_h \end{bmatrix} = G_t \begin{bmatrix} x_1 \\ \vdots \\ x_h \end{bmatrix}$$

can recover the source symbols $x_1, \ldots, x_h$ as long as the matrix G_t of global encoding vectors $\mathbf{g}(e_1), \ldots, \mathbf{g}(e_h)$ has rank h. This will be true with high probability if the local encoding vectors are generated randomly and the symbols lie in a finite field of sufficient size. According to [6], if the field size is 2^{16} and the number of edges in the network is at most $|E| = 2^8$, then the matrix G_t at any given receiver will have full rank with probability at least $1 - 2^{-8} = 0.996$. We show in Section 4 that a field size of 2^8 is usually sufficient in practice, and 2^{16} is more than sufficient, as any loss due to field size becomes negligible compared to other losses typical at any given receiver.

In a packet network, the symbols $y(e)$ carried along an edge e can be grouped into packets. In the Internet, a typical maximum packet size excluding headers is somewhat larger than 1400 bytes. Thus each IP packet can carry about $N = 1400$ symbols if the field size is 2^8 or about $N = 700$ symbols if the field size is 2^{16}. Thus we packetize the symbols $y(e)$ flowing on each edge e into vectors $\mathbf{y}(e) = [y_1(e), y_2(e), \ldots, y_N(e)]$ of the appropriate length (depending on the field size), and now each of these *vectors* can be expressed as a linear combination $\mathbf{y}(e) = \sum_{e':out(e')=v} m_e(e')\mathbf{y}(e')$ of the *vectors* $\mathbf{y}(e')$ on the edges e' entering $v = in(e)$. Likewise, we packetize the source symbols x_i flowing into the sender on the artificial edges e'_i into vectors $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,N}]$, so that any receiver can recover (with high probability) the h source vectors $\mathbf{x}_1, \ldots, \mathbf{x}_h$ from any h received packets,

$$\begin{bmatrix} \mathbf{y}(e_1) \\ \vdots \\ \mathbf{y}(e_h) \end{bmatrix} = \begin{bmatrix} y_1(e_1) & y_2(e_1) & \cdots & y_N(e_1) \\ \vdots & \vdots & \ddots & \vdots \\ y_1(e_h) & y_2(e_h) & \cdots & y_N(e_h) \end{bmatrix} = G_t \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_h \end{bmatrix} = G_t \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{h,1} & x_{h,2} & \cdots & x_{h,N} \end{bmatrix}.$$

We now come to the foundational idea of the paper: we include *within each packet* flowing on edge e the h-dimensional global encoding vector $\mathbf{g}(e)$. In this way, the global encoding vectors needed to invert the code at any receiver can be found in the arriving packets themselves. This scheme can be simply accomplished by prepending the ith unit vector to the ith source vector $\mathbf{x}_i$, $i = 1, \ldots, h$, and processing the vectors at each node as usual. Any receiver can then recover the source vectors $\mathbf{x}_1, \ldots, \mathbf{x}_h$ using Gaussian elimination on the vectors in its h (or more) received packets,

$$\begin{bmatrix} g_1(e_1) & \cdots & g_h(e_1) & y_1(e_1) & y_2(e_1) & \cdots & y_N(e_1) \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1(e_h) & \cdots & g_h(e_h) & y_1(e_h) & y_2(e_h) & \cdots & y_N(e_h) \end{bmatrix} = G_t \begin{bmatrix} 1 & & 0 & x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\ & \ddots & & \vdots & \vdots & \ddots & \vdots \\ 0 & & 1 & x_{h,1} & x_{h,2} & \cdots & x_{h,N} \end{bmatrix}.$$

The cost of this scheme is the overhead of transmitting h extra symbols in each packet. But this is reasonable: if h is 50, and the field size is 2^8, then the overhead

