Philip A. Chou and Yunnan Wu

Network Coding for the Internet and Wireless Networks

[Theory, practice, and applications of a promising routing generalization]

In today's practical communication networks such as the Internet, information delivery is performed by routing. A promising generalization of routing is network coding. The potential advantages of network coding over routing include resource (e.g., bandwidth and power) efficiency, computational efficiency, and robustness to network dynamics. This tutorial article provides an overview of the theory, practice, and applications of network coding.

INTRODUCTION

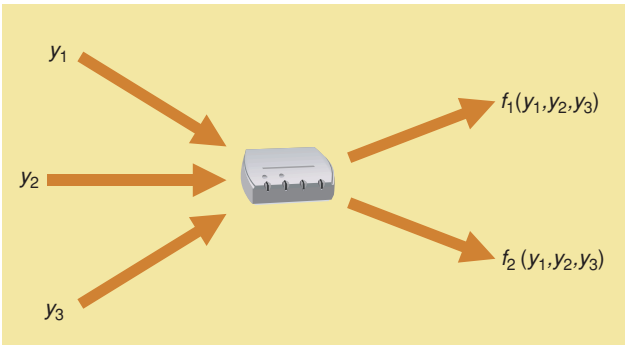
What is network coding? For a simple answer to that question, consider a router in a computer network. Today, a router can merely route, or forward, messages. Each message on an output link must be a copy of a message that arrived earlier on an input link. Network coding, in contrast, allows each node in a network to perform some computation. Thus, each message sent on a node's output link can be some function or mixture of messages that arrived earlier on the node's input links, as illustrated in Figure 1. Generally, network coding is the transmission, mixing (or encoding), and remixing (or reencoding) of messages arriving at nodes inside the network, such that the transmitted messages can be unmixed (or decoded) at their final destinations.

Digital Object Identifier 10.1109/MSP.2007.904818

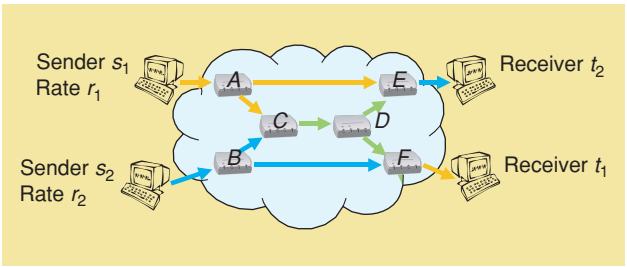
ADVANTAGE #1: MAXIMIZING THROUGHPUT

Network coding has several advantages over routing. The first is the potential of network coding to improve throughput. Consider the following situation. Two streams of information, both at bitrate B b/s, arrive at a node, contending for an output link, having capacity B b/s. With network coding, it may be possible to increase throughput by pushing both streams through the bottleneck link at the same time. The method is simple. Using network coding, the node can mix the two streams together by taking their exclusive-OR (XOR) bit-by-bit and sending the mixed stream through the link. In this case, XOR is the function computed at the node. This increases the throughput of the network if the two streams can be disentangled before they reach their final destinations. This can be done using side information if it is available downstream.

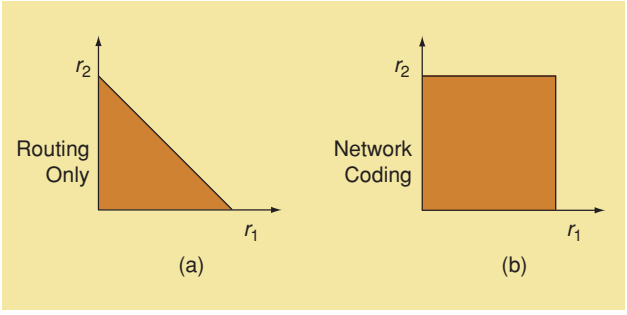
As an example, Figure 2 illustrates an idealized network of routers, links, and computers. In the example, all the links are directional, and have the same capacity, say B b/s. Computer s_1 wishes to send information to Computer t_1, and Computer s_2 wishes to send information to Computer t_2. Observe that s_1 can reach t_1 only by the path $s_1 \to A \to C \to D \to F \to t_1$, and that s_2 can reach t_2 only by the path $s_2 \to B \to C \to D \to E \to t_2$. These share the bottleneck link $C \to D$.



[FIG1] **Network Coding: Network nodes can compute functions of input messages.**



[FIG2] **Two unicast sessions contending for a bottleneck link.**



[FIG3] **Achievable communication rate regions for (a) routing only and (b) network coding.**

At what rates, r_1 and r_2, can the two sessions communicate reliably? If the network nodes can only route information, then clearly the bottleneck link $C \to D$ must be timeshared between the two sessions, giving rise to the set of achievable rates $\{(r_1, r_2) : 0 \le r_1,\ 0 \le r_2,\ r_1 + r_2 \le B\}$, which is shown in Figure 3(a). However, if the network nodes can perform network coding, then both sessions can communicate reliably at rate B, giving rise to the set of achievable rates $\{(r_1, r_2) : 0 \le r_1 \le B,\ 0 \le r_2 \le B\}$, which is shown in Figure 3(b). To achieve the pair of rates (B, B), for example, the two streams can be mixed at node C using an XOR operation, and they can be purified downstream at nodes E and F, again using XOR operations. Figure 2 shows the information flow required for this scheme. The pure streams are carried by the yellow and blue links, while the mixed stream is carried by the green links.

Clearly, it is not possible for either session to communicate reliably at a rate greater than B, since the sender and receivers in each session are connected to the network through a single link of capacity B. Hence the region in Figure 3(b) is the set of all achievable rates, which is called the capacity region for these sessions in this network.

It turns out to be very difficult, in general, to determine the capacity region for an arbitrary set of sessions in an arbitrary network. To be specific, let a network (V, E, c) be represented by a set of nodes (or vertices) V, a set of directed links (or edges) E, and real-valued capacities $c(e)$ on each link $e \in E$, and let a session (s, T) be represented by a sender $s \in V$ and a set of receivers $T \subseteq V$. (If the set of receivers consists of only a single node t, then the session is said to be a unicast session. Otherwise it is said to be a multicast session.) Given a network (V, E, c) and a set of sessions $(s_1, T_1), \ldots, (s_N, T_N)$ with respective communication rates $r_1, \ldots, r_N$, the decision problem, "Is the vector of communication rates $(r_1, \ldots, r_N)$ achievable, or not?" is still a difficult problem. In many specific networks and sets of sessions, the answer to a specific decision problem may be obvious, for example, when the specified rates are particularly high or low. Indeed, inner and outer bounds on the capacity region for any specific network and set of sessions can be readily provided. Moreover, in many instances, these inner and outer bounds match, so that the capacity region is determined. However, a precise characterization of the capacity region for an arbitrary network and set of sessions has remained elusive.

