Network Coding for the Internet and Wireless Networks

Philip A. Chou and Yunnan Wu

June 2007

MSR-TR-2007-70

Microsoft Research
One Microsoft Way,
Redmond, WA, 98052

Network Coding for the Internet and Wireless Networks

Philip A. Chou and Yunnan Wu

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. Therefore, in network coding, 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. Thus, network coding is generally the transmission, mixing (or encoding), and re-mixing (or re-encoding) of messages arriving at nodes *inside* the network, such that the transmitted messages can be unmixed (or decoded) at their final destinations.

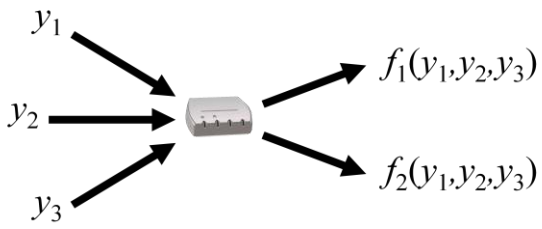

Figure 1. Network Coding: Network nodes can compute functions of input messages.

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 bits per second, arrive at a node, contending for an output link, having capacity B bits per second. 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 bits per second. 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 $C \to D$.

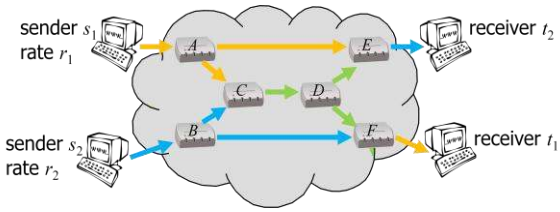

Figure 2. Two unicast sessions contending for a bottleneck link.

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 $\{(r_1, r_2)\colon\ 0 \le r_1, 0 \le r_2, r_1 + r_2 \le B\}$, which is shown in Figure 3 (left). 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 $\{(r_1, r_2)\colon\ 0 \le r_1 \le B, 0 \le r_2 \le B\}$, which is shown in Figure 3 (right). 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.

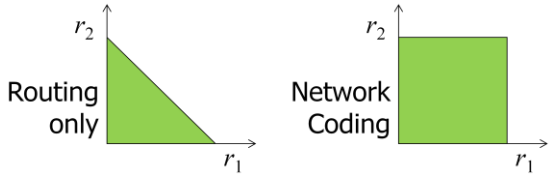

Figure 3. Achievable communication rate regions for routing only and network coding.

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 (right) 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), …, (s_N, T_N) with respective communication rates r_1, …, 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. We shall have more to say about this later.

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.

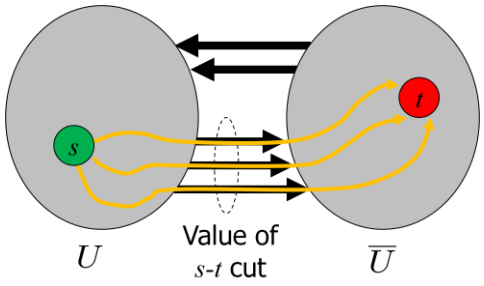

Figure 4. Upper bound on unicast capacity.

The minimum of the values of all such s–t cuts, herein designated $MinCut(s, t)$, is clearly also an upper bound on $r(s, t)$. That is, $r(s, t) \leq MinCut(s, t)$. 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 $h = MinCut(s, t)$ 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, $r(s, t) = MinCut(s, t)$. 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, $r(s, V) \leq \min_{v \in V} MinCut(s, v)$. It turns out that the upper bound, $\min_{v \in V} MinCut(s, v)$, 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 $\min_{v \in V} MinCut(s, v)$ (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, $r(s, V) = \min_{v \in V} MinCut(s, v)$. Thus, $C = \min_{v \in V} MinCut(s, v)$ 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, $MinCut(s, t)$ remains an upper bound on this rate for any $t \in T$. Thus, $r(s, T) \leq \min_{t \in T} MinCut(s, t)$. 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, $\min_{t \in T} MinCut(s, t) = 2$. 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. Thus routing along a maximal set of edge-disjoint Steiner trees cannot in general achieve throughput equal to the upper bound

$\min_{t \in T} MinCut(s,t)$. [1] Worse, finding such a maximal set of edge-disjoint Steiner trees turns out to be NP-hard (6).

