Network Coding

K V Rashmi, Nihar B Shah and P Vijay Kumar



K V Rashmi (left) is currently an ME student at IISc. Her current research interests are in coding theory, error-correction in networks and wireless communication.

Nihar B Shah (right) is currently pursuing his masters in telecommunication at IISc. He is interested in the various facets of coding theory, graph theory and information theory.

P Vijay Kumar (bottom) is a Professor at IISc, Bangalore. His research interests include coding theory, low-correlation sequences and signal design for wireless communicaiton. He is a Fellow of the IEEE and a co-recipient of the 1995 IEEE Information Theory Paper Award.

Network coding is a technique to increase the amount of information flow in a network by making the key observation that information flow is fundamentally different from commodity flow. Whereas, under traditional methods of operation of data networks, intermediate nodes are restricted to simply forwarding their incoming message symbols downstream, network coding achieves vast performance gains by permitting intermediate nodes to carry out algebraic operations on the incoming data. In this article we present a tutorial introduction to network coding as well as an application to the efficient operation of distributed data-storage networks.

1. Introduction to Network Coding

Here is a classic puzzle: *A gang of five pirates find a treasure. The old chief of the pirates is on his death bed, and is worried about the misuse of the treasure by the other four pirates after his death. So, he dumps all the treasure in a chest and puts a number lock on it. Then he whispers something to each of the remaining four pirates. Now he is assured that no pirate alone can open the chest and take the treasure, and the chest can be opened only if any two of them together wanted to do so. What did he whisper into their ears?*

The solution to this puzzle is simple. He set the number key on the lock as an arbitrary number, say 81. He divided the number key on the lock into two parts, say $x = 8$ and $y = 1$. He revealed the number x to the first pirate, y to the second, $x + y$ to the third, and $x + 2y$ to the fourth, also revealing what combination of the two parts was revealed to each of them. In this way, any two

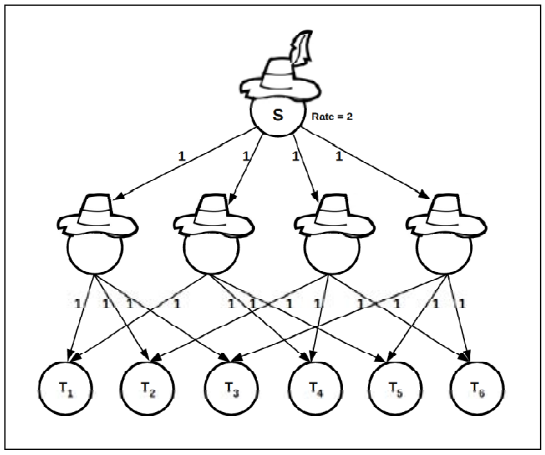


Figure 1. The pirates puzzle as a network coding problem.

pirates together can obtain the entire key, but any one alone cannot open the lock!

This problem can also be posed as a *Network Coding* problem. Construct a graph as shown in *Figure* 1. The top-most node is a source, representing the chief pirate. He chooses two numbers arbitrarily, which are the two parts of the key. We say that the source has a rate of 2 symbols per unit time. The four nodes in the next level are intermediate nodes, representing the four other pirates. Since the chief tells each of them one number, there are links from the source to each of these four nodes having capacities of 1 symbol per unit time. Thus, a single pirate will have just one of the two numbers and hence will not be able to open the lock. To enable any pair of pirates to obtain the key together, each pair of intermediate nodes is connected to a sink (labelled $T_1, \ldots, T_6$ in the figure), which demands all symbols produced by the source. Each such sink obtains the symbols from the two pirates it connects to; hence the edges have a capacity 1. Now, solving this network coding problem amounts to solving the puzzle.

Let us look at a second example called the *butterfly network* that highlights the main idea behind network coding. *Figure* 2a depicts a network with commodity flow. Node s is a *source* producing apples and strawberries.

Keywords
Information flow, multicast networks, max-flow min-cut, distributed storage.

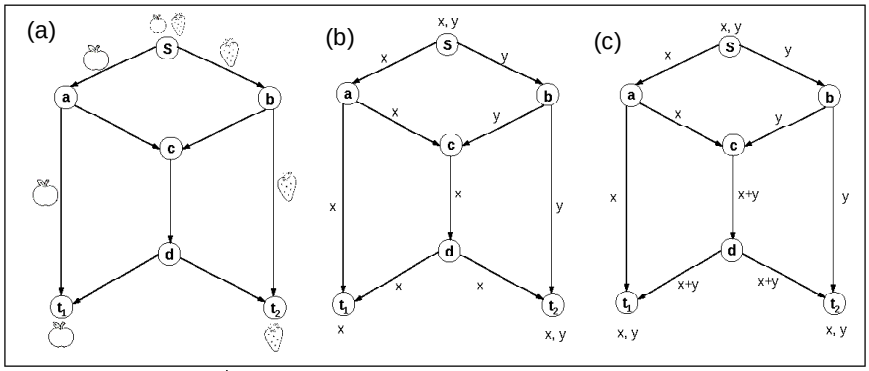


Figure 2. Comparison of commodity to information flow: (a) Commodity flow, (b) information flow with replication, and (c) information flow with replication and coding.