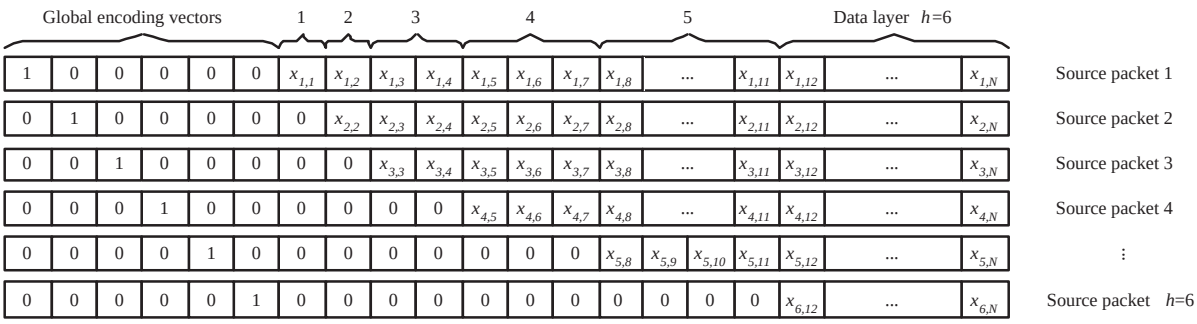

Figure 1: Source vector partitioning and redundancy for Priority Encoding Transmission.

is approximately $50/1400 \approx 3\%$. On the other hand, the benefits of the scheme are profound. It buys the ability to be completely decentralized: receivers can decode without knowing the network topology or the encoding functions; receivers can decode even if nodes or edges are added or removed in an ad hoc fashion; receivers can decode with packet losses or node or link failures without being told the locations of the losses or failures; and receivers can decode even if the local encoding vectors are time-varying and randomly chosen. We make extensive use of the latter property in next section.

Because removals, failures, or losses may reduce the minimum cut to any given receiver below h, or there may be accidental rank reduction due to a poor random choice of local encoding vectors, for further robustness erasure protection may also be necessary. Even in ideal conditions, the sender may not accurately know the broadcast capacity. Moreover, there may be situations in which the sender wishes to communicate more information to receivers with a larger receiver capacity (the maximum flow or minimum cut between the sender and receiver). Erasure coding can help in all of these situations.

The basic form of erasure protection in network coding is to send redundant packets. For example, if the last $h - k$ of the h source vectors $\mathbf{x}_1, \ldots, \mathbf{x}_h$ are known a priori to be zero, then a receiver can still decode the first k source vectors if the rank of the received global encoding vectors is at least k.

A more sophisticated form of erasure protection is based on the Priority Encoding Transmission (PET) technique of Albanese et al. [9]. PET is an unequal erasure protection scheme in which the h source vectors are each identically partitioned into h layers of increasing importance, and layers with higher importance get a higher degree of protection, or redundancy, as illustrated in Figure 1. If a receiver receives only a single global encoding vector, then it can recover the symbols in the most important layer. In general, if a receiver receives global encoding vectors with rank k, then it can recover the symbols in the most important k layers. This is especially useful when broadcasting audio or video data, which can be naturally partitioned into layers with different perceptual importance. There are a number of algorithms for optimizing such layers (e.g., [10, 11, 12] and others, which minimize the expected distortion given the distortion-rate function $D(R)$ of the source and the probability $p(k)$ of receiving rank k). It is not necessary for a receiver to know in advance the boundaries N_k between layers $k-1$ and k in the source vectors. These boundaries can be communicated as part of the packet format [13]. Using this scheme, receivers with receiver capacities higher than the broadcast capacity can receive correspondingly higher quality streams if the sending rate is sufficiently high. Hence in our simulation results we measure throughput in terms of the rank received at each receiver.

3 Buffering Model

By itself, the scheme in the previous section is not sufficient for network coding in real networks. In real networks, packets are not carried in synchrony over unit capacity edges. Instead, packets are likely to be carried sequentially with other packets related to the same set of source vectors, $\mathbf{x}_1, \ldots, \mathbf{x}_h$, over the same edge; packets on different edges are generally subject to different propagation and queueing delays; and the number of packets carried on an edge related to the same set of source vectors generally varies to due to packet loss, congestion, or other changes in the available bandwidth due to competing traffic. For all of these reasons, synchronization of the packets related to the same set of source vectors becomes an important practical issue at both encoding and decoding nodes. In this section, we propose a buffer model to address this issue.