Happily, when there is only a single session (s, T) in any given network (V, E, c), the capacity region $[0, C]$, or simply the capacity C, is now well characterized, thanks to the recent invention of network coding. Most of the remainder of this tutorial is devoted to this single session case.

SINGLE-SESSION NETWORK CODING

In this section, we cover some of the basic theory of single-session network coding, beginning with the unicast case.

THE UNICAST CASE: MINCUT BOUND, MENGER'S THEOREM, AND PATH PACKING

In the unicast case, a session (s, t) consists of a single sender s and a single receiver t. Let $r(s, t)$ designate an achievable rate of communication from s to t in a given network (V, E, c). It has long been known that an upper bound on $r(s, t)$ is the value of any s–t cut through the network. An s–t cut is a partition of the network into two subsets, U and $\overline{U}$, the first containing s and the second containing t. The value of this cut is the sum of the capacities of the links from nodes in U to nodes in $\overline{U}$, as illustrated in Figure 4.

The minimum of the values of all such s–t cuts, herein designated $\text{MinCut}(s, t)$, is clearly also an upper bound on $r(s, t)$. That is, $r(s, t) \leq \text{MinCut}(s, t)$. In 1927, Menger proved a variant of the following theorem, which was later to become known as the MaxFlow—MinCut Theorem: for undirected graphs with unit capacity edges, there always exists a set of $h = \text{MinCut}(s, t)$ edge-disjoint paths between s and t [1]. Thus, by routing information over this set of h unit-capacity edge-disjoint paths, we can achieve reliable communication between s and t at the maximum possible rate, $r(s, t) = \text{MinCut}(s, t)$. In 1956, Ford and Fulkerson (and independently Elias, Feinstein, and Shannon) provided constructive algorithms for finding the maximum flow, i.e., for packing the maximum number of unit-capacity edge-disjoint paths from s to t, in polynomial time [2], [3]. These results were later extended to include directed graphs, edges with real-valued capacities, and more efficient algorithms; see for example [4]. Thus Menger's theorem and the Ford-Fulkerson algorithm provide the basis for achieving the capacity of a network in the single-session unicast case.

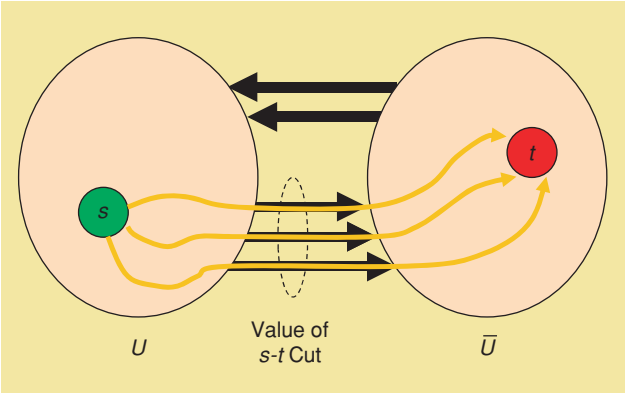
THE BROADCAST CASE: EDMONDS' THEOREM AND SPANNING TREE PACKING

In the broadcast case, a session consists of a single sender s and a set of receivers V consisting of all the nodes in the network (V, E, c). Let $r(s, V)$ designate an achievable rate at which s can broadcast common information reliably to all the other nodes in the network. Clearly, an upper bound on $r(s, V)$ is the maximum rate at which s can communicate reliably with any single receiver $v \in V$. Thus, $r(s, V) \leq \min_{v \in V} \text{MinCut}(s, v)$. It turns out that the upper bound, $\min_{v \in V} \text{MinCut}(s, v)$, is also achievable, as proved in 1972 by Edmonds, who showed for directed graphs with unit capacity edges that the maximum number of edge-disjoint directed spanning trees rooted at s is equal to $\min_{v \in V} \text{MinCut}(s, v)$ [5]. (A directed spanning tree rooted at s reaches every node in V through edges in E.) By routing information over these spanning trees, we can achieve reliable broadcast from s to V at the maximum possible rate, $r(s, V) = \min_{v \in V} \text{MinCut}(s, v)$. Thus, $C = \min_{v \in V} \text{MinCut}(s, v)$ can be considered the broadcast capacity of the network. Moreover, a maximal set of spanning trees achieving the broadcast capacity can be found in polynomial time.

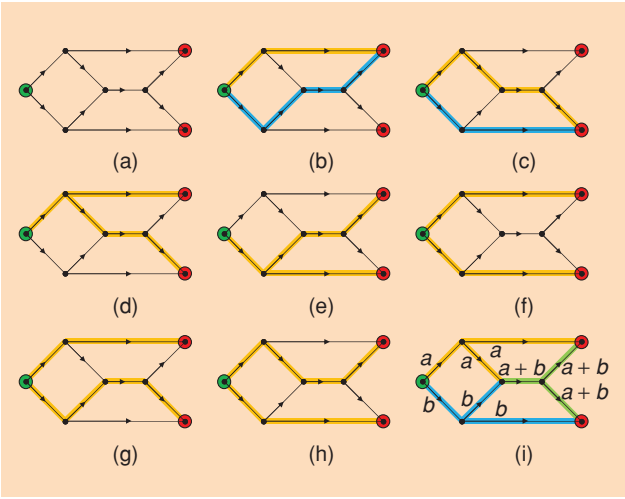
THE MULTICAST CASE: STEINER TREE PACKING AND NON-ACHIEVABILITY OF MINCUT BOUND

In the multicast case, a session consists of a single sender s and a set of receivers $T \subseteq V$. Let $r(s, T)$ designate an achievable rate at which s can multicast common information reliably to all the nodes in T. Clearly, $\text{MinCut}(s, t)$ remains an upper bound on this rate for any $t \in T$. Thus, $r(s, T) \leq \min_{t \in T} \text{MinCut}(s, t)$. Unfortunately, unlike the broadcast case, the upper bound may not be achievable by routing information through a set of edge-disjoint trees, as the following example shows.

