Network Coding: Beyond Throughput Benefits

Christina Fragouli
School of Computer and Communication Sciences
EPFL, Switzerland

***Abstract*— Network coding enables novel network functionalities and thus offers a wider canvas of choices when optimizing an information flow problem. In this paper we examine the simplest possible information flow problem, a unicast connection, and explore what we believe is one of the most attractive features network coding offers: the ability to enable near optimal performance in a completely decentralized and randomized setting. This is an especially attractive feature for wireless applications. However, it comes at the cost of an overhead in terms of rate, that can be significant for applications that operate using relatively short frame lengths, as is the case in the wireless setting. We review the efforts in the literature to either alleviate this overhead, or alternatively, to exploit it for network management and control.**[1]

I. Introduction

The research in information flow through networks can be clustered around three broad questions: (i) How much information can we send over a given network, that is, what are fundamental performance limits. (ii) How do we send it, that is, can we design provably efficient algorithms that have desirable properties and meet practical requirements. Finally, (iii), how do we build and maintain our networks, that is, how do we perform network monitoring, management and control.

Many of the problem formulations that arise in information flow over networks can be cast as optimization problems: we want to maximize the throughput for a given bandwidth, we want to minimize the delay for a given throughput, we want to optimize the reliability for a given energy efficiency.

Network coding can be viewed as increasing the search space over which we optimize, as is illustrated in the example developed in Figs. 1–3. Assume node A in Fig. 1 receives two bits u_1 and u_2 per time slot, and can only send one bit per time slot to node B. With routing, we have two choices, to send either u_1 or u_2, as in Fig. 2 (there is also the choice of sending nothing, but we will always assume we communicate through explicitly sending information in this paper). If we allow node A to perform any function f that maps two bits to one bit, we have 2^4 choices[2]. One such choice is depicted in Table I. In general, if u_1 and u_2 are elements of a finite field $\mathbf{F}_q$, i.e., take values in the finite set $\{0,\ 1,\ \ldots, q-1\}$, with routing we still only have two choices, either send u_1 or u_2, while with linear coding we have q^2 choices and with nonlinear coding q^{q^2} choices.

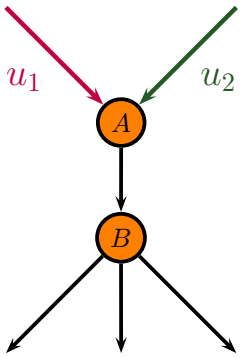

Fig. 1. Node A receives symbols u_1 and u_2.

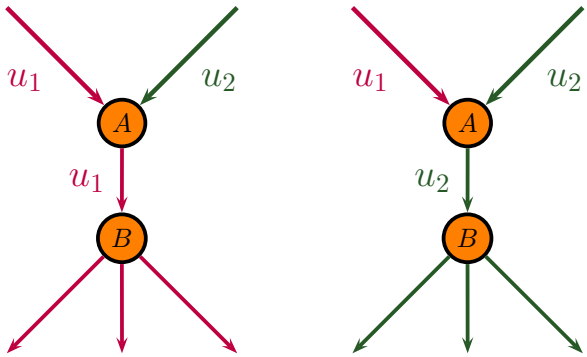

Fig. 2. With routing, node A has two choices: to send to node B either symbol u_1 or symbol u_2.

Increasing the search space over which we optimize has two benefits: First, it allows to find better values for the objective function we are optimizing for, that were not possible when the search was restricted to a

[1]This work was supported by the Swiss National Foundation through the FNS grant PP00P2_128639.

[2]Some of these choices may not be useful, for example the constant function, and some choices are equivalent for some applications, but the general argument of growth we will make remains.

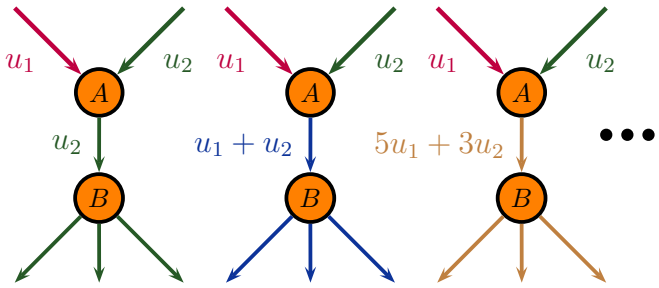

Fig. 3. If we enable node A to perform linear combining, node A can send to node B a linear combination of u_1 and u_2, and thus have a much larger set of choices.

u_1	u_2	$f(u_1, u_2)$
0	0	1
0	1	0
1	0	1
1	1	1

TABLE I

FUNCTION f TAKES AS INPUTS BITS u_1 AND u_2 AND PROVIDES AS OUTPUT ONE BIT, AS DETERMINED BY THE ABOVE TABLE.

routing solution. For example, we can achieve higher throughput, smaller delay, or higher reliability. Several papers and tutorial monographs have demonstrated such benefits, see for example [13], [12], [1], [2], [3], [4], [5]. Second, it allows to find the optimal values using smaller complexity. For example, for some cases, an optimization problem that is computationally hard, i.e., required essentially an exhaustive search of the routing solution space, can now be solved efficiently [11]. In some other cases, even a randomly selected solution in the increased space is optimal with high probability [6], [7], [8]. In both cases, the common underlying reason for the reduction in complexity is that, in the increased search space there are *many more solutions that are optimal*, and thus, we can find such a solution more easily.

This tutorial paper is motivated from the complexity benefits of network coding. Complexity benefits promise to have a very significant impact in practice – increasing the throughput in a network by a factor of two might be very desirable, but, achieving this using at the same time very simple decentralized operations might be even more important.

Instead of reviewing the set of instances where network coding offers complexity benefits, in this paper we make the conscientious choice to examine in depth the simplest information flow problem: a unicast connection, where a source sends information to a single receiver. The case of multicast, where a source sends the same information to multiple receivers, can also be treated in the same manner. Our goal is to critically examine what indeed are the complexity benefits we can get in practice, and at what cost they come. What we will argue is that, there exist significant benefits, but as always, there also exist choices to make, and we need to make them in the right way for the specific application goals we want to achieve.

The paper starts with introducing in Section II randomized network coding, which is a very attractive approach for decentralized operation; then, we discuss in Section III how using coding vectors allows to make randomized coding practical, but at the cost of an overhead that increases the packet length; we describe the efforts to reduce this overhead in Sections IV and V; and finally, we take a different viewpoint in Section VI and show that instead of getting rid of this overhead, we can in fact instead exploit it, for network monitoring and control.

We want to emphasize that randomized network coding is not the only instantiation where network coding offers complexity benefits; on the contrary, there are multiple such cases, more or less studied in the literature, and we expect there will be many more to come. However, the case we study is definitely the most well studied up to now, and thus allows us a more mature understanding of the potential benefits, as well as the design choices we need to make in network coding. Even in this case, we only examine in detail one particular aspect of the overhead, and not the additional computational complexity that we require for encoding and decoding at the network nodes; we only briefly mention these in Section VII. We conclude the paper with a summary and discussion in Section VIII.

II. RANDOMIZED NETWORK CODING IS USEFUL ...

For simplicity, we consider a network represented as a graph, where edges correspond to channels and nodes to terminals. We assume that time is slotted, and during each time-slot we can send through each edge one symbol over a finite field $\mathbf{F}_q$ (for example, if we operate over the binary field $\mathbf{F}_2$ we can send one bit per time slot). We describe this as having unit capacity edges.

The simplest possible question we can ask in information flow through networks is, how to send information from a single source to a single receiver. This is called a *unicast* connection.

How much information we can send was answered in 1956, from Ford-Fulkerson and independently from Elias, Fenstein and Shannon, in the famous min-cut max-flow theorem [15], [16]. This theorem states that the

maximum amount of information we can send equals the value of the minimum cut that separates the source from the receiver. A cut is a set of edges whose removal disconnects the source from the receiver. The minimum cut (mincut) value h equals the minimum number of edges we need to remove to disconnect the source from the receiver. For example in Fig. 4 the mincut value to receiver R equals $h = 2$, since removing the edges (B_2, R) and (B_3, R) disconnects the receiver from the network.