Nodes t_1 and t_2 are *sinks*, i.e., they are consumers for these commodities. Each edge in the network has the capacity to carry up to one fruit per unit time, and the goal is to provide the maximum number of commodities to the sinks per unit time. It is easy to see that at most one apple and one strawberry can be delivered to the consumers respectively per unit time.

Unlike commodity, information can be replicated and coded, and this is of great help when solving information flow networks. For example, *Figure* 2b shows the same network as in *Figure* 2a but with node s being an information source. Here, replication is allowed but not coding. Consequently, sink t_1 is able to get x whereas sink t_2 is able to get both the symbols x and y produced by the source.

Figure 2c depicts the case where both replication and coding are allowed. Node c passes a function of the incoming symbols on its outgoing link. This idea of intermediate nodes coding the incoming symbols before passing it on their outgoing links as introduced by Ahlswede *et al*. [1] is called network coding. Due to network coding, both sinks t_1 and t_2 are able to get both the symbols x and y simultaneously.

In this article we will present the elements of network coding on a basic class of networks for simplicity. The broad range of possible networks and their classifications are provided in *Box* 1.

Box 1. Various Classifications of Networks

Linear vs Non-linear: A network coding solution is said to be linear if all intermediate nodes take linear functions of the incoming symbols, else non-linear.

Multicast vs Non-multicast: Multicast networks have only one source and all the sinks demand the source. For such networks, upper bounds on the achievable rates can be calculated easily. Moreover, it was shown by Li *et al* [2] that one can design linear network coding schemes to achieve these upper bounds. Non-multicast networks have multiple sources with different sinks demanding different sets of sources, and the situation is much more complicated in this case.

Directed vs Undirected: A directed graph has every edge associated with a direction, and information along that edge can be carried only along that direction. If a link can carry information both ways, it is represented via two edges, one in each direction.

Acyclic vs Cyclic: In an acyclic graph, if you trace any path, you will never come back to the same point.

Following is a description of the class of networks that we will consider. The network consists of a directed graph $\mathcal{G}$, i.e., each edge in the graph has an associated direction in which it can carry data. The graph is assumed not to have any cycles. One node in the graph is a *source* which generates a certain number of symbols per unit time, termed as its *rate*. Each symbol is assumed to belong to some finite field $\mathbb{F}_q$ of size q,[1] and its value is picked arbitrarily by the source.

[1]When q is a prime number, the finite field $\mathbb{F}_q$ is just the set of numbers $\{0,1,\ldots,q-1\}$ with addition and multiplication performed modulo q. A brief introduction to finite fields is available in [3].

Each edge has an associated *capacity*, i.e., the maximum number of symbols it can carry per unit time. The rate of the source and the capacities of the edges are all non-negative integers. A subset of the nodes are *sinks*, and each sink requires all the data that the source generates. This is called a *multicast* network.

Every node can take some linear combination of the symbols that are coming into it, and pass it on the outgoing edges. The *Network Coding* problem is to determine these linear combinations that the nodes perform and pass, in order to ensure that all the source symbols are delivered to all the sinks.

Network Coding problem is to determine the linear combinations that the nodes perform and pass, in order to ensure that all the source symbols are delivered to all the sinks.

The case of networks with a single sink plays a vital role in proving results for the general case of multiple sinks.

2. The Building Block: Networks with a Single Sink

The case of networks with a single sink plays a vital role in proving results for the general case of multiple sinks. In this section, we will consider networks with one sink from a *commodity flow* viewpoint. In commodity flow, all the nodes other than the source and the sink are subject to Kirchhoff's Current Laws (KCL). Some essential terminology is introduced here.

Flow: Flow on an edge represents the number of symbols it will carry (per unit time) in the system, and will obviously not exceed the capacity of that edge. At each intermediate node, the flows *in* and *out* of the node are subject to KCL, i.e., the total flow into the node should be equal to the total flow out of it. In other words, in commodity flow, intermediate nodes do not store anything nor can they generate anything on their own; they just pass on all that they receive. Introducing one imaginary edge into the source and one imaginary edge out of the sink with flows equal to the rate of the source, the net flow into the network is equal to the net flow out of the network. This amount of flow is the value of the flow in the network.

A cut and its capacity: A cut is a partition of the nodes in the network into two parts, $\mathbf{S}$ and $\mathbf{S}^c$. The source $s \in \mathbf{S}$ and the sink $t \in \mathbf{S}^c$. The total capacity of all the edges leading from nodes in $\mathbf{S}$ to nodes in $\mathbf{S}^c$ is called the capacity of the cut. The *min-cut* capacity of the network is the minimum of the capacities of all the cuts in the network.

Ford and Fulkerson (and a little later Elias, Feinstein and Shannon) showed that the maximum value of a flow through the network equals the min-cut in the network.

Max-flow equals min-cut: Ford and Fulkerson (and a little later Elias, Feinstein and Shannon) showed that the maximum value of a flow through the network equals the min-cut in the network. They also showed that when the capacity of every edge is an integer (as we have

assumed here), the max-flow is also an integer, and so is the flow on every edge.