Figure 5(a) shows a seven-node network with unit-capacity directed links. A sending node s is on the left (in green) and two receiving nodes $T = \{t_1, t_2\}$ are on the right (in red). Figure 5 (b) and (c) show that there are two edge-disjoint directed paths from the sender to each of the two receivers. Hence, $\min_{t \in T} \text{MinCut}(s, t) = 2$. However, multicast at this rate is not achievable merely by routing information along a set of edge-disjoint trees. Figure 5 (d)–(h) show the only possible Steiner trees from s to T. (A Steiner tree, also known as a multicast tree, from s to T in a graph (V, E) is a tree rooted at s that reaches every node in T through edges in E.) Any two of these Steiner trees share at least one edge. Hence, in this network the maximum number of edge-disjoint Steiner trees from s to T is 1.



[FIG4] **Upper bound on unicast capacity.**



[FIG5] **One Multicast Session. (a) Sender s in green, receivers $T = \{t_1, t_2\}$ in red. (b), (c) Maximal sets of edge-disjoint paths from s to t_1 and t_2, demonstrating $\min_{t \in T} \text{MinCut}(s, t) = 2$. (d)–(h) All five possible multicast trees from s to T, no two of which are edge-disjoint. (i) Network coding achieves the multicast capacity.**

Thus routing along a maximal set of edge-disjoint Steiner trees cannot in general achieve throughput equal to the upper bound $\min_{t\in T}\text{MinCut}\ (s, t)$. To be more precise, the maximum throughput for routing is achieved by a fractional packing of Steiner trees, where each Steiner tree can be used for a fraction of the time, with the average usage of each edge not exceeding its capacity. It can be shown that the maximum routing throughput for this example is 1.5, which is still less than the minimum of the min-cut values, 2. Worse, finding such a maximal set of edge-disjoint Steiner trees turns out to be NP-hard [6].

MULTICAST CAPACITY

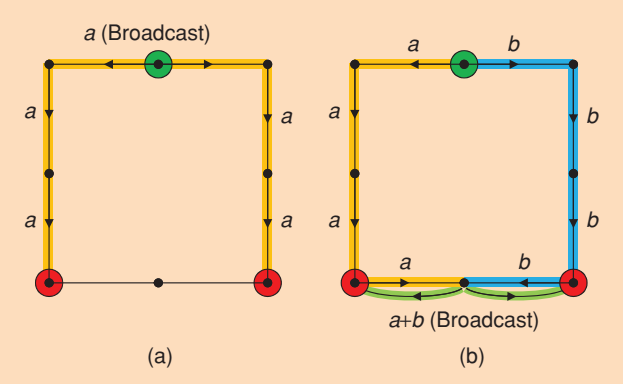
Nevertheless, reliable multicast from s to T in Figure 5 can occur at the upper bound if network coding is used. Figure 5 (i) shows how two unit-bandwidth streams, a and b, can be encoded at an interior node to produce a mixed stream, $a+b$, from which stream a can later be subtracted to recover stream b, and vice versa, thus delivering both streams to both receivers. Here, addition and subtraction are operations over a finite field, specifically XOR operations. Amazingly, reliable multicast at a rate equal to the upper bound, $\min_{t\in T}\text{MinCut}(s, t)$, can always be achieved in any network using network coding, as proved in 2000 in the seminal work of Alswede et al. [7]. (We shall not display a proof here, but see Koetter and Médard [8] for a beautiful algebraic proof, or see either of the tutorials listed in the Conclusion.) Thus, $h = \min_{t\in T}\text{MinCut}(s, t)$ can be considered to be the multicast capacity, or simply the capacity (since the formula works for unicast and broadcast as well) for an arbitrary single session in an arbitrary directed network.

Perhaps more amazingly, linear network coding is sufficient to achieve the multicast capacity, as proved in 2003 by Li et al. [9] and by Koetter and Médard [8]. Linear network coding means that the messages (e.g., y_1, y_2, and y_3 in Figure 1) can be considered vectors of elements from a finite field, and the functions performed at the nodes can be simple linear combinations over this finite field (e.g., $f_1(y_1, y_2, y_3) = \alpha_1 y_1 + \alpha_2 y_2 + \alpha_3 y_3$ and $f_2(y_1, y_2, y_3) = \beta_1 y_1 + \beta_2 y_2 + \beta_3 y_3$). Furthermore, all the decoding at the receivers can be performed using linear operations. Jaggi et al. provided a polynomial time algorithm for finding the encoding and decoding coefficients in directed acyclic networks [10], and Erez and Feder extended the approach to directed networks with cycles [11].

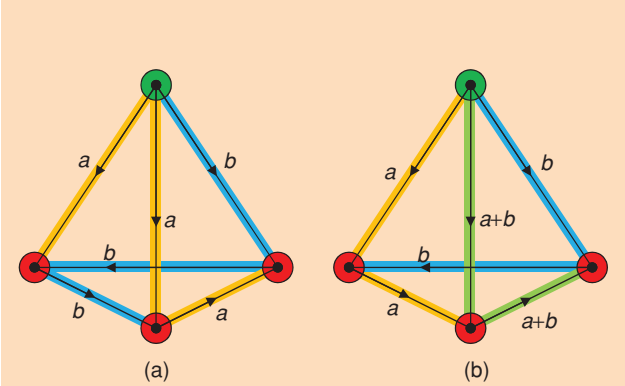
In summary, using linear network coding, the multicast capacity in a directed network can always be achieved, and the coding coefficients necessary to achieve the capacity can be computed in polynomial time. In contrast, if only routing can be used, not only is it generally impossible to achieve the multicast capacity, but computing the set of edge-disjoint multicast trees necessary to achieve the best possible routing is a problem that is NP-hard in general.

ADVANTAGE #2: MINIMIZING ENERGY PER BIT

There are advantages to network coding beyond maximizing throughput. In particular, network coding can minimize the amount of energy required per packet (or other unit) of information multicast in a wireless network. Figure 6 shows a wireless network with nodes arranged in a square, with radio ranges such that the nodes can directly communicate with neighbors horizontally and vertically, but not diagonally. There is a single multicast session with a sender s at the top center and receivers t_1 and t_2 at the bottom left and right corners. Assuming that each transmission takes one unit of energy, we can use the number of transmissions as a proxy for the amount of energy required to multicast each packet. If only routing is permitted, then it is possible to show that a minimum of five transmissions is required to multicast a packet from s to t_1 and t_2. [For example, the first transmission broadcasts the packet to the sender's two neighbors, and four other transmissions move the packet to the two receivers, as illustrated in Figure 6(a).] However, if network coding is permitted, then only four and one-half transmissions per packet are required on average, using nine transmissions for two packets a and b. [For example, three transmissions can move packet a to receiver t_1, three transmissions can move packet b to receiver t_2, two transmissions can move packets a and b to an intermediate node, and a final transmission can broadcast $a+b$ back out to the receivers, as illustrated in Figure 6(b).] It can be shown that under this model of a wireless network, linear network coding can always achieve the minimum energy per packet and the required coding coefficients can be computed in polynomial time [12]. In contrast, minimum energy multicast routing is NP-hard to compute, and may not even achieve the minimum possible energy.