In this paper, all packets related to the same set of h source vectors $\mathbf{x}_1, \ldots, \mathbf{x}_h$ are said to be in the same *generation*, and h is said to be the *generation size*. All packets in the same generation are tagged with the same *generation number*. Sequential generations receive sequentially increasing generation numbers. One or two bytes (mod 2^8 or 2^{16}) in each packet header is sufficient to distinguish between successive generations in the network.

In addition to generation numbers in the packet headers, a mechanism is needed at each node to synchronize the packet arrivals and departures. Buffering can accomplish this. In our buffer model, packets that arrive at a node on any of the incoming edges are put into a single buffer sorted by generation number. Then, whenever there is a transmission opportunity at an outgoing edge, a packet is generated containing a random linear combination of all the packets that are already in the buffer within in the "current" generation. Periodically, the current generation is advanced and the old generation is flushed from the buffer according to one of a number of possible flushing policies discussed shortly. Packets that arrive for a generation that has been flushed are discarded. Note that the packets in the buffer for a given generation increase in number over time, and the linear combinations are chosen randomly for each outgoing packet. Hence the local encoding functions are time-varying. However, as noted in the previous section, this does not present any decoding difficulties because the global encoding vectors are included in the packets.

The first packet that arrives in the buffer for a generation represents new knowledge in that it restricts each component of the source vectors $\mathbf{x}_1, \ldots, \mathbf{x}_h$ to an $(h-1)$-dimensional linear variety or coset. This is because every packet carried on an edge represents a linear constraint of the form $y = \mathbf{g} \cdot [x_1, \ldots, x_h]$ for each component. Subsequent packets may or may not further reduce the dimensionality of each coset, and accordingly are termed *innovative* or *non-innovative*. In other words, innovative packets contain vectors $\mathbf{g}$ that lie outside the subspace spanned by vectors $\mathbf{g}$ already in the buffer, while non-innovative packets contain vectors $\mathbf{g}$ that lie inside this subspace. Since non-innovative packets do not change the subspace from which outgoing vectors are randomly generated, they are useless and may be safely discarded from the buffer.

Since non-innovative information does not affect the transmission of innovative information, every node can freely transmit on all its outgoing edges without having to know whether the information it transmits will be innovative to its neighbors or not. This allows the broadcast system to be completely distributed. No node or central authority needs to know the global topology, the existence of cycles or flows, or even the general upstream and downstream directions. Instead, every node may freely transmit on all its

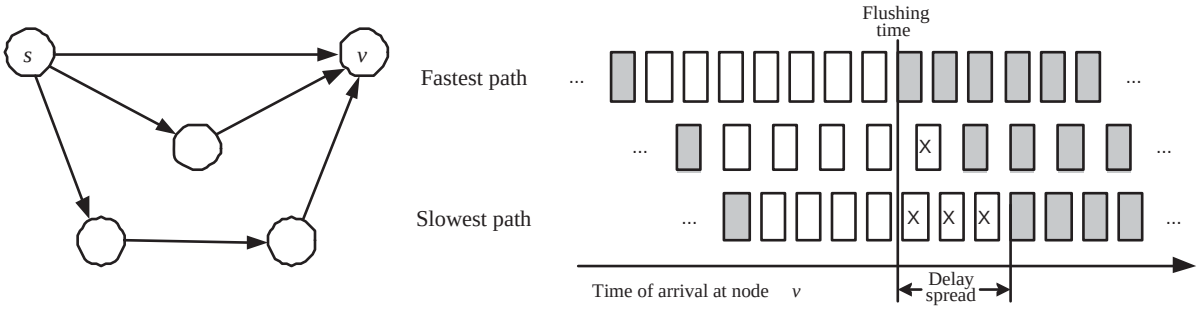

Figure 2: (a) Paths to a node with different delays. (b) Arrival times of packets transmitted over these paths. Discarded packets are marked with an $\times$.

outgoing edges and receive on all its incoming edges. This is ideal for ad hoc networks or broadcast systems that need minimal management or control information.