How we can efficiently send information at the mincut rate was also answered by Ford Fulkerson in [15] by providing a polynomial time algorithm. This algorithm identifies h edge-disjoint paths that connect the source to the destination. We then simply route the information along these paths. From each such path, the receiver receives one symbol per time-slot. Fig. 4 shows two edge-disjoint paths, that can be used to route symbols u_1 and u_2 to the receiver.

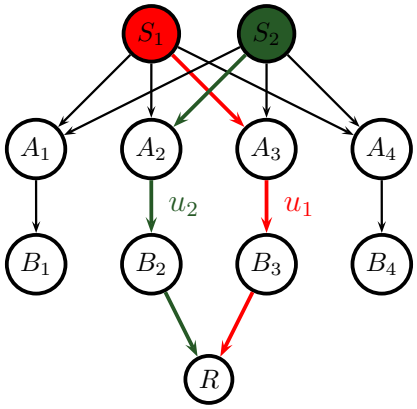

Fig. 4. Two edge-disjoint paths towards receiver R.

However, this algorithm, and routing in general, presupposes that we identify in advance the paths that we use, which in turn assumes that the network is static for our purposes. This not always the case in practical networks, where we can have significant variations of the network connectivity. To capture a dynamically changing network we assume in Fig. 5 that during each time-slot, the receiver R connects to two different nodes B_i and B_j. For example, R could be connected to B_1 and B_2 at time t, to B_2 and B_4 at time $t + 1$, etc. This could model a wireless environment, with a mobile receiver. It could also model a lossy environment, where all nodes B transmit information, but only two of the transmitted symbols are received correctly at each time-slot.

If the network nodes are constrained to perform routing, then we can route through each edge AB either

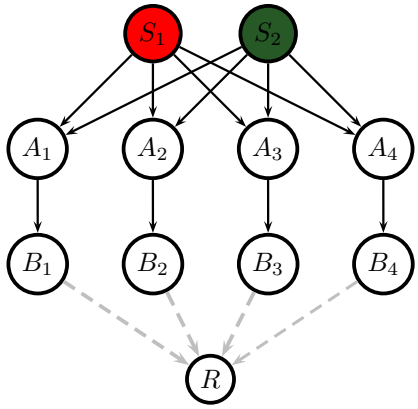

Fig. 5. A dynamic unicast connection: receiver R at each time-slot connects to two different nodes B_i and B_j. For example, R could be connected to B_1 and B_2 at time t, to B_2 and B_4 at time $t + 1$, etc.

symbol u_1 or symbol u_2, as depicted in Fig. 6. Think of these symbols as colors, red and green, that we need to assign to edges AB. Given that we have 4 edges (A_i, B_i) and two colors, it follows that there will exist at least two edges that get colored with the same color. Namely, they route the same information. If the receiver then happens to connect to these two edges, it will receive the same information from both of them. If we assume that each node A selects uniformly at random which of the two symbols to forward, then the probability of failure equals $\frac{1}{2}$. If do not use a randomized scheme but a deterministic scheme we again get a constant probability of error. For example, if we deterministically assign to half the edges (A_i, B_i) the symbol u_1 and to the other half the symbol u_2, then the probability of error will equal $\frac{k-1}{2k}$, where k is the number of (A_i, B_i) edges. This probability again goes to $\frac{1}{2}$ as k increases.

Now assume that nodes A perform linear coding. That is, each node A uniformly at random selects and sends to node B one linear combination of the symbols u_1 and u_2, say $x_i u_1 + x_j u_2$, using the coefficients $x_i,\ x_j \in \mathbf{F}_q$, the finite field of operation. It is then sufficient that the receiver R receives two linearly independent equations, as it can solve these to retrieve u_1 and u_2. Fig. 7 shows a possible choice of linear combinations. Note that, no matter to which two nodes B the receiver connects, it can retrieve two linearly independent equations to solve for u_1 and u_2. In particular, R needs to solve a set of equations of the form

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix},$$

where $\{y_1, y_2\}$ are the two symbols receiver R receivers,

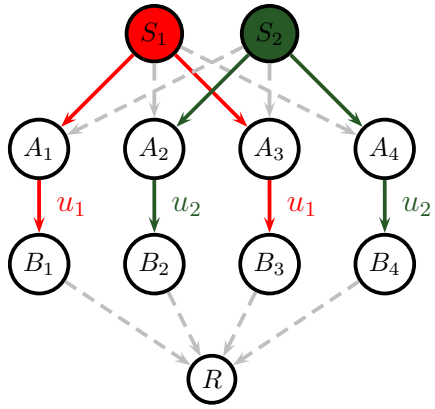

Fig. 6. Routing sends one symbol, either u_1 or u_2 through each edge (A_i, B_i).

and $\{x_i\}$ are the coefficients for the linear combinations used to create $\{y_1, y_2\}$. For example, if R observes edges (A_1, B_1) and (A_3, B_3) it needs to solve the equations

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}.$$

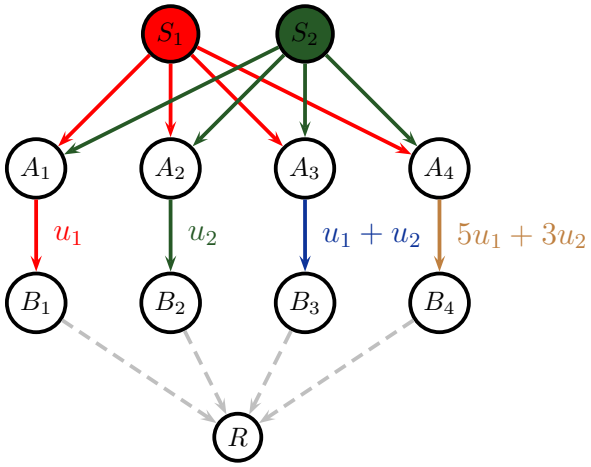

Fig. 7. Coding sends a linear combinations of the symbols u_1 and u_2 through each edge (A_i, B_i).

If the A-nodes select the coefficients $\{x_i\}$ uniformly at random over a field $\mathbf{F}_q$, the probability of failure equals

$$(1 - \frac{1}{q^2})\frac{q-1}{q^2} \approx \frac{1}{q}.$$

That is, we have now achieved a decentralized operation, with probability of error that goes to zero as the field size of operation increases. Thus, use of coding enables operation in dynamically changing environments with no centralized knowledge and near optimal performance.

Note that, we could connect an arbitrary number of receivers on the B-nodes of our network, as Fig. 8 shows. Each receiver would still receive two linearly independent equations and thus rate equal to its min-cut. This network shows a special case of the main multicasting theorem in network coding, which states that over all networks, if we allow intermediate network nodes to perform linear coding, we can then send rate h simultaneously to an arbitrary number of receivers, provided that the mincut towards each receiver equals at least h. Moreover, we can achieve this mincut in a decentralized manner, using randomized network coding. Given that we can treat multicasting and unicasting in the same manner, we will in the following only refer to a unicast connection; however, all the discussion also applies to the case of multicasting the same information to an arbitrary number of receivers.

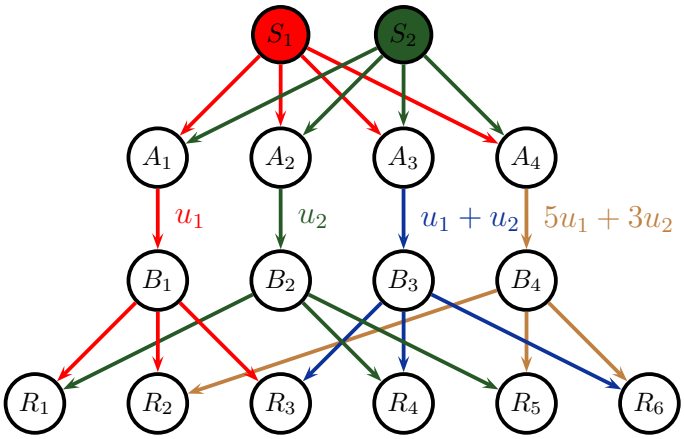

Fig. 8. We can connect an arbitrary number of receivers to the B-nodes. Provided that each receiver connects to at least two different such nodes, they can all simultaneously retrieve symbols u_1 and u_2, with the same code.