[FIG6] Network coding minimizes energy per packet. Sender s_1 is in green; receivers t_1 and t_2 are in red.



[FIG7] Network coding minimizes delay.

ADVANTAGE #3: MINIMIZING DELAY

Network coding can also minimize the delay, as measured, for example, by the maximum number of hops for a packet to reach a receiver. Figure 7 shows a network of four nodes arranged in a tetrahedron, with unit-capacity edges running down the sides and around the bottom in a cycle. There is a single sender at the top and three receivers at the bottom. It is easy to verify that the MinCut between the sender and any receiver is two. Edmonds' theorem therefore guarantees the existence of two edge-disjoint spanning trees along which the sender can route two unit-rate streams to the three receivers. Figure 7(a) shows essentially the only such spanning trees, modulo symmetries. Note that the depth of the blue tree is three, which is therefore the minimum possible overall delay if only routing can be used to communicate at rate two. In contrast, Figure 7(b) shows that if network coding can be used, it is possible to reduce the delay to two, by routing stream a along the yellow path, stream b along the blue path, and their mixture $a + b$ along the green path.

APPLICABILITY TO REAL NETWORKS: THEORY VERSUS PRACTICE

Network Coding is presumably highly applicable to communication in real networks. In the following section, we address how practical network coding might be done in real networks.

PRACTICAL NETWORK CODING

Making network coding practical relies on three key ideas: random coding, packet tagging, and buffering. Random coding allows the encoding to proceed in a distributed manner. Tagging each packet with the corresponding coding vector allows the decoding to proceed in a distributed manner. Buffering allows for asynchronous packet arrivals and departures with arbitrarily varying rates, delay, and loss. Before introducing these techniques, however, let us establish the general algebraic framework behind linear network coding.

LOCAL AND GLOBAL ENCODING VECTORS, DECODING

To begin, consider an acyclic network (V, E, c) with unit capacity edges, i.e., $c(e) = 1$ for all $e \in E$, meaning that each edge can carry one symbol per unit of time. Assume also that each symbol is an element of a finite field F. Let there be a single sender $s \in V$ and a set of receivers $T \subseteq V$. Let $h = \text{MinCut}(s, T)$ be the multicast capacity. Let $x_1, \ldots, x_h$ be the h symbols that we wish to multicast from s to T in each unit of time.

For each edge e emanating from a node v, let $y(e)$ denote the symbol carried on e. The symbol $y(e)$, regarded as an element of the finite field F, can be computed as a linear combination of the symbols $y(e')$ on edges e' entering node v, namely, $y(e) = \sum_{e'} \boldsymbol{\beta}_{e'}(e) y(e')$. The coefficients of the linear combination form a vector $\boldsymbol{\beta}(e) = [\boldsymbol{\beta}_{e'}(e)]$, known as the local encoding vector on edge e. The length of this vector is the number of edges e' entering v. The local encoding vectors on edges e leaving v characterize the network functions performed at v.

For the sake of uniformity of notation, we can introduce artificial edges $e'_1, \ldots, e'_h$ entering s, and let the symbols $y(e'_1), \ldots, y(e'_h)$ on these edges be equal to $x_1, \ldots, x_h$. Then, by induction, it is clear that the code symbol $y(e)$ on any edge $e \in E$ in the network can be computed as a linear combination of the source symbols $x_1, \ldots, x_h$, namely, $y(e) = \sum_{i=1}^{h} g_i(e) x_i$. The coefficients of this linear combination form a vector $\boldsymbol{g}(e) = [g_1(e), \ldots, g_h(e)]$, known as the global encoding vector on edge e. The global encoding vector $\boldsymbol{g}(e)$ represents the code symbol $y(e)$ in terms of the source symbols $x_1, \ldots, x_h$. It is easy to see that the global encoding vectors themselves can be computed recursively as $\boldsymbol{g}(e) = \sum_{e'} \beta_{e'}(e) \boldsymbol{g}(e')$, using the coefficients of the local encoding vectors $\beta(e)$.

Suppose now that a receiver $t \in T$ receives code symbols $y(e_1), \ldots, y(e_h)$ on edges $e_1, \ldots, e_h$ entering t. The received code symbols can be expressed in terms of the source symbols as

$$\begin{bmatrix} y(e_1) \\ \vdots \\ y(e_h) \end{bmatrix} = \begin{bmatrix} g_1(e_1) & \ldots & g_h(e_1) \\ \vdots & \ddots & \vdots \\ g_1(e_h) & \ldots & g_h(e_h) \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_h \end{bmatrix} = G_t \begin{bmatrix} x_1 \\ \vdots \\ x_h \end{bmatrix}, \quad (1)$$

where the ith row of the matrix G_t is the global encoding vector associated with edge e_i entering receiver t. Receiver t can therefore recover the h source symbols by inverting the matrix G_t and applying the inverse to its received code symbols.

RANDOM ENCODING AND INVERTIBILITY

It turns out that G_t will be invertible with high probability if all of the coefficients of all of the local encoding vectors in the network are chosen randomly, independently, and uniformly from the field F, provided that the field size is sufficiently large relative to the size of the network, as shown independently by Ho et al. [13] and Sanders et al. [14]. For example, if $|F| = 2^{16}$ and $|E| = 2^8$, then G_t will be invertible with probability at least $1 - |E|/|F| \approx 0.996$. Empirical evidence suggests that this bound is quite loose; G_t is still invertible with high probability even when $|F| = 2^8$ and the network has hundreds of edges. Thus, random linear codes work well. This is the first key idea towards making network coding practical, since each node can, in a distributed way, choose its own encoding coefficients at random, independently of the other nodes.