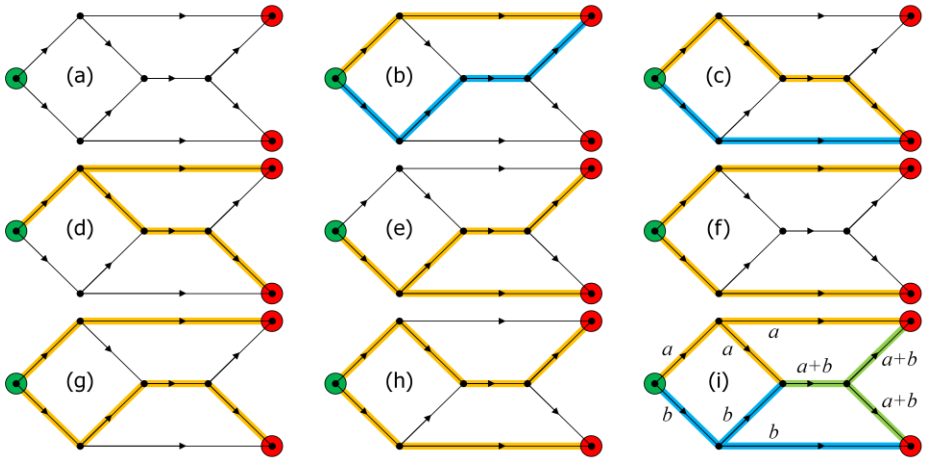

Figure 5. One Multicast Session. (a) Sender s in green, receivers $T = \{t_1, t_2\}$ in red. (b,c) Maximal sets of edge-disjoint paths from s to t_1 and t_2, demonstrating $\min_{t \in T} MinCut(s,t) = 2$. (d)-(h) All five possible multicast trees from s to T, no two of which are edge-disjoint. (i) Network coding achieves the multicast capacity.

Alswede et al.'s Theorem, 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, $\min_{t \in T} MinCut(s,t)$, can always be achieved in any network using network coding, as proved in 2000 in the seminal work of Alswede, Cai, Li, and Yeung (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, $h = \min_{t \in T} MinCut(s,t)$ 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, Yeung, and Cai (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

[1] 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.

operations. Jaggi, Sanders, 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 (left).) However, if network coding is permitted, then only 4.5 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 (right).) 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.

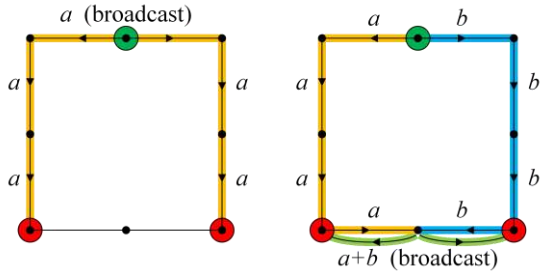

Figure 6. Network coding minimizes energy per packet. Sender s_1 is in green; receivers t_1 and t_2 are in red.

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 (left) 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 (right) 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.

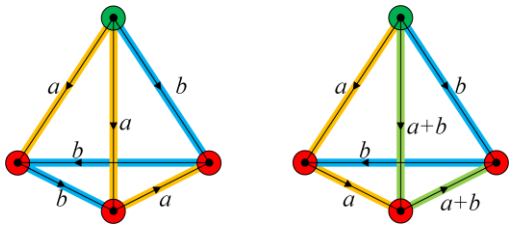

Figure 7. Network coding minimizes delay.

Applicability to Real Networks: Theory vs. Practice

Network Coding is presumably highly applicable to communication in real networks – the primary example being the Internet – both at the IP layer, for example in routers in ISPs, and at the application layer, both in dedicated infrastructure such as content distribution networks and in ad hoc networks such as peer-to-peer networks. But network coding should presumably also have applications to wireless ad hoc multihop networks, sensor networks, stationary wireless residential or community mesh networks, and so forth. In the following section, we address how practical network coding might be done in real networks, where packets are subject to random loss and delay; edges have variable capacities due to congestion and other cross traffic; node and link failures, as well as additions and deletion, are common, as they might be in peer-to-peer and ad hoc networks; cycles are everywhere; the multicast capacity may be unknown; and there is no centralized knowledge of the graph topology or the encoder or decoder functions.