Menger's Theorem: Menger showed that the number of edge-disjoint paths from the source to the sink is equal to the max-flow of the network. In our case when the edge capacities can be any non-negative integer, each edge can be split into multiple parallel edges with unit capacities, to make it amenable to Menger's Theorem.

Thus, in a single sink network, a number of symbols equal to the min-cut of the network can be transferred from source to sink per unit time by making each symbol flow on different edge-disjoint paths.

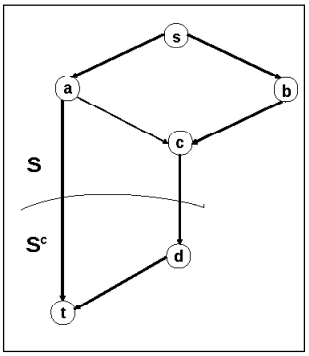


Figure 3. A cut in a network. All the edges in the network have capacities of one symbol per unit time.

Figure 3 shows a cut in a network. The value of this cut is 2 symbols per unit time, and it can be easily verified that this is the min-cut capacity. We can see that there are two edge-disjoint paths from the source to the sink on which commodities can be transmitted. Hence the max-flow in this network is also equal to 2 symbols per unit time.

3. Multicast Networks

We saw in the previous section that when there is only one sink in the network, the min-cut capacity can be achieved using commodity flow itself. The main contribution of network coding towards increasing the achievable rate of a network is in the *multicast* case, i.e., when multiple sinks are present. An example of a multicast network is shown in *Figure* 4.

The main contribution of network coding towards increasing the achievable rate of a network is in the *multicast* case, i.e., when multiple sinks are present.

We will need some notation to formally describe the general network coding setup. Let s be the source node producing μ symbols per unit time. Nodes $t_1,\ t_2, \ldots,\ t_K$ are the K sink nodes. Let $X(1),\ X(2), \cdots, X(\mu)$ be μ source messages. The output at the kth sink is denoted by $Z_k(1),\ Z_k(2), \cdots, Z_k(\mu)$ for $k = 1, \ldots, K$.

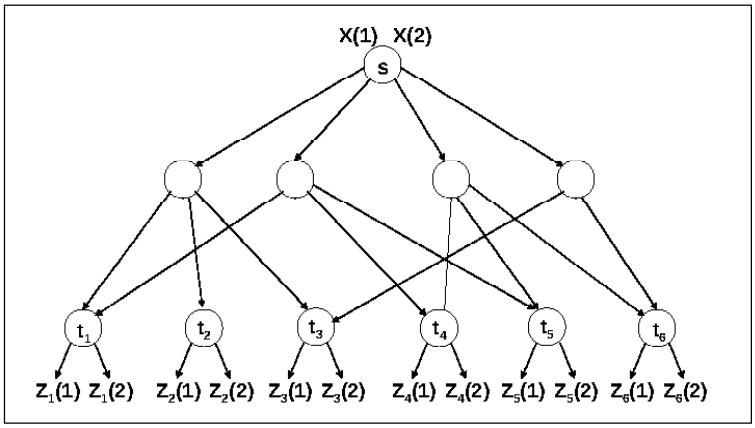


Figure 4. A multicasting example.

Each node in the network can perform linear operations on the incoming symbols. At the source node, we have,

$$Y(e) \;=\; \sum_{k} \alpha(k,e)X(k)\,,$$

where $Y(e)$ is the symbol on edge e emanating from the source, $\alpha(k,e)$ is the coefficient of the message $X(k)$ for the symbol on the outgoing edge e and the summation is over all source messages. At any intermediate node,

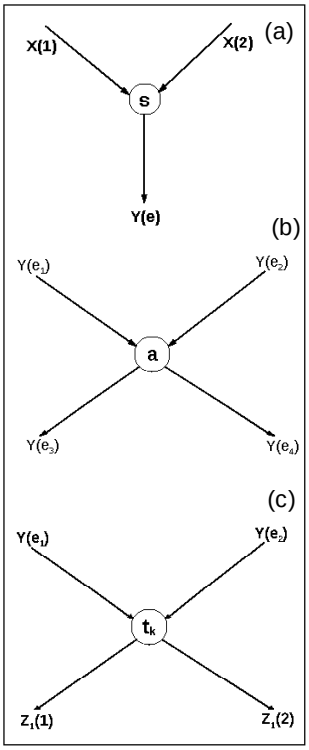
$$Y(e) \;=\; \sum_{e'} \phi(e',e)Y(e')\,,$$

where the summation is over all the incoming edges e' of the intermediate node of which e is an outgoing edge. $\phi(e',e)$ is the coefficient of the incoming message $Y(e')$ from incoming edge e' to the outgoing edge e. And at a sink,

$$Z_k(j) \;=\; \sum_{e'} \beta(e',j)Y(e'),$$

where the summation is over all the incoming edges e' of the sink node under consideration. An illustration of these operations is provided in *Figure* 5.

Figure 5. Notation for input/ output symbols at different nodes: (a) the source, (b) an intermediate node, and (c) a sink.



For simplicity, let us first consider a single sink network and understand the notation. An example of such a

network is shown in *Figure* 6. We will use $\mathcal{G}$ to denote the associated graph.