PACKET TAGGING

In a real network, symbols flow sequentially over the edges, and moreover, the symbols are grouped into packets. For example, each packet in the Internet can typically contain up to 1,400 B or so. If $|F| = 2^{16}$, then each packet can contain about 700 symbols, whereas if $|F| = 2^8$, then each packet can contain about 1,400 symbols. Thus, we can think of each packet in the network as being a vector of code symbols $\boldsymbol{y}(e) = [y_1(e), \ldots, y_N(e)]$. By likewise grouping the source symbols into packets $\boldsymbol{x}_i = [x_{i,1}, \ldots, x_{i,N}]$, the above algebraic relationships carry over to packets. That is, each packet $\boldsymbol{y}(e)$ on edge e can be computed as

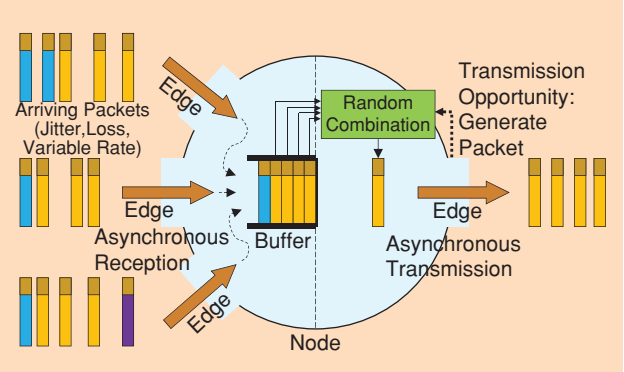
a linear combination of the packets $y(e')$ on the preceding edges or, alternatively, as a linear combination of the source packets $x_1, \ldots, x_h$, i.e., $y(e) = \sum_{e'} \beta_{e'}(e) y(e') = \sum_{i=1}^{h} g_i(e) x_i$. As before, each receiver t can recover the source packets by inverting the matrix G_t and applying the inverse to its received code packets.

Now we come to the second key idea for making network coding practical: packet tagging. Suppose every packet carried on edge e is tagged with the global encoding vector $g(e)$ associated with the edge. This can be easily accomplished, for example, by prefixing the ith information vector x_i with the ith unit vector u_i and applying the usual algebraic operations to the resulting vector. In this way each packet is automatically tagged with the appropriate global encoding vector, since $[g(e), y(e)] = \sum_{e'} \beta_{e'}(e)[g(e'), y(e')] = \sum_{i=1}^{h} g_i(e)[u_i, x_i]$.

The cost of the tag $g(e)$ is h extra symbols per packet. If $h = 50$ and $|F| = 2^8$, then the overhead is about $50/1400 \approx 3\%$, for example. The benefit of the tag, however, is that the global encoding vectors needed to decode the received packets can be found within the packets themselves. This means that the receivers do not need to know the encoding functions within the network, or even the network topology, to compute G_t. Nor does G_t need to be communicated to the receiver a priori or over a side communication mechanism. In fact, the network can be dynamic, with nodes and edges being added and removed in an ad hoc way. Node failure, link failure, and packet loss can occur in unknown locations. The encoding functions can be time varying and random. Thus, decoding can be robust, provided that the network through which the received packets $y(e_1), \ldots, y(e_h)$ are computed maintains MinCut $(s, t) \geq h$. Any receiver t for which MinCut(s, t) falls below h may not be able to decode.

BUFFERING AND GENERATIONS

By themselves, the random coding and packet tagging schemes just outlined are not sufficient to make network coding practical in real networks. In real networks, the unit capacity edges are grouped into real edges, packets on real edges are carried sequentially, the number of packets per unit time on each edge varies due to loss, congestion, and competing traffic, and cycles are everywhere. In addition, the information source may be a continuous stream of packets.



[FIG8] Buffering at a node.

If the information source is a continuous stream of packets, it can be blocked into h source packets per block. Let us say that all the code packets in the network related to the kth block of source packets $x_{kh+1}, \ldots, x_{kh+h}$ belong to generation k, where h is the generation size. To keep track of packets in same generation, each packet can be tagged with its generation number k.

Packets within a generation can now be synchronized by buffering, which is the third key idea for making network coding practical. Figure 8 illustrates a typical network node with three incoming links and one outgoing link. Packets, tagged by generation number (shown as packet color) arrive sequentially through each link, subject to jitter, loss, and variable rate. As packets arrive at the node, they are put into a common buffer sorted by generation number, with the current generation at the head of the queue.

Whenever there is a transmission opportunity on an outgoing link, an outgoing packet is formed by taking a random linear combination of packets in the current generation. Transmission opportunities can occur, for example, for constant bit rate links at fixed intervals, for TCP connections whenever the TCP window slides over, or for wireless links when the MAC gains access to the channel. Periodically, the current generation of packets can be flushed out of the buffer according to a flushing policy. Packets in previous generations that arrive late at a node can be discarded.

EARLIEST DECODING

Decoding at any node can be achieved by collecting h or more packets in a given generation, stacking their symbols row-by-row, extracting the symbols in the packet tags to form G_t, and applying the inverse of G_t, if it exists, to the symbols in the packet payloads. (Equivalently, Gaussian elimination could be performed on the matrix of symbols formed by the stacked packet tags and payloads.) The algorithmic decoding delay in this block decoding method is the length of time for the receiver to collect h packets, which is of course proportional to the generation size h.

A decoding method with lower algorithmic decoding delay is earliest decoding, in which Gaussian elimination is performed immediately after each packet is received, on the matrix of symbols formed by the packets stacked so far. Since G_t tends to be lower triangular (i.e., the ith received packet tends to be a linear combination of the first i source packets because of causality of computation in the network), it is typically possible to decode the first i source packets after receiving fewer more than i code packets. Experimental results show that the algorithmic decoding delay of earliest decoding is often on the order of a few source packets, much smaller than that of block decoding.

ROUTING EXAMPLE

The above techniques for practical network coding were simulated in [15] on the network shown in Figure 9(a), purportedly representing the SpintLink ISP network in North America, as determined by the University of Washington Rocketfuel project [16]. The network consists of 89 nodes and 972 bidirectional edges, with edge capacities scaled inversely proportionally to the link costs inferred by Rocketfuel. A node in Seattle was arbitrarily chosen as the sender, and 20 other nodes were arbitrarily

chosen as receivers, spanning a range with MinCut(s, t) from 450 Mbps (between Seattle and Chicago) to 833 Mbps (between Seattle and San Jose).