In the example in Fig. 7, the receiver has to solve a 2×2 system of equations. In general, over an arbitrary network where the mincut to each receiver equals h and where intermediate nodes perform linear combining, the overall system performs a linear transformation captured by an $h \times h$ transfer matrix $\mathbf{A}$. The receiver has to solve an $h \times h$ set of equations of the form

$$\begin{bmatrix} y_1 \\ \vdots \\ y_h \end{bmatrix} = \mathbf{A} \begin{bmatrix} u_1 \\ \vdots \\ u_h \end{bmatrix},$$

where $\mathbf{A}$ is the transfer matrix from the sources to the receiver. Note that to decode, the receiver needs to know the matrix $\mathbf{A}$. But how does the receiver get this knowledge?

One approach is to assume that each node that performs coding (the A-nodes in our example) selects always the same fixed linear combination, and communicates this information to the receiver. The receiver, knowing the linear coefficients each node uses, can calculate the overall transfer matrix. However, using a fixed code is challenging over practical networks, because networks are lossy and subject to delays - a fixed code would again require a relatively static network, and good synchronization between the network nodes. Thus, using static coding instead of routing, seems as if we have exchanged one requirement for static knowledge with another. Is there an alternative approach?

III. ... But it Requires Coding Vectors!

An alternative approach is to assume no static knowledge about the network topology or the coding coefficients, in fact allow intermediate nodes at each time slot to randomly to select what coefficients to use, but learn online what are the specific linear combinations that the destination receives. We can do this through some form of training, that is termed coding vectors in the network coding literature [17].

Operation with coding vectors is as follows. Assume that we have h information packets $\{\boldsymbol{p}_i\}$ to send, and each $\boldsymbol{p}_i$ contains L bits. We can treat each such packet as equivalently containing $T = \frac{L}{\log_2 q} = \frac{L}{m}$ symbols over a finite field $\mathbf{F}_q$ with $q = 2^m$. That is, we can interpret m consecutive bits of a packet as one symbol over the field $\mathbf{F}_{2^m}$, with each packet consisting of a vector of $T = L/m$ symbols in $\mathbf{F}_{2^m}$. For example, if $q = 2^4$, we can consider each packet of length L bits to equivalently consist of $\frac{L}{4}$ symbols over $\mathbf{F}_{16}$. In the previous section, we implicitly assumed that each packet $\boldsymbol{p}_i$ has length $T = 1$, i.e., $L = m$, and consists of a single symbol u_i. In the following, we will denote the symbols contained in packet $\boldsymbol{p}_i$ as $\{\boldsymbol{p}_i^k\}$, for clarity of notation.

When intermediate nodes linearly combine their received packets, the summation has to occur for every one of the T symbol positions. For example, to create the linear combination $3\boldsymbol{p}_1 + 8\boldsymbol{p}_2$, a node would perform $3\boldsymbol{p}_1^k + 8\boldsymbol{p}_2^k$, $k = 1 \dots T$, where $\boldsymbol{p}_1^k$ and $\boldsymbol{p}_2^k$ are the k^{th} symbols of $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, respectively.

We call the h source packets a *generation*. The generation specifies what is the set of source packets that we are allowed to combine together. In a network we may have simultaneously different generations, generated by different sources, or by the same source at different times.

To each of the h source packet $\boldsymbol{p}_i$ in the generation, we append a different coding vector $\boldsymbol{p}_i^C$ of h symbols over $\mathbf{F}_{2^m}$, or equivalently, hm bits. These h vectors form a basis of the h-dimensional space $\mathbf{F}_q^h$. For example, the sources can employ the orthonormal basis $\{\boldsymbol{e}_i\}$ as coding vectors, that is, $\boldsymbol{p}_i^C = \boldsymbol{e}_i = (0, \dots, 0, 1, 0, \dots, 0) \in \mathbf{F}_q^h$, where $\boldsymbol{e}_i$ has zeros everywhere and 1 at the ith position. Thus the packets sent by the sources are of the form

$$[\boldsymbol{e}_i \mid \boldsymbol{p}_i], \tag{1}$$

where we assumed without loss of generality that the coding vector is placed at the beginning of the packet. The coding vector keeps track of the linear combination of the source packets that is contained in the packet. For example, a node that receives the packets $[\boldsymbol{e}_1 \mid \boldsymbol{p}_1]$ and $[\boldsymbol{e}_2 \mid \boldsymbol{p}_2]$ may create the packet

$$3[\boldsymbol{e}_1 \mid \boldsymbol{p}_1] + 8[\boldsymbol{e}_2 \mid \boldsymbol{p}_2] = [3\boldsymbol{e}_1 + 8\boldsymbol{e}_2 \mid 3\boldsymbol{p}_1 + 8\boldsymbol{p}_2].$$

The coding vector $3\boldsymbol{e}_1 + 8\boldsymbol{e}_2 = (3, 8, 0, \dots, 0)$ reflects the linear combination of the source packets. In general a packet propagating in the network will have the form

$$\boldsymbol{p} = [\boldsymbol{p}^C \mid \boldsymbol{p}^I], \tag{2}$$

where $\boldsymbol{p}^I \in \mathbf{F}_q^L$ is a linear combination of source packets and $\boldsymbol{p}^C \in \mathbf{F}_q^h$ is the coding vector that contains the linear coefficients for the combined source packets.

Each receiver that receives h packets $\{\boldsymbol{p}_i^C\}$ with linearly independent coding vectors can recover the original source information. To do so, the receiver solves the linear equations $\{\boldsymbol{p}_i^I = \sum_{j=1}^h x_i^j \boldsymbol{p}_j\}$, where for each received packet $\boldsymbol{p}_i^C$ the h coefficients x_j^i are contained in its coding vector $\boldsymbol{p}^C$. That is,

$$\begin{bmatrix} \boldsymbol{p}_1^I \\ \boldsymbol{p}_2^I \\ \vdots \\ \boldsymbol{p}_h^I \end{bmatrix} = \underbrace{\begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^h \\ x_{21}^1 & x_2^2 & \cdots & x_2^h \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1}^1 & x_n^2 & \cdots & x_n^h \end{bmatrix}}_{\boldsymbol{A} \in \mathbf{F}_q^{n \times n}} \begin{bmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \vdots \\ \boldsymbol{p}_h \end{bmatrix}, \tag{3}$$

where the ith row of matrix $\boldsymbol{A}$ is the coding vector corresponding to received packet $\boldsymbol{p}_i$. If the receiver collects h linearly independent coding vectors, the matrix $\boldsymbol{A}$ is full rank, and thus the original packets can be recovered.

Coding vectors allow for a very simple, distributed operation, at the cost of an overhead. In particular, if the packet length consists of $T = \frac{L}{\log q}$ symbols over the finite field $\mathbf{F}_q$, and we code together h packets, the overhead $\mathcal{O}$ of the coding vectors per packet equals

$$\mathcal{O} = h \log_2 q.$$

We thus have the following trade-off. With routing, and centralized knowledge, we can send $hT \log_2 q$ information bits to receiver R. However, if we do not have centralized knowledge, there is a constant probability

that routing will only deliver a fraction of the information rate. In the example of Fig. 6 with probability $1/2$ we would only get half the rate. In contrast, if we implement randomized network coding with coding vectors, then with probability that goes to one as the field size of operation increases, we would send a number of information bits equal to

$$h(T-h)\log_2 q.$$

Routing has no overhead; but has a high probability of failure in dynamically changing environments. Coding has an overhead - that of coding vectors - but is very robust to dynamic changes. Thus the ideal would be, if we could remove the overhead of the coding vectors.