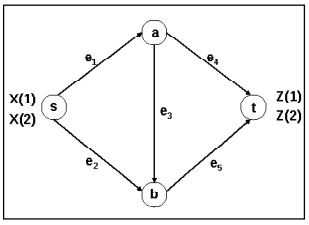


Figure 6. Single sink network $\mathcal{G}$ considered for illustration.

The operations that the source node s performs can be represented by the following matrix operation:

$$[Y(e_1)\ Y(e_2)\ \cdots\ Y(e_5)]$$
$$= [X(1)\ X(2)] \begin{bmatrix} \alpha(1,e_1) & \cdots & \alpha(1,e_5) \\ \alpha(2,e_1) & \cdots & \alpha(2,e_5) \end{bmatrix},$$

i.e., $\underline{Y}^t = \underline{X}^t A$.

The matrix A is termed as *Source-Coupling Matrix* and it encapsulates how the source symbols enter the network. For the example network, A is given by

$$A = \begin{bmatrix} \alpha(1,e_1) & \alpha(1,e_2) & 0 & 0 & 0 \\ \alpha(2,e_1) & \alpha(2,e_2) & 0 & 0 & 0 \end{bmatrix}.$$

Now, let us look at how these symbols propagate in the network. Every intermediate node in the network takes linear combinations of the symbols arriving on its incoming edges and passes them on its outgoing edges. First, a new graph $\mathcal{G}'$ is constructed with edges of $\mathcal{G}$ as the nodes, called *Edge Graph*. Edges in $\mathcal{G}'$ are directed from incoming edges at a node (in $\mathcal{G}$) to outgoing edges of the node (in $\mathcal{G}$). The edge graph for the example is shown in *Figure* 7.

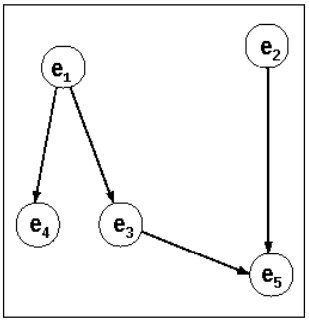


Figure 7. Edge graph $\mathcal{G}'$ or the example network $\mathcal{G}$.

The edge graph provides a *partial ordering* of the edges of $\mathcal{G}$. Partial ordering of edges in a graph gives an ordering for certain pairs of edges. In *Figure* 7 there is an ordering between e_1 and e_3, i.e., e_1 feeds e_3, and similarly between e_2 and e_5. But there is no ordering between e_3 and e_4 or between e_1 and e_2.

Total ordering of edges in a graph provides an ordering for all pairs of edges. Topological sorting of the nodes in $\mathcal{G}'$ gives a total ordering of edges of $\mathcal{G}$ (*Figure* 8). This is simply a 'single-line' ordering of the edges in $\mathcal{G}$ making sure that the partial ordering is respected.

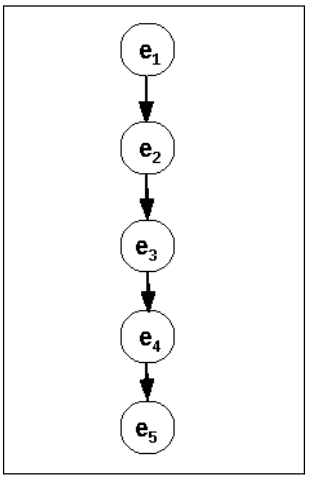


Figure 8. Topological sorting of $\mathcal{G}'$ giving a total ordering of edges of $\mathcal{G}$.

A square matrix is said to be *nilpotent* if some power of the matrix is a zero matrix.

Since each node takes a linear combination of the incoming symbols to compute the outgoing symbols, it is natural to consider an *edge matrix* which captures the coefficients used in the linear combination to obtain an outgoing symbol in terms of the incoming symbols at each node. The edge matrix for the example graph is

$$F = \left[\begin{array}{c|ccccc} \text{from/to} & e_1 & e_2 & e_3 & e_4 & e_5 \\ \hline e_1 & 0 & 0 & \phi(e_1,e_3) & \phi(e_1,e_4) & 0 \\ e_2 & 0 & 0 & 0 & 0 & \phi(e_2,e_5) \\ e_3 & 0 & 0 & 0 & 0 & \phi(e_3,e_5) \\ e_4 & 0 & 0 & 0 & 0 & 0 \\ e_5 & 0 & 0 & 0 & 0 & 0 \end{array}\right].$$

Observe that the topological sorting of the edges of $\mathcal{G}$ provides a nice structure to the edge matrix F. It is always upper triangular with 0's on the diagonal.

A square matrix is said to be *nilpotent* if some power of the matrix is a zero matrix. In this case,

$$F^2 = \begin{bmatrix} 0 & 0 & 0 & 0 & \phi(e_1,e_3)\phi(e_3,e_5) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathrm{F}^3 = [0].$$

Hence F is a nilpotent matrix.

Assume that the network is delay free, i.e., symbols propagate through all the layers simultaneously without any delay.