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 allow 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 $h = MinCut(s, T)$ 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, $y(e) = \sum_{e'} \beta_{e'}(e) y(e')$. The coefficients of the linear combination form a vector $\boldsymbol{\beta}(e) = [\beta_{e'}(e)]$, 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 $\boldsymbol{g}(e) = [g_1(e), .., g_h(e)]$, 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 $\boldsymbol{\beta}(e)$.

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

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

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 $1 - |E|/|F| \cong 0.996$. 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 1400 bytes 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 1400 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 $\boldsymbol{y}(e')$ on the preceding edges or, alternatively, as a linear combination of the source packets $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_h$, i.e., $\boldsymbol{y}(e) = \sum_{e'} \beta_{e'}(e)\boldsymbol{y}(e') = \sum_{i=1}^{h} g_i(e)\boldsymbol{x}_i$. As before,

$$\begin{bmatrix} \boldsymbol{y}(e_1) \\ \vdots \\ \boldsymbol{y}(e_h) \end{bmatrix} = \begin{bmatrix} g_1(e_1) & \cdots & g_h(e_1) \\ \vdots & \ddots & \vdots \\ g_1(e_h) & \cdots & g_h(e_h) \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_h \end{bmatrix} = G_t \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_h \end{bmatrix},$$

so 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 $\boldsymbol{g}(e)$ associated with the edge. This can be easily accomplished, for example, by prefixing the ith information vector $\boldsymbol{x}_i$ with the ith unit vector $\boldsymbol{u}_i$ 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 $[\boldsymbol{g}(e), \boldsymbol{y}(e)] = \sum_{e'} \beta_{e'}(e)[\boldsymbol{g}(e'), \boldsymbol{y}(e')] = \sum_{i=1}^{h} g_i(e)[\boldsymbol{u}_i, \boldsymbol{x}_i]$.

The cost of the tag $\boldsymbol{g}(e)$ 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 $\boldsymbol{y}(e_1), \ldots, \boldsymbol{y}(e_h)$ 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.

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 $\boldsymbol{x}_{kh+1}, \ldots, \boldsymbol{x}_{kh+h}$ 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.

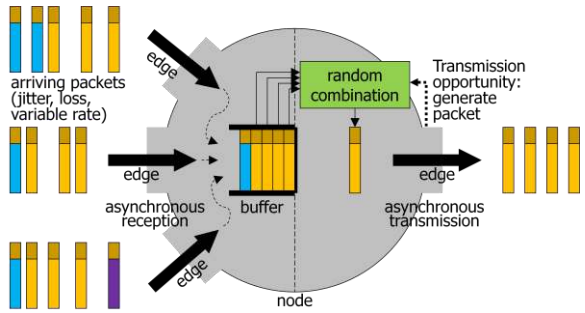

Figure 8. Buffering at a node.

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, as illustrated in Figure 9. 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.

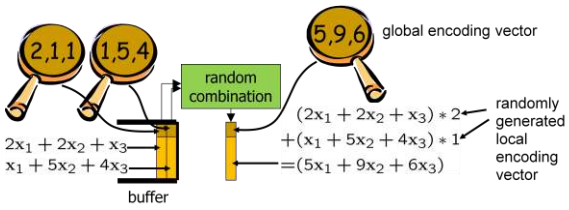

Figure 9. Forming an outgoing packet.

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 the block decoding.

Routing Example

The above techniques for practical network coding were simulated in (15) on the network shown in Figure 10, 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).

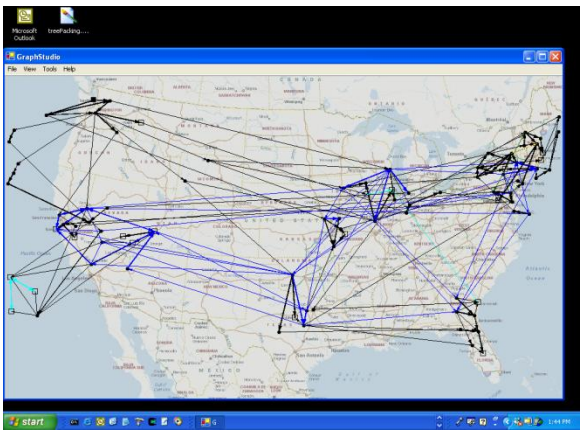

Figure 10. Sprint network.

For field size $|F| = 2^{16}$ and generation size $h = 100$, Figure 11 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 Mbps), 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 $\min_{t \in T} MinCut(s,t)$, it is possible in practice to achieve reliable multicast very close to capacity. For further details, see (15).