IV. Removing the Coding Vectors: Noncoherent Network Coding

Coding vectors come at an overhead that is not significant if we are interested in long packet lengths L, but that can very fast become prohibitive if we need to operate with short frame lengths. For example, wireless networks offer a very promising application for network coding, since they form a dynamically changing environment where the decentralized operation network coding allows could be very useful. However, in wireless we are constrained to operate using relatively short block lengths and thus the overhead coding vectors require could be significant. Example 1 gives a specific such instantiation for sensor networks.

Example 1: Consider a sensor network consisting of 100 nodes, each sending a message to a sink. To implement network coding using coding vectors over a field of size $q = 2^4$, we would need to use 50 bytes of each packet simply for the coding vectors. In the TinyOS operating system [18], which is perhaps the most popular for sensor nodes, a typical frame length allows approximately 30 bytes for data transmissions. Thus clearly this is not a viable approach. ■

Subspace coding is a beautiful idea, recently proposed in [19], [20], that dispenses of the need to convey coding vectors. The observation in this scheme is that, neither the receiver nor the sources need to know the transfer matrix to the receiver, *i.e.*, the specific linear coefficients in matrix $\mathbf{A}$ in (3). In this case, in analogy to the case of wireless communications, when neither the transmitter nor the receiver know the channel, we say that we have *noncoherent* communication.

In noncoherent communication, we can take advantage of the fact that the network performs only linear operations, and communicate information using subspaces[3], which are unaffected by the linear operations performed on them. Indeed, if the source inserts in the network h linearly independent vectors of length L, these span a subspace π of the L-dimensional vector space; any linear transformation the network performs to these vectors will still result to vectors inside the same subspace π. If the receiver collects h linearly independent vectors, these will span π, and thus the receiver will know that the source "inserted" π in the network. Thus, the source can send different information messages by using different h-dimensional subspaces; that is, the source can use a subspace codebook that maps each information message to a set of vectors spanning a different subspace. The following toy example illustrates how subspace coding works.

Example 2: Assume that we use vectors of length $L = 3$, and operations over a field of size $q = 3$. For two vectors v_1 and v_2 in $\mathbf{F}_3^3$, denote as $\pi =< v_1, v_2 >$ the two-dimensional subspace of $\mathbf{F}_3^3$ that these vectors span. Assume that a source has four different messages to send to a receiver. To do so, it uses a subspace codebook that contains the following two-dimensional subspaces of the three-dimensional space

$$\pi_1 =< [1\ 0\ 0],\ [0\ 1\ 0] >, \quad \pi_2 =< [1\ 0\ 0],\ [0\ 0\ 1] >,$$
$$\pi_3 =< [0\ 1\ 0],\ [0\ 0\ 1] >, \quad \pi_4 =< [0\ 1\ 0],\ [1\ 0\ 1] > .$$

To send a message to the receiver, the source inserts in the network the basis vectors of the corresponding subspace; for example, to send the first message, it sends the vectors $[1\ 0\ 0]$, $[0\ 1\ 0]$. These vectors are linearly combined by the intermediate network nodes in an arbitrary fashion, and the receiver receives some linear combinations of these. However, there is no need to learn at the receiver what the exact linear combinations are, but simply to collect two linearly independent vectors. The receiver may get for example the vectors $[1\ 1\ 0]$, and $[2\ 1\ 0]$. These vectors span π_1; thus the receiver decodes that the first message was send. ■

A natural question to ask is, what are is the information rate we can achieve with noncoherent network coding, and what are the benefits we get as compared to the coding vectors approach, in terms of the overhead. To study this, we need to calculate the capacity of noncoherent network coding [21], [22], [24], [25].

To derive information theoretical bounds, we need to first define what is our channel. We will use the approach in [21], [22]. We assume communication in time-slots; in each time-slot, the source inserts in the network m packets of length T over some finite field $\mathbf{F}_q$, and each receiver collects n packets that consist of random combinations of the source packets. The mincut h between the source and the receiver equals

[3] A subspace is a subset of a vector space, that forms a vector space in itself.

$h = \min\{m, n\}$. At time slot[4] t, the receiver observes

$$Y(t) = \mathbf{A}(t)X(t), \tag{4}$$

where $X(t) \in \mathbf{F}_q^{m\times T}$, $\mathbf{A}(t) \in \mathbf{F}_q^{n\times m}$, and $Y(t) \in \mathbf{F}_q^{n\times T}$. At each time-slot, the receiver receives n packets of length T (captured as rows of the matrix $Y(t)$) that are random linear combinations of the m packets injected by the source (captured as rows of the matrix $X(t)$).The packet length T can be interpreted as the coherence time of the channel, during which the transfer matrix $\mathbf{A}$ remains constant. Each element of the transfer matrix $\mathbf{A}$ is chosen uniformly at random from $\mathbf{F}_q$, changes independently from time-slot to time-slot, and is unknown to both the source and the receiver.

The channel described by (4) can be interpreted as a discrete memoryless channel with input alphabet $\mathcal{X} \triangleq \mathbf{F}_q^{m\times T}$ and output alphabet $\mathcal{Y} \triangleq \mathbf{F}_q^{n\times T}$. The capacity of this channel is given by

$$C = \max_{P_X(x)} I(X;Y), \tag{5}$$

where $P_X(x)$ is the input distribution and $I(X;Y)$ the mutual information between the variables X and Y. To achieve the capacity, a coding scheme may employ the channel (4) multiple times and in this case each codeword is a sequence of input matrices from $\mathcal{X}$.

The following theorem characterizes the capacity of noncoherent network coding assuming our channel model [22]. We first state this theorem formally, and then interpret it.

Theorem 1: Consider the channel given in (4) and assume that $\mathbf{A}$ is drawn uniformly at random from $\mathbf{F}_q^{n\times m}$ and independently from block to block. Then there exists finite q_0 such that for $q > q_0$ the optimal input distribution is non-zero only for the matrices whose rank belongs

$$\mathcal{A} = \left\{\min[(T-n)^+, m, n, T], \ldots, \min[m, n, T]\right\}, \tag{6}$$

which we call the active set. The capacity of the channel is

$$\begin{aligned} C &= \left[\log_2\left(\sum_{i\in\mathcal{A}} q^{i(T-i)}\right) + \mathbb{1}_{\{T:\text{odd}\}} + \mathcal{O}(q^{-1}\log q)\right] \\ &= \left[\mathbb{1}_{\{T:\text{odd}\}} + i^*(T-i^*)\log_2 q + \mathcal{O}(q^{-1}\log q)\right], \end{aligned} \tag{7}$$

where $i^* = \arg\min_{i\in\mathcal{A}} |T/2 - i| = \min\{m, n, \lfloor T/2\rfloor\}$. Moreover, the optimal input distribution is uniform over all matrices X of the same dimension, and the probability of employing matrices X of rank i equals

$$\alpha_i^*(x) = 2^{-C} q^{i(T-i)} \left[1 + \mathcal{O}(q^{-1}\log q)\right], \quad \forall i \in \mathcal{A},$$

where C is the capacity of the channel.

This theorem says that the maximum rate we can achieve using packets of length T behaves as $i^*(T - i^*)\log q$, where $i^* = \min\{m, n, \lfloor T/2\rfloor\}$. This result is derived assuming large values of q, however, numerical simulations indicate a very fast convergence to this value as q increases. Fig. 9 depicts the capacity for small values of q, calculated using the Differential Evolution toolbox for matlab [26].

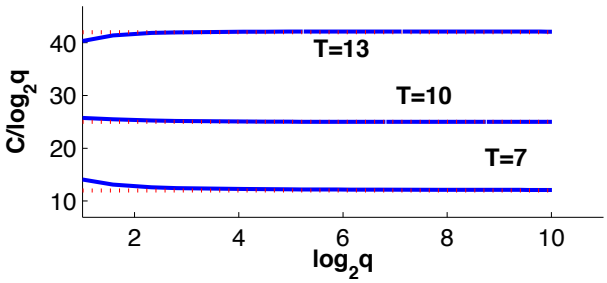

Fig. 9. Numerical calculation of the capacity for small values of q and $m = 11$, $n = 7$. The dotted line depicts $i^*(T - i^*)$.

From this theorem we can derive the following conclusions for noncoherent network coding.