Packets that are transmitted but turn out to be non-innovative nevertheless use up bandwidth that could be used for other purposes. This bandwidth can be saved, within a distributed setting, if each node monitors the innovation rates along its incoming edges and arranges with its neighbors to restrict their transmission rates to their innovation rates. In this way, non-contributing edges can essentially be removed from the graph.

An alternative approach to economizing network resources is first to run a distributed maxflow algorithm (such as [14]) between the sender s and each receiver $t \in T$, and then to restrict subsequent network coding to these flows. This is the approach we take for the results in Section 4. We nevertheless perform Gaussian elimination at each node as each packet arrives and is inserted into the buffer. This keeps the vectors in the buffer in standard form, allowing immediate determination of whether a packet is innovative or not. Non-innovative packets are discarded.

Decoding at a receiver is no different than the Gaussian elimination performed at any node. In this sense, every node can be a receiver. Performing Gaussian elimination after every received packet ensures the earliest possible decoding for every source vector. In fact the matrix G_t of global encoding vectors received at a node tends to be approximately lower triangular, and hence it is usually possible to decode the kth source vector after receiving fewer more than k packets. We will show in Section 4 that in practice such *earliest decoding* yields a much lower decoding delay than *block decoding*, and in fact in many cases the decoding delay is almost independent of the block length h.

One of the main issues in our buffering model is the policy of when to flush the current generation and advance to the next generation. The simplest policy is to flush the current generation when the first packet of the next generation arrives on any incoming edge. This is a robust as well as simple policy, and most of the results reported in Section 4 use this policy. However, it sustains some loss in throughput compared to capacity. This is due to the finiteness of each generation and the *delay spread* at a node. Delay spread is the difference in time that it takes the first packet in a generation to reach a node over the fastest and slowest paths. Figure 2 illustrates how delay spread might lead to a reduction in achievable throughput. Each row of boxes represents the times of arrival at a node v of packets traveling from the source s over different paths, with the top row representing the path with the lowest delay and the other rows representing paths with higher delay. The current (white) generation is flushed on arrival of the first packet in the next (gray) generation, causing subsequent packets that arrive for the current generation to be discarded. The resulting loss in throughput is approximately proportional to the

fraction of discarded packets within a generation,

$$\begin{aligned} \text{Throughput loss} &\propto \text{delay spread (seconds)/generation duration (seconds)} \\ &= \text{delay spread (seconds)} \times \text{sending rate (packets per second)}/hI, \end{aligned}$$

where h is the generation size and I is the interleaving length. (Assume $I = 1$ for the moment.) Hence networks with a low bandwidth-delay product are expected to have a low loss in throughput. For networks with a high bandwidth-delay product, there are several options. Increasing the generation size h can inversely decrease the throughput loss, in terms of received rank, as we verify in the next section. However, it also increases the packet header size linearly, which negatively impacts net throughput in bits per second. Another solution, which is very effective, is increasing the interleaving length I. The interleaving length is the number of logical sessions into which the original multicast session is partitioned and separately buffered. A generation with generation number n is assigned to session i if $i = n \bmod I$. Thus the sending rate in each logical session and hence its bandwidth-delay product, and hence its throughput loss, is essentially reduced by a factor of I. Throughput loss can be reduced still further by more sophisticated flushing policies. However, cycles that are essential to achieving the broadcast capacity can cause an inherently large delay spread for some nodes. These phenomena are studied in [15].