For field size $|F| = 2^{16}$ and generation size $h = 100$, Figure 9(b) shows the average rank of the set of global encoding vectors received in each generation, normalized to the sending rate (e.g., a received rank of 75 out of 100 is normalized to 75% of the sending rate) as a function of sending rate (in Mb/s), for five different receivers. The MinCut between Seattle and each receiver is shown in the key. It can be seen that as the sending rate increases, the received rank increases proportionally, up to the MinCut, at which point it saturates. Thus for any collection of receivers T, if the sending rate is limited to the multicast capacity $\min_{t\in T}\text{MinCut}(s, t)$, it is possible in practice to achieve reliable multicast very close to capacity. For further details, see [15].

APPLICATIONS

Despite the above example, building network coding into IP-level routers in the Internet is unlikely to be practical in the near future, for a variety of reasons, e.g., the need to keep computation and per-flow state out of the core, the need for backward compatibility with a massive deployed base, and the need for multipath routing to make network coding effective. However, building network coding into overlay networks is quite feasible. In overlay networks, nodes are application-level programs running in computers and edges are transport-level connections between computers. Overlay networks can be infrastructure-based, as typified by content distribution networks such as Akamai or Limelight, or they can be ad-hoc or peer-to-peer (P2P) networks of end hosts drawn together temporarily to fulfill a particular communication task, such as live broadcast, media on demand, file download, instant messaging, storage, telephony, conferencing, or gaming. In this section, we take a quick romp through such applications of network coding, ending with applications to wireless and sensor networks.

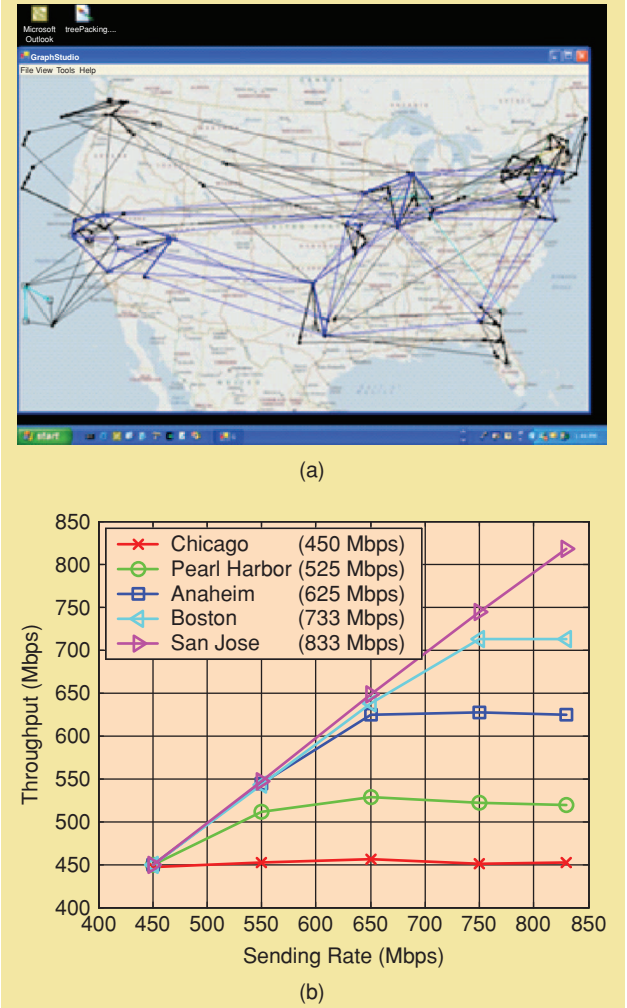
FILE DOWNLOAD

One of the most common network communication tasks is downloading a file from a server to a client computer. Traditionally, the downloaded file is unicast from the server to the client, but if delay is ignored, this can be viewed as a multicast of the file from the server to a large collection of clients using a large amount of buffering. From the multicast point of view, it can be seen that network coding can potentially increase the throughput and hence reduce the average download time.

To see how this can be done, consider downloading the file over a P2P network formed from all the nodes that are currently downloading the file. Newly arriving nodes join the network by connecting to a subset of the existing nodes. The original and still most popular protocol for P2P file downloading is BitTorrent [17]. In BitTorrent, the file is divided evenly into h pieces. Each node negotiates to obtain pieces of the file from its neighbors, until the node obtains all h pieces and can depart the network. After a node obtains a new piece, it announces the acquisition to its neighbors, so that every node knows which pieces every neighbor has. When requesting a particular piece from a neighbor, a node typically requests a local rarest piece, that is, a piece that is least common among all of the node's neighbors. This ensures that pieces are propagated approximately uniformly through the network, avoiding information bottlenecks.

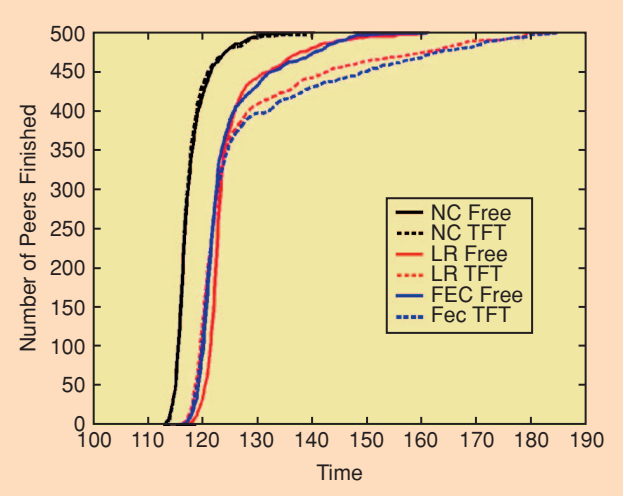
A new protocol for P2P file downloading based on network coding is Avalanche [18]. In Avalanche, the file is again divided evenly into h pieces. This time, the pieces $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_h$ are regarded as vectors of elements over a finite field. Instead of transmitting uncoded pieces to its neighbors, a node transmits coded pieces, where each coded piece $\boldsymbol{y}(e)$ is a random linear combination $\sum_{e'} \beta_{e'}(e)\boldsymbol{y}(e')$ of the coded pieces $\boldsymbol{y}(e')$ already received by the node, and hence $\boldsymbol{y}(e)$ is ultimately some linear combination $\sum_{i=1}^{h} g_i(e)\boldsymbol{x}_i$ of the original, uncoded pieces. Each coded piece $\boldsymbol{y}(e)$ is tagged with its global encoding vector $\boldsymbol{g}(e) = [g_1(e), \ldots, g_h(e)]$. Thus when a node receives enough coded pieces with linearly independent global encoding vectors, it can reconstruct the original file, and can depart the network.