1) Subspace coding is optimal: Confirming our intuition, this theorem states it is optimal to communicate using subspace coding.

Additionally, it specifies the dimension of the subspaces we should use as codewords. Note that, since every time we use the channel we send m vectors each of length T symbols, we can use subspaces of $\mathbf{F}_q^T$ that have dimension m, but also subspaces that have dimension $m-1$, $m-2$, etc., simply by sending some vectors linearly dependent with others. Theorem 1 specifies what exactly are the dimensions we should employ. Interestingly, it turns out that this depends on the relative values of m, n and T. In particular, for $T \leq n$, we should use subspaces of all dimensions. As T increases, the set of used dimensions gradually decreases, until we reach $T \geq \min\{m, n\} + n$. For T larger than this value, it is sufficient to use subspaces of a single dimension, equal to $\min\{m, n\}$. This is depicted in Fig. 10.

2) Subspace coding vs. coding vectors: Our motivation for looking at noncoherent communication was to avoid the overhead of the coding vectors. A natural question is, what is the difference in the achievable rates between using coding vectors and noncoherent coding, and how much have we gained?

[4]In the rest of the paper we will omit for convenience the time index t.

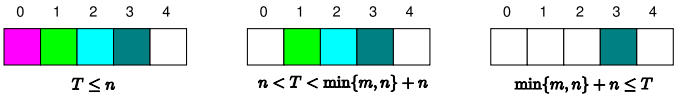

Fig. 10. Active subspace dimensions for $m = 4$, $n = 3$.

TABLE II
INFORMATION LOSS FROM USING CODING VECTORS WHEN $n = m$.

	$T \leq 2m$	$T > 2m$
$C - R_{cv}$	$o(1)$	$o(1) = (i^* - 1)(T - i^*)\frac{\log_2 q}{q} + O(q^{-1})$

Table II summarizes this difference. As we see from this table, subspace coding does not offer benefits as compared to the coding vectors approach, at least for large field size.

Table II is calculated as follows. Depending on the length T, we will use a generation that contains a different number k of source packets, with $k \leq h$ (it is in fact easy to see that for $T \leq 2h$, it is optimal to use a generation of size $K = T/2$). Using a generation of size k in practice means that the source inserts k packets in the network, with coding vectors also of length k. Thus the achievable rate R_{cv} in this case equals

$$R_{cv} = k(T - k) \log_2 q,$$

since each packet includes a coding vector of length k and $T - k > 0$ information symbols. We then substract this rate from the noncoherent capacity C.

To conclude, noncoherent network coding achieves better rates than use of coding vectors for packet frames of length $T < 2h$, where h is the mincut between the source and the receiver, but surprisingly, does not offer rate benefits for $T \geq 2h$. That is, whether we learn the channel or not, we achieve the same rate. Given that subspace coding involves higher complexity than use of coding vectors for the encoding and decoding, for $T \geq 2h$ we might as well use coding vectors.

V. Is There an Alternative Approach? Compressed Coding Vectors

Noncoherent coding does not offer the rate benefits we expected as compared to use of coding vectors; that is, both use of coding vectors and subspace coding result on approximately the same overhead, or loss rate, as compared to routing. It seems that there is an underlying reason why this happens.

A possible interpretation is that, both in the case of coding vectors and subspace coding, we allow potentially all source packets to get combined together to create each single received coded packet. Recall that in our model, h is the generation size, or equivalently the perceived mincut of the network. So for each coded packet, the receiver needs to know h linear coefficients to decode it. Moreover, we have assumed that these h linear coefficients are chosen independently at random. However, for some networks, this is too strong a requirement and results in an unnecessary loss of information rate.

We here propose an alternative approach that employs shortened or compressed coding vectors to efficiently convey the coding coefficients [27]. The observation our approach leverages is that, in many cases, it is sufficient to employ coding vectors that allow at most m source packets to get combined. This naturally occurs in some applications, where for example only source packets originating from neighboring nodes get combined. We can also artificially restrict the number of source packets that get combined, by appending to each coded packet a few bits to count the number of source packets it contains. Note that, the receiver will eventually still need to solve a set of h linear equations to retrieve the source data; our approach only shortens the coding vectors that convey the linear coefficients to the receiver.

Our design problem can now be stated as follows. Given a generation that contains h source packets, each receiver is going to observe packets that contain linear combinations of *at most* m source packets. We want to design coding vectors that allow us, by receiving each combined packet, to determine which linear combination of the source packets it contains. The classical coding vectors design would utilize coding vectors of length h. Here we explore what, under our assumptions, is the smallest length r of coding vectors we need to employ, and how can we select them. For m much smaller than h, the classical coding vectors become sparse. We can thus compress them, by replacing them with shorter vectors, that still allow the receivers to extract the original coding vectors and decode the source messages. Our construction utilizes properties of algebraic error correcting codes, and proceeds as follows.

Select a linear code $\mathcal{C} = [h, k, d]_q$ where $d = \min(2m+1, h+1)$ with k as large as possible. Consider the $r \times h$ parity check matrix $\mathbf{H}_\mathcal{C}$ where $r \triangleq h - k$. As coding vector, assign to source packet $\boldsymbol{x}_i$ the ith column of the matrix $\mathbf{H}_\mathcal{C}$, which we will denote as $\boldsymbol{h}_i$. That is,

$$\boldsymbol{h}_i = \boldsymbol{e}_i \cdot \mathbf{H}_\mathcal{C}^T. \tag{8}$$

We call these vectors *compressed coding vectors*. Thus the sources insert to the network the packets

$$[\boldsymbol{h}_i \mid \boldsymbol{p}_i]. \tag{9}$$

Intermediate nodes linearly combine their received packets, exactly as in the classical coding vector approach. The coded packets propagating in the network will now have the form

$$\boldsymbol{p} \triangleq [\hat{\boldsymbol{p}}^C \mid \boldsymbol{p}^I], \tag{10}$$

where $\hat{\boldsymbol{p}}^C \in \mathbb{F}_q^r$ denotes the compressed coding vector appended to packet $\boldsymbol{p}$. This is related to the classical coding vector $\boldsymbol{p}^C$ that describes the linear transform from the source packets as

$$\hat{\boldsymbol{p}}^C = \boldsymbol{p}^C \cdot \mathbf{H}_{\mathcal{C}}^T. \tag{11}$$

If m packets are allowed to be combined, with m much smaller than the length h of the coding vector $\boldsymbol{p}^C$, this can be viewed as compressing the sparse vector $\boldsymbol{p}^C$, and hence the compressed coding vector terminology.

The following example gives a particular choice for the compressed coding vectors.

Example 3: Suppose the number of packets in every generation is $h = 15$ and each packet in the network contains linear combinations of at most $m = 2$ packets, which leads to $d = 2m + 1 = 5$. Let also $q = 2^4$. The code $\mathcal{C}$ can be chosen to be the Reed-Solomon code with parameters $\mathcal{C} = [15, 11, 5]_q$. The parity check matrix of $\mathcal{C}$ can be written as follows

$$\mathbf{H}_{\mathcal{C}} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{15-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(15-1)} \\ 1 & \alpha^3 & \alpha^6 & \cdots & \alpha^{3(15-1)} \\ 1 & \alpha^4 & \alpha^8 & \cdots & \alpha^{4(15-1)} \end{bmatrix},$$

$$\mathbf{H} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 & \cdots & \mathbf{p}_h \end{bmatrix},$$

where α is a primitive element of $\mathbf{F}_{2^4}$. Each column of $\mathbf{H}_{\mathcal{C}}$ can be assigned to one of $h = 15$ source packets. ■

How do we decode?

Upon receiving a packet $\boldsymbol{p}$ with compressed coding vector $\hat{\boldsymbol{p}}^C$, the receiver needs to recover the original coding vector to construct the system of linear equations in (3). Is this even possible? and can we do it efficiently? The reason this construction enables the receivers to decode follows from a well known property that the columns of matrix $\mathbf{H}_{\mathcal{C}}$ satisfy. Namely, if a code $\mathcal{C}$ has minimum distance $d = 2m + 1$, then any set of $d - 1 = 2m$ columns of the matrix $\mathbf{H}_{\mathcal{C}}$ are linearly independent [28]. As a result, any linear combination of a specific set of m columns results in a unique vector, that cannot be created by linearly combining any other set of m columns. More formally, there is an injective map between the vectors $\boldsymbol{p}^C$ and $\hat{\boldsymbol{p}}^C$ in (11).