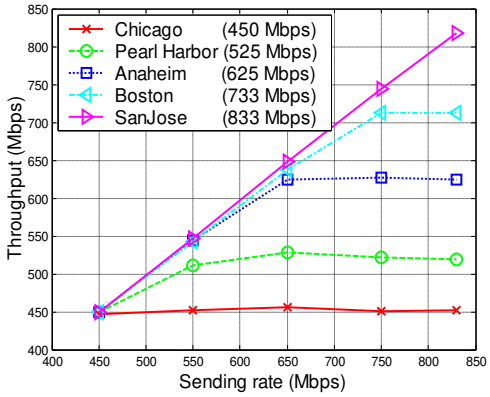

Figure 11. Received rank as a function of sending rate.

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[2]. 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, whether the file is a web page, a picture, a music track, a movie, a program, or other kind of document. 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

[2] 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.

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 $[g_1(e), \ldots, g_h(e)]$. 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, 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.

Gkantsidis et al. show in Figure 12 that network coding can outperform a BitTorrent-like local rarest approach by 10-30% (19)[3]. Similar conclusions were found in (20).

[3] 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.

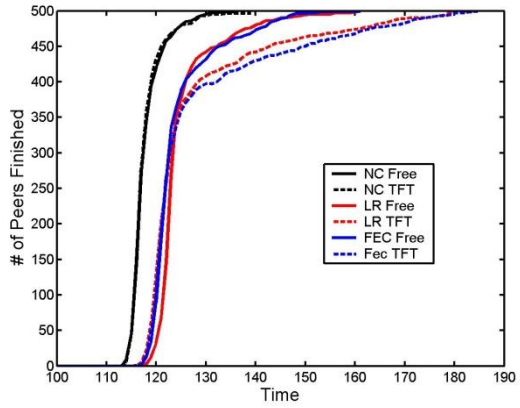

Figure 12. Network Coding vs. Local Rarest vs. FEC. Courtesy of Gkantsidis and Rodriguez.

Video on Demand, Live Media Broadcast, Game Spectating, and Instant Messaging

Video on demand is essentially a file download in which the pieces of the downloaded file must arrive in order and be decoded in real time, after some small delay. Network coding can be applied in this case by breaking the file into generations, which can be downloaded sequentially. A similar technique can be applied, or course, to *live media broadcast*. Earliest decoding can be used to further reduce delay. Similar to live media broadcast is *game spectating*, in which spectators can watch other gamers play. Instant messaging (IM) is also similar to live broadcast, but with typically low bit rate (and bursty) text messages and a softer delay constraint. However, IM is increasingly including larger messages such as images and audio clips. Flooding, which is usually used for IM in P2P networks, is inefficient when used on larger files. Network coding can be applied in all of these cases to improve efficiency in overlay networks.

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 $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ each of size M/k, coded into $n > k$ pieces $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ 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 piggybacking* 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 13 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.

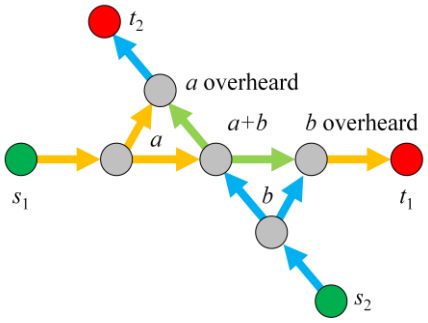

Figure 13. Opportunistic network coding in a wireless mesh.

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 14 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 x 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. For more information, see (25).

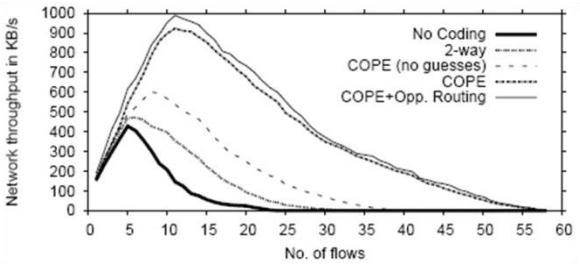

Figure 14. Advantage of network coding over routing. Courtesy of Katti et al.