The symbols propagate in the network in a layer by layer fashion, i.e., from incoming edges of the nodes at one level to outgoing edges of those nodes and then into outgoing edges of nodes of next level and so on. Assume that the network is delay free, i.e., symbols propagate through all the layers simultaneously without any delay. Then, if the total number of layers in the network is N, the propagation of symbols can be represented in the following matrix form called the *Multi-Step Coupling Matrix*:

$$= I + F + \cdots + F^N$$

$$= \quad [I - F]^{-1}.$$

Here the nilpotent property of F is made use of to terminate the summation. Now, we come to the last part of the symbol propagation in the network, i.e., at the sink. Again using the matrix representation, for the example network we have

$$[Z(1)\ Z(2)]$$
$$= [\,Y(e_1)\ Y(e_2)\ \cdots\ Y(e_5)] \begin{bmatrix} \beta(e_1,1) & \beta(e_1,2) \\ \vdots & \vdots \\ \beta(e_5,1) & \beta(e_5,2) \end{bmatrix},$$

i.e., $\underline{Z}^t = \underline{Y}^t B$,

and the matrix B is called the *Sink-Coupling Matrix*. For the example network, B is given by

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \beta(e_4,1) & \beta(e_4,2) \\ \beta(e_5,1) & \beta(e_5,2) \end{bmatrix}.$$

Now we know how symbols propagate from the source to the sink, and are ready to write the overall input-output relation for our example network:

$$[Z(1)\ Z(2)] = [X(1)\ X(2)]\ A\ [I\ +\ F\ +\ F^2]\ B.$$

The sink can recover the source messages if and only if the input-output relation matrix $A\ [I\ +\ F\ +\ F^2]\ B$ is invertible. Hence the values of the variables $\alpha,\ \beta,\ \phi$ are to be chosen such that the matrix $A\ [I\ +\ F\ +\ F^2]\ B$ is full rank, i.e., have non-zero determinant. Now, we can leverage Menger's theorem result to obtain values for these variables in the single sink case. Following is a simple solution using the edge-disjoint paths, which achieves min-cut for the example network.

$$A \quad = \quad \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix},$$

Now we know how symbols propagate from the source to the sink, and are ready to write the overall input-output relation for our example network

The sink can recover the source messages if and only if the input-output relation matrix $A\ [I + F\ + F^2]\ B$ is invertible.

$$F = \left[\begin{array}{c|ccccc} from/to & e_1 & e_2 & e_3 & e_4 & e_5 \\ \hline e_1 & 0 & 0 & 0 & 1 & 0 \\ e_2 & 0 & 0 & 0 & 0 & 1 \\ e_3 & 0 & 0 & 0 & 0 & 0 \\ e_4 & 0 & 0 & 0 & 0 & 0 \\ e_5 & 0 & 0 & 0 & 0 & 0 \end{array}\right],$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

This gives the overall input-output matrix,

$$A\,[I \;+\; F \;+\; F^2]\,B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Hence the sink can recover both the source messages,

$$Z(1) = X(1)\,,$$
$$Z(2) = X(2)\,.$$

The network we considered for illustration had only one sink. So a natural question is, what if there are multiple sinks? This scenario where there is a single source and multiple sinks and all the sinks want the source is termed as *multicast* scenario. Surprisingly, using network coding, min-cut can be achieved for all the sinks simultaneously. Koetter and Medard [4] showed an easy algebraic way in which this can be done. Let us revisit the butterfly example with single source s and two sinks t_1 and t_2 as shown in *Figure* 9a. Since there are two sinks in this network, we will have two input-output relations,

Using network coding, min-cut can be achieved for all the sinks simultaneously. Koetter and Medard [3] showed an easy algebraic way in which this can be done.

$$[Z_1(1)\;Z_1(2)] \;=\; [X(1)X(2)]\;A[I + F + F^2 + F^3]\;B^{(1)},$$

$$[Z_2(1)\;Z_2(2)] \;=\; [X(1)X(2)]\;A[I + F + F^2 + F^3]\;B^{(2)}.$$

For source s to be able to deliver 2 symbols per unit time to both the sinks t_1 and t_2, both the matrices $A\,[I + F + F^2 + F^3]\,B^{(1)}$ and $A\,[I + F + F^2 + F^3]\,B^{(2)}$ need to be full rank. An interesting way of looking at these determinants is to treat them as polynomials with the entries $\alpha(i, e_j),\ \phi(e_i, e_j),\ \beta^{(1)}(e_j, i),\ \beta^{(2)}(e_j, i)$ of the matrices $A, F, B^{(1)}, B^{(2)}$ respectively as the variables. Showing that these determinants are non-zero is equivalent to showing that the respective polynomials are non-zero polynomials.

Now, let us apply Menger's theorem to sink t_1. Since the min-cut between s and t_1 equals 2, there are two edge-disjoint paths from s to t_1 as shown in *Figure* 9b. As in the single sink example we just saw, there is an assignment of 0 or 1 value to the variables $\alpha(i, e_j)$, $\phi(e_i, e_j), \beta^{(1)}(e_j, i)$ such that

$$A[I + F + F^2 + F^3]B^{(1)} = I\ .$$

i.e.,

$$\det\left(A[I + F + F^2 + F^3]B^{(1)}\right) = 1 \neq 0\ .$$

Hence, viewed as a polynomial, $\det\left(A[I + F + F^2 + F^3]\right.$ $\left.B^{(1)}\right)$ is a non-zero polynomial over $\mathbb{F}_q$.