To efficiently perform the decoding, we note that it is sufficient to find what are the non-zero positions of the vector $\boldsymbol{p}^C$. If we know this, and using the knowledge of the matrix $\mathbf{H}_{\mathcal{C}}$, we can uniquely also recover the linear coefficients in the original coding vectors. If we have the original coding vectors, we can then solve the usual system of linear equations and decode.

TABLE III

TIME FOR EXHAUSTIVE SEARCH IN SECONDS. EXPERIMENTS ARE RUN ON A SINGLE CORE OF AN INTEL CENTRINO DUO2, AT 3 GHZ.

h/m	2	3	4
15	0.00018	0.0020	0.017
31	0.00097	0.024	0.48
63	0.0047	0.24	10.4

One approach to find the nonzero positions is through exhaustive search. For small values for m and h this in a fast computer can be feasible, as Table III illustrates. However, there are $\binom{h}{m}$ possible m-sets of non-zero positions to consider, and this number grows exponentially in m for constant h.

A more practical approach is to use some known algebraic codes for $\mathcal{C}$ like BCH codes, Reed-Solomon codes [30], Goppa codes [29], algebraic geometry codes [31], etc., to recover the original coding vectors efficiently. For all of the codes mentioned above there exists a version of the Berlekamp-Massey algorithm [34], [35] which allows the receivers to find the location of non-zero elements of the original coding vectors.

What have we gained?

Using the compressed coding vectors method, the length of coding vectors reduces from h to $r = h - k$. We can formally prove that the construction we have described achieves in fact the minimum possible value of r for our problem [27]. Thus the benefits we get depend on the required size r, that we next calculate.

From the Singleton bound for a code $\mathcal{C}$ we have

$$\begin{aligned} k &\leq h - d + 1 \\ &= h - \min(2m, h), \end{aligned}$$

so for $\frac{h}{2} \leq m \leq h$ we have $k = 0$ which implies that we can select w.l.o.g. the full rank $h \times h$ parity matrix $\mathbf{H}_{\mathcal{C}}$ to be the identity matrix. In this case, we recover the usual network coding approach with coding vectors $\{\boldsymbol{e}_i\}$ appended to the sources packets, and there is no benefit from our approach.

From the Gilbert-Varshamov bound [32] we have an upper bound for the length of compressed coding vectors r that for the case $m < \frac{h}{4}$ can be simplified to

$$r \leq hH_q\left(\frac{d-1}{h}\right) = hH_q\left(\frac{2m}{h}\right), \tag{12}$$

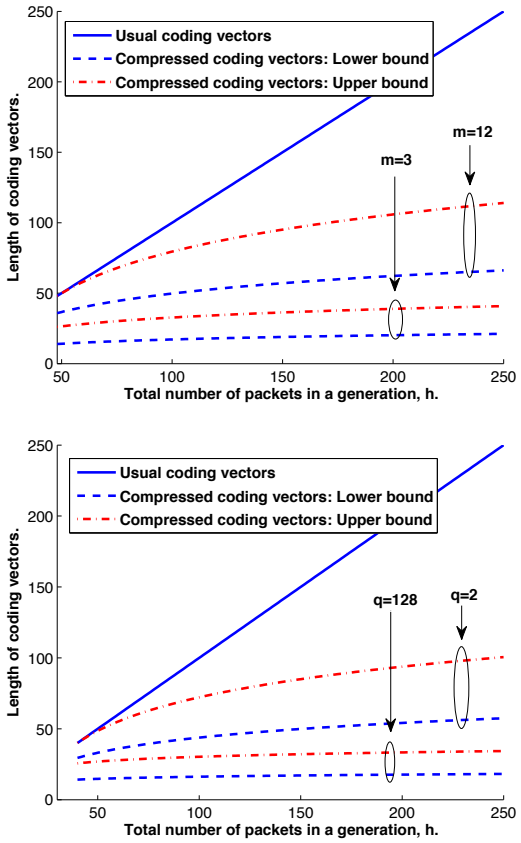

Fig. 11. Bounds on the length of the compressed coding vectors, r. First figure: when $m = 3$ and $m = 12$ sources get combined, as a function of the number of packets in a generation h, $q = 2$. Second figure: when $m = 10$ sources get combined, as a function of the number of packets in a generation h for two values of field size $q = 2$ and $q = 128$.

where $H_q(\delta) = \delta \log_q(q-1) - \delta \log_q \delta - (1-\delta)\log_q(1-\delta)$ is the q-ary entropy function. Also, the Sphere Packing bound leads to a lower bound on the length of compressed coding vectors where for $m < \frac{h}{2}$ we can simplify it to obtain

$$r \geq hH_q\left(\frac{d-1}{2h}\right) - \frac{1}{2}\log_q\left(4(d-1)\left(1-\frac{d-1}{2h}\right)\right)$$
$$= hH_q\left(\frac{m}{h}\right) - \frac{1}{2}\log_q\left(8m\left(1-\frac{m}{h}\right)\right). \quad (13)$$

From (12) and (13), for fixed values of m and as the number h of source packets grows, we have

$$m\log_q h + \mathcal{O}(1) \leq r \leq 2m\log_q h + \mathcal{O}(1),$$

So using the proposed method, we can reduce the growth of coding vectors from $\mathcal{O}(h)$ to $\mathcal{O}(m \log h)$.

Example 4: Using a table of the best codes known (from [28] and [37]), we can see for example that, there exist binary linear codes of length $h = 127$ with redundancy $r = 35$ and minimum distance $d = 2m+1 = 11$, which is in fact a shortened version of a $[128, 93, 11]$ Goppa code [29]. Thus in a network with 127 source packets in each generation if at most $m = 5$ source vectors get combined, we need to use coding vectors of length $r = 35$ instead of $h = 127$. ■

In the previous example, it is assumed that the network nodes perform binary network coding; *i.e.*, nodes only XOR the packets. However, if the field size is increased, we can have shorter compressed coding vectors, as we can see in the following Example 5.

Example 5: A Reed-Solomon code is a $[h, k, d]_q$ linear code with $h = q - 1$ and $k + d - 1 = h$ [30]. To make a comparison with the binary codes in Example 4, we use $h = 127$ and consider using the parity matrix of a Reed-Solomon code over a field of size $q = 2^8$. If we set $d = 2m + 1 = 11$, the length of the compressed coding vectors equals $r = h - k = d - 1 = 10$. Thus, as compared to the case of classical coding vectors, the coding vector headers decrease from 112 bytes to only 9 bytes; this is a much higher compression than what we had achieved in Example 4. ■

VI. Another Viewpoint: Keeping the Coding Vectors Can be Useful

The idea of compressing the coding vectors leverages the fact that, in a practical network, the empirical distribution of the coding vectors is not uniform - there is some structure. We will argue in this section that this is because coding vectors implicitly carry information about the network topology as well as its state, where by state we refer to link or node failures, congestion in some parts of the network, etc. Instead of attempting to remove this structure to gain rate benefits by compressing the coding vectors, we can instead accept the associated rate loss, and use this information that the coding vectors passively collect, for network monitoring and control. In this section we motivate this alternative approach, through a short discussion on the topological information the coding vectors carry, and a practical application in bottleneck discovery for peer-to-peer networks.

A. Topology Inference

Assume a source S has a generation of h independent packets $\{u_1, \ldots, u_h\}$, u_i, to distribute to a set of receivers, through the use of randomized network coding over a field $\mathbf{F}_q$ and coding vectors.

Now assume that, contrary to what we considered before, the mincut towards the receiver is much smaller than the generation size h; this is infact most of the times the case the practice. Thus, to disseminate the information, the source keeps injecting packets into the

network; intermediate nodes have buffers where they store all the packets they have collected, and use these collected packets to create the linear combinations they forward towards the receivers.