Sensor Networks

As a final application of network coding, consider the visionary work of Petrović, Ramchandran, and Rabaey (26), in which sensors are conceived to be so small that they can be mixed into cans of paint, can be spread onto surfaces, and can harvest energy from the environment, to form sensing surfaces over a built-in communication backplane. To reduce each node's size and energy requirements, the narrowband radio's off-chip quartz oscillator is replaced by an on-chip LC circuit. However, the center frequency of the LC circuit is random, subject to the statistical variations of the manufacturing process. This means, essentially, that each node picks a random channel on which to transmit, and another random channel on which to receive. Petrović et al. show, using a random graph construction, that the throughput between any two nodes is constant if only routing is used, but grows linearly in the number of channels if network coding is used, and the radio ranges are optimally chosen. The reason network coding helps greatly here is that the randomly mixed packets can implicitly find their way towards the destination, whereas explicitly identifying routes is difficult when the graph connectivity is unknown. See (26) for details.

Advanced topics

Security

When applying network coding in practice, we need to be careful about potential security threats. One form of attack, called a *pollution attack*, is particularly relevant when using the practical network coding techniques mentioned earlier. Consider a malicious node injecting a junk packet into the network. Although this might also occur in a traditional system without network coding, its effect is far more serious with network coding. If the junk packet is mixed into the buffer of a node, the buffer will be polluted, the output of the node will become junk, and this may soon propagate to the entire network. Fortunately, recent studies have proposed

several counter-measures, including a homomorphic hash scheme (27), a homomorphic signature scheme (28), and a secure random checksum scheme (27). All of these techniques try to detect a polluted packet before it gets mixed into the buffer.

Resource Optimization

In practical network coding, generating an output packet is easy: simply combine the buffered packets using a set of randomly generated coefficients. But how fast should output packets be generated? If the network resources follow a "use it or lose it" model, then output packets can be generated as fast as possible. However, for typical networking scenarios, this is not the case. Hence the rate at which mixture packets are generated must be properly controlled. This problem can be formulated as a mathematical optimization, where the variables are the allocated link rates and the sending rate at the source. Such an optimization can be viewed as a generalization of classical network flow optimization for routing in transport and information networks. In network flow optimization for routing, the key notion is a flow, which characterizes the resources needed for unicast; in network flow optimization for coding, the key notion is a (link-wise) maximum of flows, which characterizes the resources needed for network coding-based multicast. Several recent studies propose centralized and distributed, primal and dual algorithms for the optimization; see, e.g., (29)(30)(31)(32)(33), and the references therein.

Multi-session Network Coding

Single-session network coding is well understood in both theory and practice. In contrast, the general problem of multi-session network coding, where multiple communicating sessions with independent data share a network of lossless links with rate constraints, has proven to be extremely challenging. denote theThere are many examples that indicate the intricacy of the problem. For instance, a rather surprising result is that linear coding is generally insufficient to achieve all points in the capacity region (34). Why is multi-session network coding so hard? In single-session network coding, every receiver wants the same sources of information; hence it is not too surprising that randomly mixing the packets in the network is the best thing to do. In multi-session network coding, however, if packets are randomly mixed in the network, a receiver may receive many coded packets but still cannot decode them since they are "polluted" by unwanted sources. In general, mixing more packets together increases the diversity of the mixture, as it may potentially be useful to more receivers; however, that makes it difficult for receivers to clean out the pollution from a mixture packet. Intuitively, it is the need for careful mixing and demixing that makes this problem hard. For some progress on constructive multi-session network coding, see, e.g., (35) and the references therein.

Despite the difficulty in arriving at concrete constructive schemes, the capacity region for multi-session network coding in *acyclic* graphs has been characterized using information theoretic techniques. This characterization is based on a notion called *entropy space*, introduced by Raymond Yeung. Let N be a set of n random variables. Let H_n denote the $(2^n - 1)$-dimensional Euclidean space where the coordinates $\{h_A : \emptyset \subset A \subseteq N\}$ correspond to the nonempty subsets of N. A vector $\boldsymbol{h} \in H_n$ is called *entropic* if for a certain joint distribution of random variables in N, h_A is equal to the joint entropy of the random variables in A, for every A. Thus an entropic