Similarly Menger's theorem when applied to sink t_2 which also has min-cut 2 (*Figure* 9b), says that there is an assignment of 0 or 1 value to the variables $\alpha(i, e_j), \phi(e_i, e_j)$, $\beta^{(2)}(e_j, i)$ such that

$$A\ [I\ +\ F\ +\ F^2\ +\ F^3]\ B^{(2)} = I\ ,$$

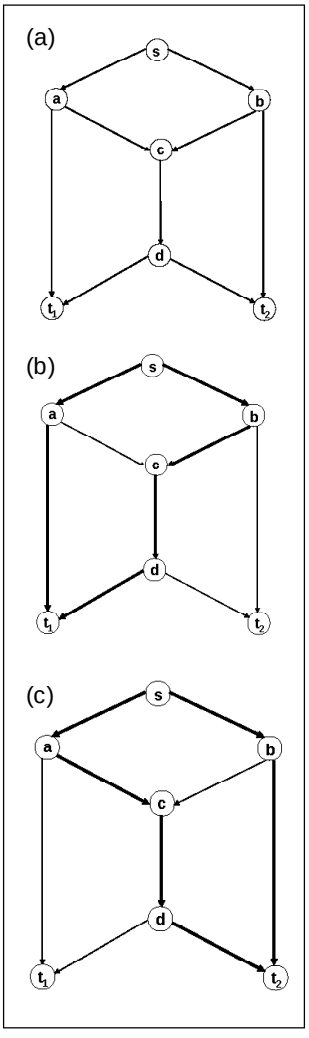
i.e.,

$$\det\left(A\ [I\ +\ F\ +\ F^2\ +\ F^3]\ B^{(2)}\right) = 1 \neq 0\ .$$

Hence, $\det\left(A\ [I\ +\ F\ +\ F^2\ +\ F^3]\ B^{(2)}\right)$ is also a non-zero polynomial over $\mathbb{F}_q$. Thus, the product

$$\det\left(A\ [I\ +\ F\ +\ F^2\ +\ F^3]\ B^{(1)}\right)$$
$$\times\ \det\left(A\ [I\ +\ F\ +\ F^2\ +\ F^3]\ B^{(2)}\right)$$

Figure 9. Menger's theorem applied to the butterfly network: Two edge-disjoint paths from the source to each sink. All edges have capacities of 1 symbol per unit time.



is also a non-zero polynomial.

Following is a lemma regarding non-zero polynomials which is very useful.

Lemma 3.1 *Let $f(X_1, \ldots, X_m)$ be a non-zero polynomial over $\mathbb{F}_q$ with a degree at most d in any variable. If $q > d$, there exists at least one assignment $(\theta_1, \ldots, \theta_m)$ of the variables $(X_1, \ldots, X_m)$ such that*

$$f(\theta_1, \ldots, \theta_m) \neq 0.$$

The probabilistic form of this lemma is called the *Schwartz–Zippel Lemma.*

Proof: The proof uses induction on the number of variables. The result is trivially true for $m = 1$. Assume that the result holds for $m - 1$ variables. Then $f(X_1, \ldots, X_m)$ can be regarded as a polynomial in $(X_2, \ldots, X_m)$ with coefficients that are themselves polynomials in X_1 over $\mathbb{F}_q$. Each such coefficient is of degree $\leq d$, and a value can be assigned to X_1 such that at least one coefficient is non-zero. This gives a non-zero polynomial in $m - 1$ variables of degree $\leq d$. ■

Thus, if $q > d$, there exists an assignment of values to the variables $\alpha(i, e_j)$, $\phi(e_i, e_j)$, $\beta^{(1)}(e_j, i)$, $\beta^{(2)}(e_j, i)$ from $\mathbb{F}_q$ such that the product of the polynomials evaluates to a non-zero value. In this case, we see that there will be an assignment of values to the variables such that both determinants are non-zero, i.e., both sinks can recover all the data. This finally leads us to the basic theorem of network coding, as stated below.

Theorem 3.2 *Let q be a sufficiently large power of 2. We treat a symbol over $\mathbb{F}_q$ as a unit of information. In a directed, delay-free, acyclic graph, with single source s and multiple sinks $t_1, t_2, \ldots, t_K$ and where all edges have integral capacity, if the capacity of the min-cut from source to each of the K sinks is at least μ, then there*

exists a linear network solution that will deliver μ units of information to each of the K sinks simultaneously. ■

4. An Application – The Distributed Storage Problem

In a distributed storage system, the data of the order of petabytes is stored across multiple nodes. Each data centre runs up to the size of a warehouse! When dealing with such large magnitudes of data, achieving reliable storage is a non-trivial task. Another aspect of any distributed storage system is handling failures of storage nodes. When a node fails, it is replaced by a new node by downloading data from the existing nodes. This amount of download consumes a large amount of network bandwidth, and hence needs to be minimized.

When dealing with such large magnitudes of data, achieving reliable storage is a non-trivial task. Another aspect of any distributed storage system is handling failures of storage nodes. When a node fails, it is replaced by a new node by downloading data from the existing nodes.