At any given time, every node in the P2P network is a receiver. Hence the problem is actually a broadcast problem. In this case,

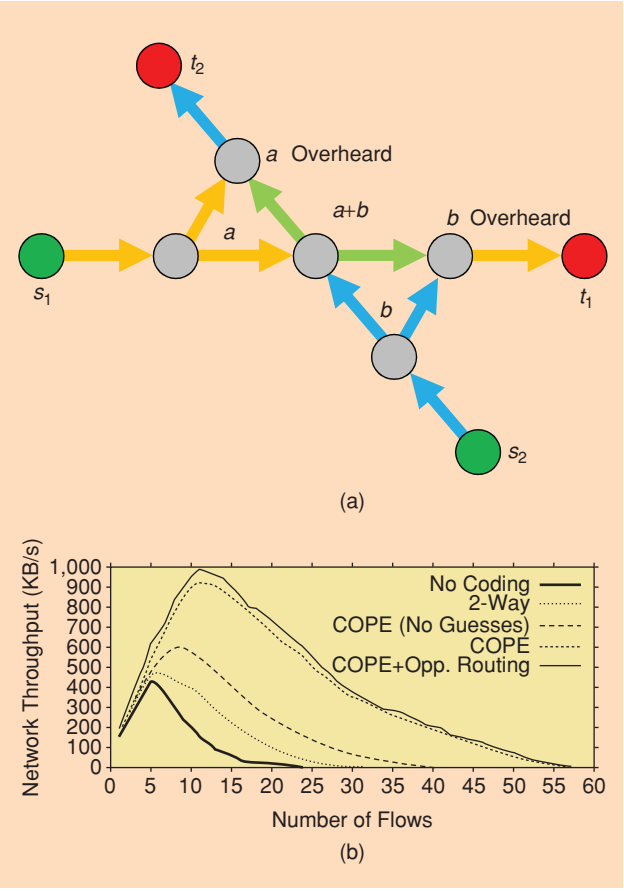


[FIG9] (a) Sprint Network. (b) Received rank as a function of sending rate.

theoretically, network coding provides no advantage in throughput over routing, since there exists an optimal set of edge-disjoint multicast trees over which routing can be performed at the broadcast capacity. In BitTorrent, the set of multicast trees is induced by the paths that are followed by the pieces as they are distributed from the sender to the receivers. The difficulty is finding an optimal set of multicast trees in a distributed way, especially when the network topology is changing underneath.



[FIG10] **Network Coding vs. Local Rarest vs. FEC. Courtesy of Gkantsidis and Rodriguez.**



[FIG11] **(a) Opportunistic network coding in a wireless mesh. (b) Advantage of network coding over routing. Courtesy of Katti et al.**

Gkantsidis et al. show in Figure 10 that network coding can outperform a BitTorrent-like local rarest approach by 10–30% [19]. In the figure, network coding (NC) is compared to local rarest (LR) and forward error correction (FEC) schemes. (In the FEC scheme, forward error correction is applied to the original uncoded pieces to obtain a large number of coded pieces, which are then distributed through the network using LR.) The figure also compares free and tit-for-tat (TFT) versions of each scheme. In the TFT versions, the difference between the number of pieces sent and received by any node is bounded by two. Similar conclusions were found in [20].

DISTRIBUTED STORAGE

A file can be reliably stored in a distributed way across a collection of unreliable nodes, if there is sufficient redundancy. Using a Reed-Solomon code, for example, a file of size M can be partitioned into k pieces $x_1, \ldots, x_k$ each of size M/k, coded into $n > k$ pieces $y_1, \ldots, y_n$ each of size M/k, and distributed across n storage nodes, such that the original file can be retrieved as long as at least k nodes are on line, by reading a coded piece of size M/k from each of k nodes and decoding. Unfortunately, maintaining this level of reliability in the face of node churn can be a challenge. Specifically, whenever a node fails permanently, it must be replaced by a new node. The new node must regenerate a coded piece of size M/k by reading a coded piece of size M/k from each of k other nodes, decoding, and re-encoding. This results in communicating essentially the entire file across the network for each node failure, which can be a very large amount of maintenance communication when n is large, nodes fail frequently, and files are not otherwise accessed frequently. Dimakis et al. have shown that if network coding is used, however, at the cost of a small increase in storage (by a factor $\beta < 2$), a new node can regenerate its coded data of size $\beta M/k$ by obtaining randomly re-encoded data of size $\beta M/k^2$ from each of k other nodes. This results in communicating essentially only a small fraction of the file (size $\beta M/k$) across the network for each node failure [21]. Other applications of network coding to distributed storage can be found in [22], [23].

WIRELESS MESH NETWORKS

Another convenient place to deploy network coding, besides an application-level overlay network, is a link-layer underlay network such as a wireless mesh network, on top of which an IP network can run transparently. In Figure 6, we have already seen how the number of transmissions required for two wireless nodes to exchange packets through an intermediary can be reduced from four to three using network coding. In fact this can be extended to multiple hops. If a wireless node s sends a stream of packets to a wireless node t over a long series of hops, then t can send an equal-rate stream of packets in the reverse direction to s for free, i.e., without any additional transmissions on the intermediate hops, for a savings approaching 2:1. Wu et al., who invented the technique, gave the name physical piggy-

backing to using XORs to combine packets at a wireless node for local decoding by neighbors with side information [24].

Katti et al. extended the work of Wu et al. to the case where the side information is obtained by overhearing [25]. Figure 11(a) illustrates a stream of packets a transmitted from s_1 to t_1 and another stream of packets b being transmitted from s_2 to t_2, along paths that cross at a central wireless node. Neighbors of the central node overhear packets a and b as they are transmitted to the central node as indicated. Thus the central node can code both packets a and b into a single packet $a + b$, which can be decoded immediately by both neighbors using the overheard packets as side information.

Katti et al. propose opportunistic coding (COPE), in which each node maintains a queue of received uncoded packets $p_1, p_2, \ldots, p_n, \ldots$ destined (according to their packet headers and the node's routing table) for next hop recipients $r_1, r_2, \ldots, r_n, \ldots$. The node then pops the packet p_1 and steps through the queue to greedily add packets for mixing, while ensuring that all of the next hop recipients can immediately decode the resulting mixture. A recipient can immediate decode a mixture packet if it knows all but one uncoded packet. For example, a mixture packet $p_1 + p_3 + p_4$ is valid if node r_1 knows p_3 and p_4, node r_3 knows p_1 and p_4, and node r_4 knows p_1 and p_3.