vector $\boldsymbol{h}$ describes a possible vector value taken by the $(2^n - 1)$ joint entropies of the n random variables. The set of all such entropic vectors, denoted by Γ_n^*, is called the entropy space. Based on the entropy space, an inner bound and an outer bound of the capacity region for multi-session network coding in acyclic graphs are presented in (36). We now explain the outer bound briefly. Let N specifically refer to the set of n random variables comprising one random variable for each source and one random variable for each edge. The outer bound is essentially characterized via a set of linear constraints on $\boldsymbol{h} \in \overline{\Gamma_n^*}$, where $\overline{\Gamma_n^*}$ refers to the closure of $\overline{\Gamma_n^*}$. Intuitively, the set of linear constraints express: (i) the joint entropy of the sources equals the sum of the individual entropies (i.e., the sources are independent), (ii) the conditional entropy of each edge variable given its predecessor edges is zero (i.e., each edge is generated by its predecessors), (iii) at each receiver, the conditional entropy of its needed sources given its incoming edges is zero (i.e., the receiver indeed can recover the sources it needs), (iv) the entropy of each edge variable cannot exceed the edge capacity, and (v) the source rate cannot exceed the entropy of the source variable. More recently, the gap between the bounds is closed in (37) with a careful bounding of the constrained regions in the entropy space. To prove the achievability of the region, random coding techniques similar in flavor to what Shannon used in his papers are used to show the existence of solutions. For details, please refer to (36)(37).

Joint Network Coding and Distributed Source Coding

Distributed source coding refers to the problem of separate compression of correlated sources. For example, suppose two correlated newspapers, X_1 and X_2, are available at two locations. We want to maximally compress them into $f(X_1)$ and $g(X_2)$, while ensuring that a receiver can recover X_1 and X_2 from $f(X_1)$ and $g(X_2)$. This problem was answered in 1972 by Slepian and Wolf (38), who characterized the fundamental limit for distributed lossless compression. Recently, progress has been made regarding practical design of codes approaching the limit; please see (39) for an excellent tutorial.

Now consider a generalization of this problem, where multiple correlated sources are to be multicast to multiple receivers over a network of lossless links with bit-rate constraints. This problem is also a generalization of the single-session network coding problem. The fundamental limit for this problem is recently found by Song and Yeung (40) and Ho et al. (41). It turns out that random linear mixing is also optimal for this generalized problem.

However, random linear mixing does not immediately lead to a low-complexity solution here. Whereas in single-session network coding, a decoder solves the unknowns from an equal number of independent linear equations, in the generalized problem a decoder faces more unknowns than equations. The optimal decoder maximizes the *a posteriori* probability of the unknowns given the observed linear equations and the correlation statistics. If unstructured random linear mixing is used, the decoding complexity can be too high to make the scheme practical. A natural thought is then whether we can induce some structure to reduce the complexity. For some progress along this line, please see (42) and the references therein.

Joint Network Coding and Channel Coding

In Figure 6, we saw that network coding can make packet exchange more efficient in wireless networks. The gist is that if a node A has a, a node B has b, and intermediate node C has both a and b, then C can broadcast a XOR b to A and B, simultaneously transferring b to A and a to B. Now suppose that the rates of the wireless links from C to A and B are 3 bps and 2 bps, respectively. Using the earlier technique, we can send a XOR b at rate 2 bps. Interestingly, with *joint network coding and channel coding*, a better result is possible: a 2-bit packet $a = [a_1, a_2]$ and a 3-bit packet $b = [b_1, b_2, b_3]$ can be delivered to their intended receivers in one second. Here is how to do this. As illustrated in Figure 15, we use 8-PSK to send three bits m_1, m_2, m_3. The first two bits are the XOR of a with the first two bits of b; the third bit is b_3. Node A has a better channel, with rate 3 bps; we assume it is able to recover all three transmitted bits and hence recover b. Node B already knows the last bit. Hence to node B the transmitted symbol seems to be from a 4-point constellation, and it can resolve the ambiguity. We conclude from this "appetizer" example that jointly performing network coding and channel coding can achieve higher rates. For more information, please see (43)(44).

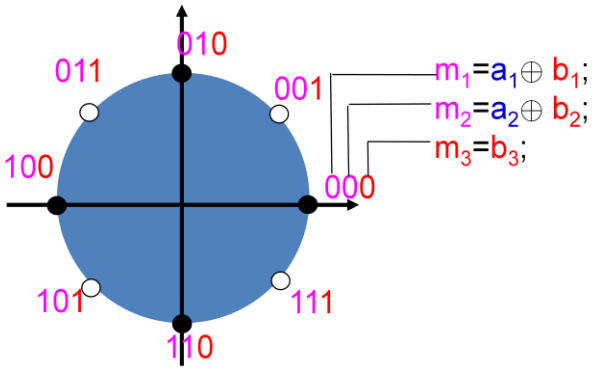

Figure 15. Joint network coding and channel coding.