Given below is a description of a problem in distributed storage and how network coding can be used to solve this.

4.1 *The Problem*

A file of size B is to be stored in a distributed manner across n storage nodes, each having a storage capacity of α units of data (symbols). Each symbol belongs to a finite field $\mathbb{F}_q$ of size q. A data collector (DC) which downloads data stored in any k out of the n nodes should be able to *reconstruct* the entire file. Hence each node needs to store at least B/k symbols. We consider the setting in which each node stores this minimum amount of data, i.e.,

$$\alpha = B/k\,.$$

k out of the n nodes are *systematic*, i.e., store data in uncoded form. A data collector connecting to these k nodes obtains uncoded data without any need for further computation. Hence any data collector would preferably connect to these k nodes. When a systematic node fails,

A data collector (DC) which downloads data stored in any *k* out of the *n* nodes should be able to *reconstruct* the entire file.

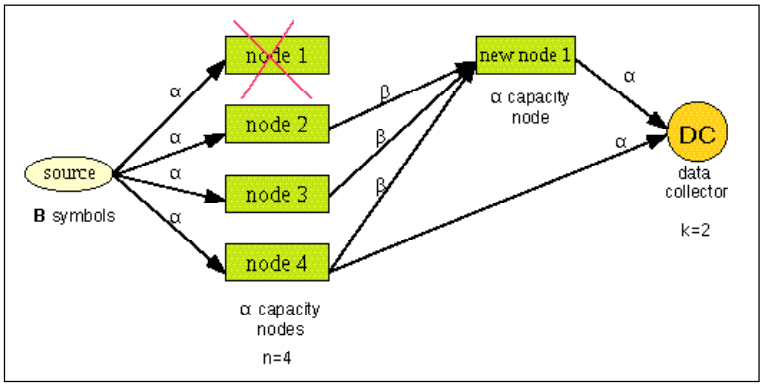


Figure 10. An example of reconstruction and regeneration in a n = 4, k = 2 distributed storage system.

the new node replacing the failed node is permitted to download β symbols each from the $n-1$ existing nodes in order to obtain (and store) the same data that the failed node had stored. This is termed as *exact regeneration*. *Figure* 10 shows reconstruction and regeneration operations in a distributed storage network with $n = 4$ and $k = 2$.

The amount of download $(= (n-1)\beta)$ during repair of a failed node is termed as *repair bandwidth*. The repair operation consumes a lot of network bandwidth, and also since repair needs to be done fast, it is of great interest to reduce the repair bandwidth of the systematic nodes.

In the subsequent sections, we will formulate this problem as a non-multicast network coding problem. We will derive a min-cut bound, and provide codes which achieve this min-cut.

We now present the distributed storage problem as an instance of a non-multicast network coding problem in which the graph of the network is directed, delay-free and acyclic.

4.2 *As a Non-Multicast Problem*

We now present the distributed storage problem as an instance of a non-multicast network coding problem in which the graph of the network is directed, delay-free and acyclic.

The network is viewed as having k source nodes, each corresponding to a systematic node and generating α symbols each per unit time. The non-systematic nodes

are simply viewed as intermediate nodes. Since it is possible to store only α symbols in a non-systematic node, this is taken care of in the graph by (i) splitting each non-systematic node, say node m, into two nodes: m_{in} and m_{out} with an edge of capacity α linking the two with (ii) all incoming edges arriving into m_{in} and all outgoing edges emanating from m_{out}. Since a non-systematic node can store any linear combinations of the source data, each non-systematic node is connected via α capacity edges to every systematic node. This way of representing a storage node was introduced in [5].

The sinks in the network are of two types. The first type correspond to data collectors which connect to some collection of k nodes in the network (with α capacity links) for the purposes of data reconstruction. Hence there are $\binom{n}{k}$ sinks of this type. The second type of sinks represent a new node that is replacing a failed systematic node. Sinks of this type are assumed to connect to the remaining $n-1$ nodes with links of capacities β each. Hence there are k sinks of this type.

Figure 11 shows a part of the network for the general problem, and depicts one of the DCs which connect to some k nodes, and a new node corresponding to failure of the first systematic node. A sink representing a new node on failure of node i is denoted as i'.

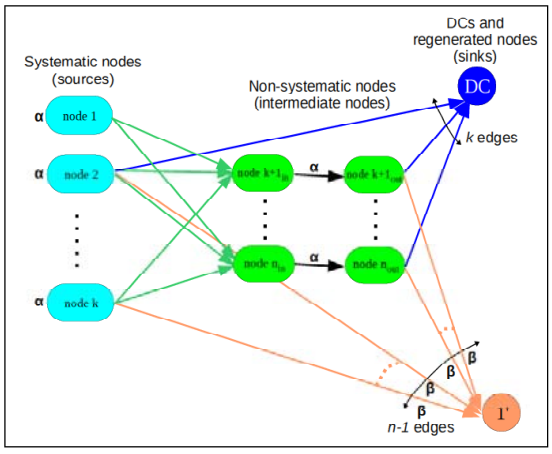


Figure 11. Part of the multicast network of the general distributed storage problem. Unmarked edges have capacity α.