4 Simulation Results

We implemented the network coding scheme described in the previous sections using an event-driven network simulator written in C++. We performed extensive experiments on the graphs of six ISP backbones obtained from the Rocketfuel project at the University of Washington [16, 17]. Because the Rocketfuel methodology does not directly infer edge capacities, we arbitrarily set the capacity of each edge to 1Gbps/$weight$, where $weight$ is the cost inferred by Rocketfuel of transmitting over the edge relative to other edges. We set the edge latencies according to the data from Rocketfuel, that is, equal to the link propagation delay due to the speed of light.

Due to space limitations, here we report results only for the SprintLink ISP graph. Results for the other ISP graphs are similar. We placed the sender at Seattle and arbitrarily selected 20 receivers, trying to select nodes with different receiver capacities. We then reduced the original graph, which has 89 nodes and 972 bi-directional edges, to a union of the maximum flows from the sender to each receiver, resulting in 89 nodes and 207 bi-directional edges. This subgraph is sufficient to preserve the broadcast capacity while economizing network resources. In the following, we include performances for five receivers: Chicago, Pearl Harbor, Anaheim, Boston, and San Jose, which have respective receiver capacities 450, 525, 625, 733, and 833 Mbps. The broadcast capacity is 450 Mbps.

Here we study throughput and decoding delay as a function of time, sending rate, edge latency, field size, generation size, and interleaving length. Unless otherwise specified, the sending rate is 450 Mbps, the field size is 2^{16}, the generation size is 100, and the interleaving length is 100. We send approximately 20000 packets in each experiment, e.g., broken into two contiguous sets of 100 interleaved generations, each generation containing 100 packets.

Figure 3(left) shows the received rank as a function of time (or generation number) for

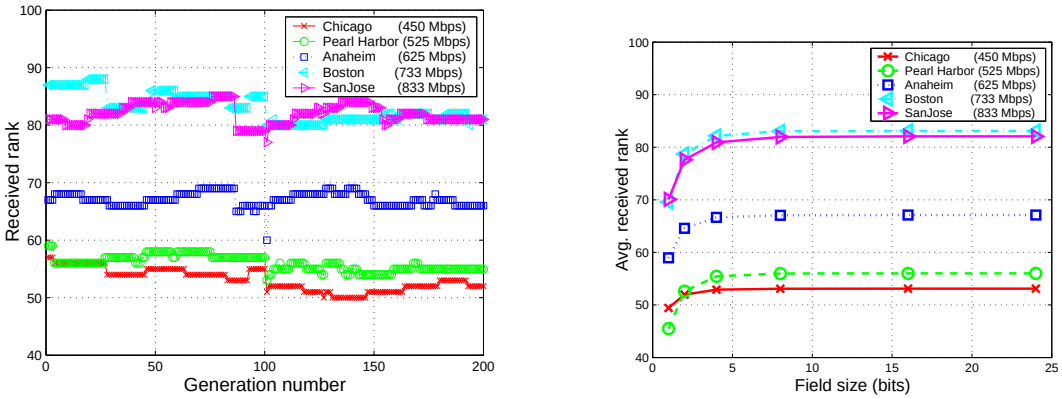

Figure 3: (Left) Received rank over time. (Right) Average received rank vs. field size. SendingRate=830 Mbps.

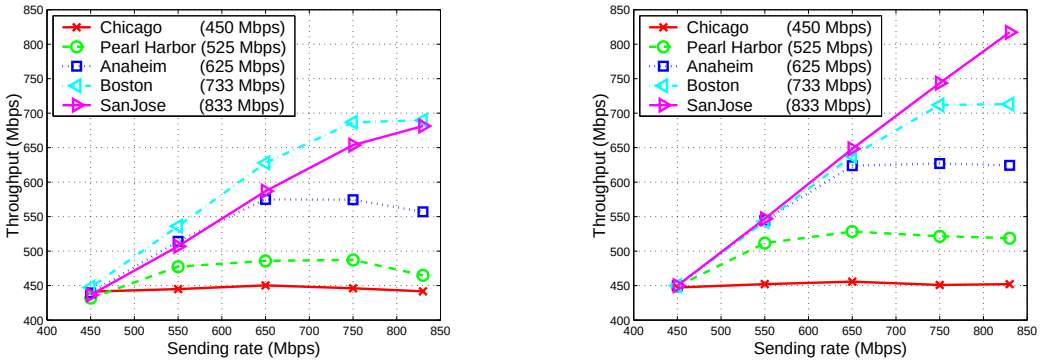

Figure 4: Throughput vs. sending rate, with (left) original edge latencies and (right) edge latencies reduced by a factor of 100.