Figure 11(b) shows total network throughput as a function of the number of sender/receiver pairs in a simulation of 100 WiFi nodes randomly placed in a 800m $\times$ 800m area, transmitting UDP packets as fast as the MAC allows, for no coding and variations of network coding. It can be seen that when there are only a few flows, coding opportunities are limited, but when communication becomes dense, opportunistic network coding can increase the total throughput by a factor of three or more, completely transparently to the IP networking layer [25].

CONCLUSION

Network coding is a new tool of both theoretical and practical importance. To read further, consult the references herein, or the tutorials by Yeung et al. [26] and Fragouli et al. [27]. For a complete version of this tutorial paper, which includes a discussion of some of the more advanced topics in network coding, please see the technical report [28].

AUTHORS

Philip A. Chou (pachou@microsoft.com) received the Ph.D. degree from Stanford University in 1988. Currently he is a principal researcher at Microsoft Corporation (Redmond, WA, USA), affiliate professor at the University of Washington, and adjunct professor at the Chinese University of Hong Kong. His research interests include data compression, communications, and pattern recognition, with applications to video, images, audio, speech, and documents. He is a Fellow of the IEEE.

Yunnan Wu (yunnanwu@microsoft.com) received the Ph.D. degree from Princeton University in January 2006. Since August 2005, he has been a Researcher at Microsoft Corporation (Redmond, WA, USA). His research interests include networking, graph theory, information theory, and wireless communications. He received Student Best Paper awards at the 2000 SPIE VCIP and 2005 IEEE ICASSP conferences. He was awarded a Microsoft Research Graduate Fellowship for 2003–2005.

REFERENCES

[1] K. Menger, "Zur allgemeinen Kurventheorie," *Fund. Math.*, vol. 10, pp. 95–115, 1927.

[2] L.R. Ford and D.R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton Univ. Press, 1962.

[3] P. Elias, A. Feinstein, and C.E. Shannon, "A note on the maximum flow through a network," *IRE Trans. Inform. Theory*, vol. 2, pp. 117–119, 1956.

[4] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*. New York: Springer-Verlag, 2003.

[5] J. Edmonds, "Edge-disjoint branchings," in *Combinatorial Algorithms*, R. Rustin, ed., New York: Academic, 1973, pp. 91–96.

[6] K. Jain, M. Mahdian, and M.R. Salavatipour, "*Packing steiner trees*," in *Proc. 14th Symp. Discrete Algorithms*, ACM-SIAM, 2003.

[7] R. Ahlswede, N. Cai, S.-Y.R. Li, and R.W. Yeung, "Network information flow," *IEEE Trans. Inform. Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.

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

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

[10] 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. Inform. Theory*, vol. 51, pp. 1973–1982, June 2005.

[11] E. Erez and M. Feder, "Convolutional network codes for cyclic networks," in *Proc. 1st Workshop Network Coding, Theory, and Appl. (NetCod)*, Riva del Garda, Italy, Apr. 2005.

[12] Y. Wu, P.A. Chou, and S.-Y. Kung, "Minimum-energy multicast in mobile ad hoc networks using network coding," *IEEE Trans. Commun.*, vol. 53, no. 11, pp. 1906–1918, Nov. 2005, also presented at the Information Theory Workshop, San Antonio, TX, Oct., 2004.

[13] T. Ho, M. Médard, R. Koetter, D.R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Trans. Inform. Theory*, vol. 52, no. 10, pp. 4413–4430, Oct. 2006.

[14] P. Sander, S. Egner, and L. Tolhuizen, "Polynomial time algorithms for network information flow," in *Proc. Symp. Parallel Algorithms and Architectures ACM,* San Diego, CA, June 2003, pp. 286–294.

[15] P.A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. 41st Allerton Conf. Commun., Control and Comput.*, Monticello, IL, Oct. 2003.

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

[17] B. Cohen, "Incentives build robustness in bittorrent," in *Proc. 1st Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, June 2003. [Online]. Availble: http://www.bittorrent.com/

[18] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *Proc. INFOCOM*, Miami, FL, Mar. 2005, pp. 136–146.

[19] C. Gkantsidis, J. Miller, and P. Rodriguez, "Anatomy of a p2p content distribution system with network coding," in *Proc. 5th Int. Workshop Peer-to-Peer Syst.*, Santa Barbara, CA, Feb. 2006.

[20] G. Ma, Y. Xu, M. Lin, and Y. Xuan, "A content distribution system based on sparse network coding," in *Proc. 3rd Workshop Network Coding, Theory, and Applications*, San Diego, 2007.

[21] A.G. Dimakis, P.G. Godfrey, M.J. Wainwright, and K. Ramchandran, "Network coding for peer-to-peer storage," in *Proc. INFOCOM*, Anchorage, Alaska, 2007.

[22] S. Acedanski, S. Deb, M. Médard, and R. Koetter, "How good is random linear coding based distributed networked storage?" in *Proc. 1st Workshop Network Coding, Theory, and Applications*, Riva del Garda, Italy, Apr. 2005.

[23] A.G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Decentralized erasure codes for distributed networked storage," *IEEE Trans. Inform. Theory*, June 2006.

[24] Y. Wu, P.A. Chou, and S.-Y. Kung, "Information exchange in wireless networks with network coding and physical-layer broadcast," in *Proc. 39th Ann. Conf. Inform. Sci. Syst.*, Baltimore, MD, Mar. 2005, [Online]. Available: http://research.microsoft.com/~yunnanwu. Also available as Microsoft Research Tech. Rep. MSR-TR-2004-78, Aug. 2004.

[25] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "XORs in the air: Practical wireless network coding," in ACM *SIGCOMM*, Pisa, Italy, Sept. 2006.

[26] R.W. Yeung, "Network coding theory," *Found. Trends Commun. Inform. Theory*, vol. 2, pp. 241–381, 2005.

[27] C. Fragouli, J.-Y.L. Boudec, and J. Widmer, "Network coding: An instant primer," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 63–68, Jan. 2006.

[28] P.A. Chou and Y. Wu, "Network coding for the internet and wireless networks," Microsoft Research, Tech. Rep. MSR-TR-2007-70, June 2007. SP