The edges in the cut consist of the $n-k$ edges from the non-systematic nodes to 1', each of which have a capacity of β symbols.

4.3 *The Min-Cut Bound*

Recall that a cut is a partition of the nodes into two sets. In the non-multicast setup, a cut needs to have at least one source node on the source-side of the cut whose corresponding sink lies on the sink-side. Consider the cut with nodes $1', 2, \ldots, k$ on the sink-side of the cut, and the remaining nodes on the source-side. Since sink $1'$ is on the sink-side with its corresponding source 1 on the source-side, the α units of information from source 1 need to pass through the cut. The edges in the cut consist of the $n-k$ edges from the non-systematic nodes to $1'$, each of which have a capacity of β symbols. Thus we need

$$(n-k)\beta \geq \alpha \ .$$

Rewriting this equation, we get a bound on β as

$$\beta \geq \frac{\alpha}{n-k} \ .$$

In the next section, we will demonstrate codes which actually achieve this bound on β, hence also showing that this is indeed the min-cut.

4.4 *Solution to this Network*

In this section we will provide a solution to the distributed storage problem for the parameters ($n = 4,\ k = 2,\ B = 4$). Interested readers can refer to [6] for a solution for the general parameters.

The network corresponding to this set of parameters is shown in *Figure* 12. These parameters force $\alpha = \frac{B}{k} = 2$, and to achieve the cut-set bound we set $\beta = 1$. Let all symbols belong to the finite field $\mathbb{F}_5$. Let the two source symbols produced by the first source be x_1 and x_2, and the two source symbols generated by the second source be y_1 and y_2. The code is as follows.

The first non-systematic node stores the two symbols

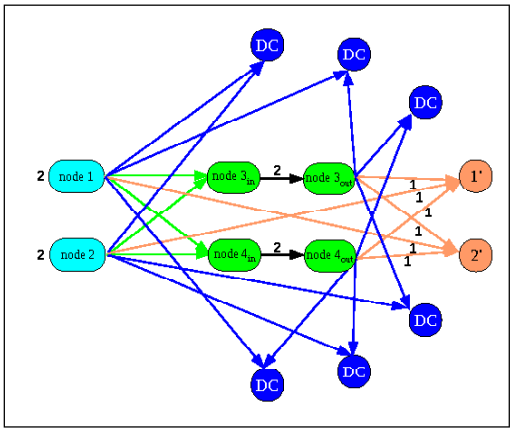


Figure 12. Complete network for the distributed storage problem for n = 4, k = 2 and α = 2. Unmarked edges have capacity α = 2.

$2x_1 + 2x_2 + y_1$ and $x_2 + 2y_1 + 2y_2$. The second non-systematic node stores $2x_1+4x_2+2y_1$ and $x_2+2y_1+4y_2$.

The four symbols from any two nodes correspond to four independent linear equations in four variables, which can be solved to obtain values of the source symbols. Hence reconstruction when a data collector connects to any two nodes is satisfied.

For regeneration of the first systematic node, the other three nodes pass their first symbols, namely y_1, $2x_1 + 2x_2 + y_1$ and $2x_1 + 4x_2 + 2y_1$. On subtracting y_1 from the other two symbols passed, the new node can obtain the values of $2x_1 + 2x_2$ and $2x_1 + 4x_2$. These are two linearly independent equations in two variables, and can be easily solved to obtain the values of x_1 and x_2. Thus the first systematic node is exactly regenerated.

Similarly, for regeneration of the second systematic node, each of the three existing nodes passes its second symbols. In an analogous fashion, the new node can obtain the symbols y_1 and y_2.

Thus this code achieves the cut-set bound for the network, and we have an optimal $(4, 2)$ exact regenerating code for distributed storage.

Suggested Reading

[1] R Ahlswede, N Cai, S Y R Li, and R W Yeung, Network information flow, *IEEE Trans. Inform. Theory*, Vol.46, pp.1204–1216, July 2000.

[2] S Y R Li, R W Yeung, and N Cai, Linear network coding, *IEEE Trans. Inform. Theory*, Vol.49, p.371, Feb.2003.

[3] Priti Shankar, Decoding Reed Solomon Codes Using Euclids Algorithm, *Resonance*, Vol.12, No.4, 2007.

[4] Ralf Koetter and Muriel Medard, An algebraic approach to network coding, *IEEE/ACM Transactions on Networking*, Vol.11 No.5, pp.782–795, Oct. 2003.

[5] A G Dimakis, P B Godfrey, M Wainwright and K Ramchandran, Network Coding for Distributed Storage Systems, *Proc. IEEE INFOCOM*, 2007.

[6] N B Shah, K V Rashmi, P V Kumar and K Ramchandran, Explicit codes minimizing repair bandwidth for distributed storage, *Proc. Information Theory Workshop*, Cairo, January 2010.

Address for Correspondence
K V Rashmi, Nihar B Shah and P Vijay Kumar
Department of Electrical Communication Engineering
Indian Institute of Science
Bangalore 560 012.
Email:
rashmikv@ece.iisc.ernet.in
nihar@ece.iisc.ernet.in
vijay@ece.iisc.ernet.in