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. (45) and Fragouli et al. (46).

Authors

Philip A. Chou (pachou@microsoft.com) received the PhD 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 PhD 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. *Zur allgemeinen Kurventheorie.* **Menger.** 1927, Fund. Math., Vol. 10, pp. 95--115.

2. **Ford, L. R. and Fulkerson, D. R.** *Flows in Networks.* Princeton : Princeton University Press, 1962.

3. *Note on Maximum Flow through a Network.* **Elias, P., Feinstein, A. and Shannon, C. E.** s.l. : IRE, 1956, Trans. Information Theory, Vol. 2.

4. **Schrijver, A.** *Combinatorial Optimization: Polyhedra and Efficiency.* s.l. : Springer-Verlag, 2003.

5. **Edmonds, J.** Edge-Disjoint Branchings. [ed.] R. Rustin. *Combinatorial Algorithms.* New York : Academic Press, 1973.

6. *Packing Steiner Trees.* **Jain, K., Mahdian, M. and Salavatipour, M. R.** Baltimore : ACM-SIAM, January 2003. Symp. on Discrete Algorithms (SODA).

7. *Network Information Flow.* **Alswede, R., et al.** s.l. : IEEE, 2000, Trans. Information Theory, Vol. 46.

8. *An Algebraic Approach to Network Coding.* **Koetter, R. and Médard, M.** 5, s.l. : IEEE/ACM, October 2003, Trans. Networking, Vol. 11, pp. 782-795.

9. *Linear Network Coding.* **Li, S.-Y. R., Yeung, R. W. and Cai, N.** s.l. : IEEE, 2003, Trans. Information Theory, Vol. 49, pp. 371-381.

10. *Polynomial time algorithms for multicast network code construction.* **Jaggi, S., et al.** 6, s.l. : IEEE, 2005, Trans. Information Theory, Vol. 51, pp. 1973-1982.

11. *Convolutional Network Codes for Cyclic Networks.* **Erez, E. and Feder, M.** Riva del Garda : s.n., April 2005. NetCod.

12. *Minimum-energy multicast in mobile ad hoc networks using network coding.* **Wu, Y., Chou, P. A. and Kung, S.-Y.** 11, s.l. : IEEE, November 2005, Trans. Communications, Vol. 53.

13. *A Random Linear Network Coding Approach to Multicast.* **Ho, T., et al.** 10, October 2006, IEEE Trans. Information Theory, Vol. 52.

14. *Polynomial Time Algorithms for Network Information Flow.* **Sanders, P., Egner, S. and Tolhuizen, L.** San Diego : ACM, June 2003. Symp. Parallelism in Algorithms and Architectures (SPAA).

15. *Practical Network Coding.* **Chou, P. A., Wu, Y. and Jain, K.** Allerton : s.n., October 2003. Conference on Communication, Control, and Computing.

16. *Measuring ISP topologies with Rocketfuel.* **Spring, N., Mahajan, R. and Wetherall, D.** Pittsburg, PA : s.n., 2002. ACM SIGCOMM.

17. **Cohen, B.** Incentives Build Robustness in BitTorrent. [Online] May 2003. http://bitconjurer.org/BitTorrent/bittorrentecon.pdf.

18. *Network Coding for Large Scale Content Distribution.* **Gkandsidis, C and Rodruiguez, P.** Miami : IEEE, March, 2005. INFOCOM.

19. *Comprehensive View of a Live Network Coding P2P System.* **Gkantsidis, G., Miller, J. and Rodriguez, P.** Rio de Janeiro : ACM SIGCOMM/USENIX, 2006. IMC.

20. *A Content Distribution System Based on Sparse Network Coding.* **Ma, G., et al.** San Diego : s.n., 2007. NetCod.

21. *Network Coding for Peer-to-Peer Storage.* **Dimakis, A. G., et al.** Anchorage : IEEE, May 2007. INFOCOM.

22. *How Good is Random Linear Coding Based Distributed Networked Storage?* **Acedanski, S., et al.** Riva del Garda : s.n., April 2005. NetCod.

23. *Decentralized Erasure Codes for Distributed Networked Storage.* **Dimakis, A. G., Prabhakaran, V. and Ramchandran, K.** s.l. : IEEE, June 2006, Trans. Information Theory.