If we consider the coding vectors that each network node i has collected at a given time t, then these vectors will span a subspace $\Pi_i(t)$ of the h-dimensional space $\mathbf{F}_q^h$. These subspaces will have nesting properties that reflect the structure of the network. For example, in Fig. 12, assume that at time t nodes A has received 3 coding vectors that span a subspace $\Pi_A(t)$, and nodes B and C have each received 2 coding vectors each. We will then have that $\Pi_B(t)$ and $\Pi_C(t)$ are subspaces of $\Pi_A(t)$. Moreover, with high probability these two subspaces will be distinct [38]. Studying properties of the subspaces that network nodes collect, one can show that in fact the coding vectors allow to perfectly reconstruct the network topology (under some mild conditions) for arbitrary network topologies, as the following theorem (proved in [38]) states.

Theorem 2: In a network employing randomized network coding over $\mathbf{F}_q$, a sufficient condition to uniquely identify the topology with high probability as $q \gg 1$, is that

$$\Pi_i(t) \neq \Pi_j(t) \quad \forall\, i, j \in V, \quad i \neq j, \tag{14}$$

for some time t. We can achieve this by observing the subspaces the network nodes have collected at appropriate times.

Note that collecting the subspace information needs to occur only once at a small additional overhead, especially for large values of generation size h. Leveraging this information for designing practical network monitoring schemes is a subject of current research.

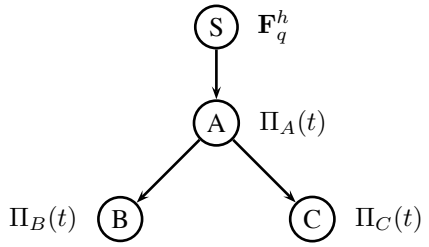

Fig. 12. The source S sends information to nodes A, B and C that are connected through a tree; assuming the coding vectors the source employs span the space $\mathbf{F}_q^h$, then at time t the coding vectors these nodes have collected, span subspaces $\Pi_A(t), \Pi_B(t)$ and $\Pi_C(t)$, respectively.

B. Bottleneck Discovery in Peer-to-Peer Networks

Several works have proposed employing network coding for content distribution, pointing out a number of potential benefits, such as simple decentralized operation,

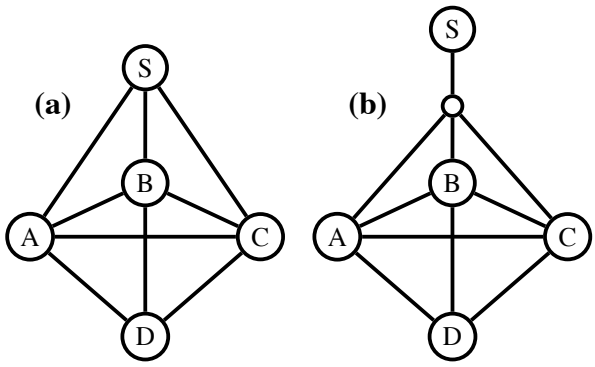

Fig. 13. The source S distributes packets to the peers A, B, C and D over the overlay network in fig. (a), that uses the underlying physical network in fig. (b).

and no need for tight topology control [41], [42], [43], The only requirement on the topology is a connectivity pattern which allows information to flow fast among the network nodes, i.e., creates a large mincut between pairs of nodes.

However, this does not always occur naturally. Peer-to-peer are very dynamically changing networks, where hundreds of nodes may join and leave the network within seconds. All nodes in this network are connected to a small number of neighbors (four to eight). An arriving node is allocated neighbors among the active participating nodes, which accept the solicited connection unless they have already reached their maximum number of neighbors. As a result, nodes that arrive at around the same time tend to get connected to each other, since they are all simultaneously available and looking for neighbors. That is, we have formation of clusters and bottlenecks in the network, that impede the information flow.

To avoid this problem, one of the methods adopted in practical protocols is to ask all nodes to periodically drop one neighbor and reconnect to a new one among an active peers list. This randomized rewiring results in a fixed average number of reconnections per node independently of how good or bad is the formed network topology. Thus to achieve a good, on the average, performance in terms of breaking clusters, it entails a much larger number of rewiring than required, and unnecessary topology changes.

An alternative approach is to have peers initiate topology rewirings when they detect they are in a cluster. Consider the toy network depicted in Figure 13(a) where the edges correspond to logical (overlay network) links. The source S has h packets to distribute to four peers. Nodes A, B and C are directly connected to the source S, and also among themselves with logical links, while node D is connected to nodes A, B and C. In this overlay

network, there exist three edge-disjoint paths between the source and any other nodes.

Assume now (as shown in Figure 13(b)) that the logical links SA, SB, SC share the bandwidth of the same underlying physical link, which forms a bottleneck between the source and the remaining nodes of the network. As a result, assume the bandwidth on each of these links is only $1/3$ of the bandwidth of the remaining links. Note that even if we kept track of the complete logical network structure, we would not know the existence of the bottleneck and the asymmetry between the link bandwidths.

Node D however, can infer this information by observing the coding vectors it receives from its neighbors A, B and C. Indeed, when node A receives a coded packet from the source, it will forward a linear combination of the packets it has already collected to nodes B and C and D. Now each of the nodes B and C, once they receive the packet from node A, they also attempt to send a coded packet to node D. But these packets will not bring new information to node D, because they will belong in the linear span of coding vectors that node D has already received. Similarly, when nodes B and C receive a new packet from the source, node D will end up being offered three coded packets, one from each of its neighbors, and only one of the three will bring to node D new information.

More formally, the coding vectors nodes A, B and C will collect will effectively span the same subspace. The coded packets these nodes will offer to node D to download will belong in significantly overlapping subspaces and thus be redundant. Node D can infer from this passively collected information that there is a bottleneck between nodes A, B, C and the source, and can thus initiate a connectivity change.

Experimental Results

We will here show simulation results that compare peer-initiated algorithms leveraging the coding vector information [39], and randomized rewiring. We start from the topology illustrated in Figure 14, with three clusters: cluster 1 has 15 nodes and contains the source, cluster 2 has also 15 nodes, while the number of nodes in cluster 3 increases from 15 to 250. The source is node 1, and belongs in the first cluster. The bottleneck links are indicated with arrows (and thus indicate the underlying physical link structure). All algorithms perform similarly in terms of average collection time; however, Fig. 15 clearly shows that the randomized algorithm results in a much larger number of rewirings.

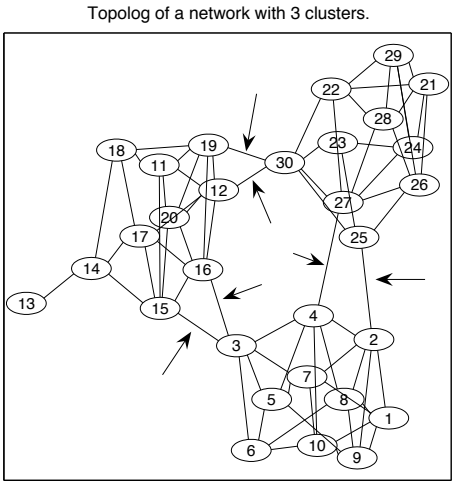

Fig. 14. Topology with three clusters: cluster 1 contains nodes 1–10, cluster 2 nodes 11–20 and cluster 3 nodes 21–30.

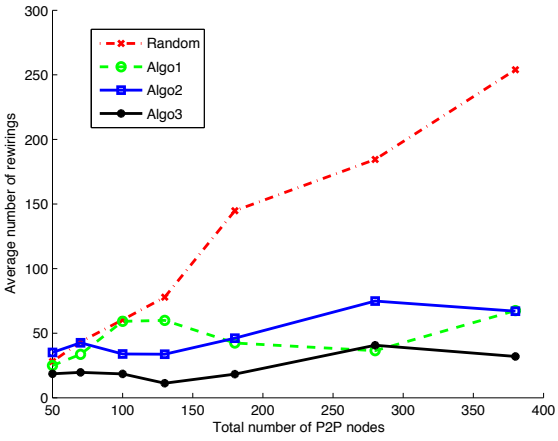

Fig. 15. Average number of rewirings, for a topology with three clusters: cluster 1 has 15 nodes, cluster 2 has 15 nodes, while the number of nodes in cluster 3 increases from 20 to 250. The algorithms Algo1-Algo3 use differently the information in the coding vectors, and are described in [39].