each of the five receivers, plotted in different line styles for each receiver. The maximum rank is $h = 100$, which would correspond to a throughput equal to the sending rate of 830 Mbps. Receivers with lower receiver capacity receive correspondingly lower rank. It can be seen that the variation over time for each receiver is on the order of several percent. Figure 3(right) shows average received rank vs. field size for the different receivers, again for a sending rate of 830 Mbps. It can be seen that the average received rank peaks for field size $= 2^8$ or 2^{16} regardless of receiver capacity.

Figure 4(left) shows throughput as a function of sending rate. Throughput is measured as the sending rate times the average received rank divided by the generation size. Observe that throughput for each receiver approximately grows with sending rate, but then saturates below the receiver's capacity. The gap between a receiver's maximum throughput and its capacity is attributable to delay spread. Figure 4(right) shows the same information, when edge latencies are reduced by a factor of 100. This is equivalent to reducing the physical size of the network or else reducing both the edge capacities and sending rate by a factor of 100. The maximum throughput is then closer to capacity because the delay-bandwidth product of the network is reduced.

Figures 5(left) and 5(right) respectively show the average loss in throughput as a function of generation size h and interleaving length I. It can be seen that the loss is approximately inversely proportional to both h and I, validating our delay spread model.

Finally, Figures 6(left) and 6(right) respectively show the average packet delay as a function of generation size h and interleaving length I, for the Anaheim receiver, which is

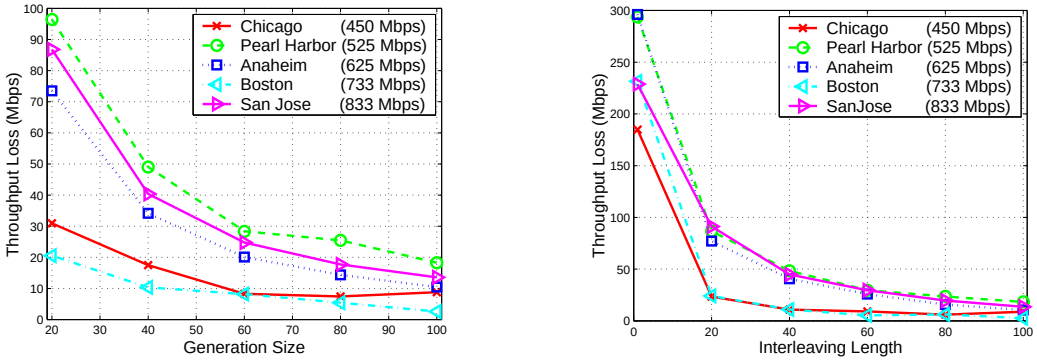

Figure 5: Throughput loss vs. (left) generation size and (right) interleaving length.

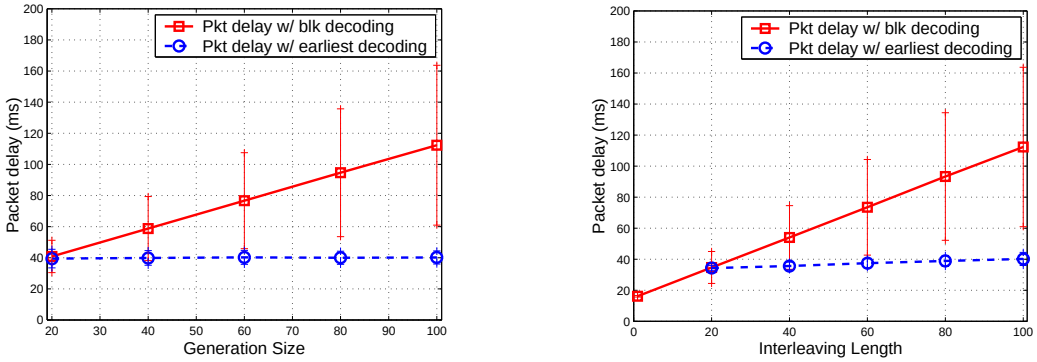

Figure 6: Packet delay vs. (left) generation size and (right) interleaving length.