24. *Information Exchange in Wireless Networks with Network Coding and Physical-Layer Broadcast.* **Wu, Y., Chou, P. A. and Kung, S.-Y.** Baltimore : s.n., March 2005. CISS.

25. *XORs in the Air: Practical Wireless Network Coding.* **Katti, S., et al.** Pisa : ACM, September 2006. SIGCOMM.

26. *Overcoming Untuned Radios in Wireless Networks with Network Coding.* **Petrović, D., Ramchandran, K. and Rabaey, J.** Riva del Garda : s.n., April 2005. NetCod.

27. *Cooperative Security for Network Coding File Distribution.* **Gkandsidis, C and Rodruiguez, P.** Barcelona, Spain : s.n., 2006. IEEE INFOCOM.

28. *Signatures for Network Coding.* **Charles, D., Jain, K. and Lauter, K.** Princeton : s.n., 2006. 40th Annual Conference on Information Sciences and Systems.

29. *Distributed utility maximization for network coding based multicasting: a shortest path approach.* **Wu, Y. and Kung, S.-Y.** 8, August 2006, IEEE Journal on Selected Areas in Communications, Vol. 24, pp. 1475--1488.

30. *Minimum-Cost Multicast Over Coded Packet Networks.* **Lun, D. S., et al.** 6, June 2006, IEEE Transactions on Information Theory, Vol. 52, pp. 2608--2623.

31. *Distributed utility maximization for network coding based multicasting: a critical cut approach.* **Wu, Y., Chiang, M. and Kung, S.-Y.** 2006. 2nd workshop on Network Coding, Theory, and Applications.

32. *Min-Cost Selfish Multicast with Network Coding.* **Bhadra, S, Shakkottai, S. and Gupta, P.** Novemeber 2006, IEEE Transactions on Information Theory, pp. 5077--5087.

33. *On Achieving Maximum Multicast Throughput in Undirected Networks.* **Li, Z., Li, B. and Lau, L. C.** 6, June 2006, IEEE Trans. Information Theory, Vol. 52, pp. 2467--2485.

34. *Linearity and Solvability in Multicast Networks.* **Dougherty, R., Freiling, C. and Zeger, K.** 10, 2004, IEEE Trans. Information Theory, Vol. 50, pp. 2243--2256.

35. *On constructive multi-source network coding.* **Wu, Y.** Seattle, WA : s.n., 2006. IEEE Int'l. Symp. Information Theory.

36. *Zero-error network coding for acylic network.* **Song, L. and Yeung, Y. W.** 12, Dec. 2003, IEEE Trans. on Information Theory, Vol. 49, pp. 3129--3139.

37. *The Capacity region for Multi-source Multi-sink Network Coding.* **Yan, X., Yeung, R. W. and Zhang, Z.** Nice, France : s.n., 2007. IEEE International Symposium on Information Theory.

38. *Noiseless coding of correlated information sources.* **Slepian, D. and Wolf, J. K.** July 1973, IEEE Trans. Information Theory, Vol. 19, pp. 471--480.

39. *Distributed source coding for sensor networks.* **Xiong, Z., Liveris, A. and Cheng, S.** September 2004, IEEE Signal Processing Magazine, Vol. 21, pp. 80--94.

40. *Network information flow -- multiple sources.* **Song, L. and Yeung, R. W.** 2001. IEEE Int'l Symp. Information Theory.

41. *Network Coding for Correlated Sources.* **Ho, T., et al.** Princeton : s.n., March 2004. Conf. Information Sciences and Systems (CISS).

42. *On practical design for joint distributed source and network coding.* **Wu, Y., et al.** Riva del Garda, Italy : s.n., 2005. 1st Workshop on Network Coding, Theory, and Applications (NETCOD).

43. *Nested codes with multiple interpretations.* **Xiao, L., et al.** Princeton, NJ : s.n., 2006. 40th Annual Conf. Information Science and Systems.

44. *Broadcasting when Receivers Know Some Messages A Priori.* **Wu, Y.** Nice, France : s.n., 2007. IEEE Int'l Symp. Information Theory.

45. **Yeung, R. W., et al.** Network coding theory. *Foundation and Trends in Communications and Information Theory.* 2005, Vol. 2, 4,5, pp. 241--381.

46. *Network Coding: An Instatant Primer.* **Fragouli, C., Le Boudec, J.-Y. L. and Widmer, J.** 1, January 2006, ACM SIGCOMM Computer Communication Review, Vol. 36.