a typical receiver. This is the delay between the time a source vector $\mathbf{x}_i$ is encoded by the sender and decoded by the receiver. It can be seen that the average packet delay (as well as the standard deviation, indicated by vertical lines) increases linearly with generation size and/or interleaving length, for block decoding, while remaining almost constant for earliest decoding.

5 Conclusion

We introduced a scheme for practical network coding in real networks, and simulated the scheme on graphs of several Internet Service Providers. The scheme uses buffering to synchronize arbitrary packet arrivals and departures at each node, random encoding to deal with varying numbers of packets in the buffer, and a packet format that includes global encoding vectors to provide the receivers with just the right information to decode the packets under such time-varying network conditions. The scheme can achieve throughput close to capacity with low delay.

Acknowledgments

The authors would like to thank Dr. Jin Li for the use of his erasure coding library.

References

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

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

[3] R. Koetter and M. Médard. An algebraic approach to network coding. *IEEE/ACM Trans. Networking*. To appear.

[4] S. Jaggi, P. A. Chou, and K. Jain. Low complexity optimal algebraic multicast codes. In *Proc. Int'l Symp. Information Theory*, Yokohama, Japan, June 2003. IEEE.

[5] P. Sander, S. Egner, and L. Tolhuizen. Polynomial time algorithms for network information flow. In *Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 286–294, San Diego, CA, June 2003. ACM.

[6] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen. Polynomial time algorithms for network code construction. *IEEE Trans. Information Theory*. Submitted.

[7] T. Ho, D. R. Karger, M. Médard, and R. Koetter. Network coding from a network flow perspective. In *Proc. Int'l Symp. Information Theory*, Yokohama, Japan, June 2003. IEEE.

[8] T. Ho, R. Koetter, M. Médard, D. R. Karger, and M. Effros. The benefits of coding over routing in a randomized setting. In *Proc. Int'l Symp. Information Theory*, Yokohama, Japan, June 2003. IEEE.

[9] A. Albanese, J. Blömer, J. Edmonds, M. Luby, and M. Sudan. Priority encoding transmission. *IEEE Trans. Information Theory*, 42:1737–1744, November 1996.

[10] G. Davis and J. Danskin. Joint source and channel coding for image transmission over lossy packet networks. In *Conf. Wavelet Applications to Digital Image Processing*, Denver, CO, August 1996. SPIE.

[11] A. E. Mohr, E. A. Riskin, and R. E. Ladner. Unequal loss protection: Graceful degradation of image quality over packet erasure channels through forward error correction. *IEEE J. Selected Areas in Communications*, 18(6):819–829, June 2000.

[12] R. Puri and K. Ramchandran. Multiple description source coding through forward error correction codes. In *Proc. Asilomar Conference on Signals, Systems, and Computers*, Asilomar, CA, October 1999. IEEE.

[13] G. Leibl, T. Stockhammer, M. Wagner, J. Pandel, G. Baese, M. Nguyen, and F. Burkert. An RTP payload format for erasure-resilient transmission of progressive multimedia streams. Internet Draft draft-ieft-avt-uxp-00.txt, IETF, February 2001. Expired.

[14] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35(4):921–940, October 1988.

[15] Y. Wu, P. A. Chou, and K. Jain. Practical network coding. Technical report, Microsoft Research. In preparation.

[16] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proc. SIGCOMM*, Pittsburg, PA, August 2002. ACM.

[17] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring link weights using end-to-end measurements. In *Proc. Internet Measurement Workshop*, Marseille, France, November 2002. ACM.