VII. Additional Overheads: The finite Field Size

Implementing coding requires network nodes to have the capability of performing linear operations (additions and multiplications) over finite fields. To implement such operations, we could for example store addition and multiplication tables for each fixed field $\mathbf{F}_q$.

The size of the finite field we need the use depends on the number of receivers. For multicasting to N receivers, the current network coding algorithms operate assuming a field size $q > N$. However, since there is no reason the number of receivers interested in a multicasting content would be fixed, it is not clear how to decide in advance

what finite field size to use. We can always assume a very large N and accordingly over-provision for q, but as the complexity of the finite field operations and the coding vectors overhead increases with the field size, this is clearly not the most economic approach. We may end up spending the overhead we gained from compressing the coding vectors for example, by inefficiently selecting the finite field of operation.

An alternative approach to performing finite field operations is provided by *vector network coding* [45], [46], [47]. The idea is that, instead of performing scalar operations over a finite field, we perform operations on vectors. The source transmits h vectors of length T, where the elements of the vectors are over a fixed finite field, for example, the binary field $\mathbb{F}_2$. Thus, intermediate network nodes only need to have implemented operations over the binary field; as the number of the receivers grows, we can simply increase the length of the employed vectors, while always performing addition and multiplication over the binary field.

In vector network coding each intermediate network node receives from each incoming edge a binary vector of length T; it combines the received vectors by multiplying them with $T \times T$ coding matrices and then adds them to create the new vectors to propagate towards the destinations. That is, intermediate nodes linearly combine their incoming vectors using coding matrices, where these matrices play the same role as scalar coding coefficients in traditional network coding. The code design consists in selecting the length T and the $T \times T$ coding matrices so that each receiver receives information at rate h.

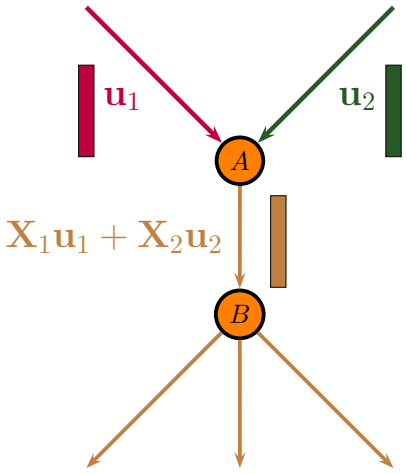

Fig. 16. In vector network coding, node A combines the received vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ by multiplying them with $T \times T$ matrices $\mathbf{X}_1$ and $\mathbf{X}_2$.

Scalar operations over a field of size q^L can be translated to vector operations employing $L \times L$ matrices using a well known mapping between finite fields operations and operations using matrices and vectors [44]. Thus, code designs for scalar network coding over a field $\mathbb{F}_{q^L}$ can be directly translated to code designs for vector network coding using $L \times L$ matrices with elements in $\mathbb{F}_q$. However, directly designing codes for vector network coding can still be useful; indeed, there exist q^{L^2} $L \times L$ matrices over $\mathbb{F}_q$, while a translation from scalar coding only employs q^L of these matrices. Thus, vector network coding offers a larger space of choices for optimizing cost parameters. For example, we can use structured matrices for the encoding at the network nodes to reduce the encoding complexity [47], [48]; or, we can attempt to minimize the communication block length L [45], [46].

Vector network coding is an area that is just starting to be explored in the network coding community, but early results seem very promising [45], [46], [47]. For example, it might be that in a fraction $\frac{1023}{1024}$ of cases, we will be able to find in polynomial time binary coding matrices of size at most 3×3, for an arbitrary number of receivers [46]. Moreover, the randomized algorithms translate from finite fields to matrices, thus allowing a randomized network operation [48].

VIII. Summary and Discussion

When solving optimization problems over networks, use of network coding increases the solution space. Thus, we can achieve points we could not achieve before - such as the mincut to each receiver when multicasting - or, we can achieve points we could achieve before in much simpler ways - for example, the mincut for a unicast connection over a dynamically changing graph.

We examined in detail one particular manifestation of the complexity benefits network coding can offer, the possibility to achieve rate very close to the mincut capacity, in a completely decentralized operation mode. We started our discussion from the well known fact that routing allows to achieve the mincut capacity, however, pre-supposes a static knowledge of the network. If the network dynamically changes, routing will not be able to achieve good performance. Using randomized coding over a finite field $\mathbf{F}_q$, on the other hand, allows to achieve the mincut in a completely decentralized fashion with probability that goes to one as the field size increases.

However, with random coding, each receiver, to decode, needs to know the transfer matrix and solve linear equations. To learn the transfer matrix, we need to employ coding vectors, that essentially are a form of training. Coding vectors allow for a very simple, distributed operation, at the cost of an overhead.

We thus have the following trade-off. Routing has no overhead; but has a high probability of failure in dynamically changing environments. Coding has an overhead - that of coding vectors - but is very robust to dynamic changes. Thus the ideal would be, if we could completely eliminate or at least reduce the overhead of the coding vectors.

Noncoherent network coding does not require knowledge of the transfer matrix for decoding, and thus one could expect less overhead as compared to the coding vectors approach that explicitly learns the transfer matrix. However, it turns out that both use of coding vectors and subspace coding results to the same overhead, or loss rate, as compared to routing.

To reduce this overhead, we can leverage the fact that the coding vectors will not be completely random but will have some structure. This is because coding vectors implicitly carry information about the network topology as well as its state. We can remove this structure to gain rate benefits by compressing the coding vectors. Alternatively, we can instead accept the associated rate loss, and use the information that the coding vectors passively collect, for network monitoring and control.

Finally, network coding requires an overhead not only in terms of rate, but also in terms of operational complexity. In particular, it requires encoding and decoding operations over a finite field. Recent work in vector coding shows there exist efficient algorithms to perform network coding while always using the binary field, by using matrix operations for the linear combining.

To conclude, network coding makes possible to have reliable communication when it was not possible before, for example in very dynamically changing networks; there are several ways to do this, through coding vectors, subspace coding, compressed coding vectors; and each way has its own benefits and drawbacks.

Many research questions are currently under investigation or remain open even in the specific application we have examined. To mention a few, are there other, perhaps more efficient ways to compress the coding vectors? Does subspace coding allows other benefits apart from rate, for example, in terms of error correction or security? Are there other ways to use the information the coding vectors bring? Parallel lines of work also look at optimizing for specific applications requirements and for a variety of traffic scenarios. We hope that by addressing such questions, apart from satisfying a theoretical interest, we can build a foundation for designing future communication systems that operate under demanding conditions and meet performance requirements challenging to maintain before.

REFERENCES

[1] R. W. Yeung, S.-Y. R. Li, N. Cai, and Z. Zhang, "Network coding theory: A tutorial" *Foundation and Trends in Information Theory*, 2006.

[2] C. Fragouli and E. Soljanin, "Network coding: Fundamentals," *Foundations and Trends in Networking*, vol. 2, pp. 1–133, 2007.

[3] C. Fragouli and E. Soljanin, "Network coding: Applications," *Foundations and Trends in Networking*, vol. 2, pp. 135–269, 2008.

[4] T. Ho and D. S. Lun, "Network coding: An introduction," *Cambridge University Press, Cambridge, U.K.*, 2008.

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

[6] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung. "Network information flow", *IEEE Transactions on Information Theory*, vol.46, no. 4, pp. 1204-1216, July 2000.

[7] T. Ho, M. Mèdard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory*, vol. 52, pp. 4413–4430, Oct. 2006.

[8] S. Jaggi, P. Sanders, P. Chou, M. Effros, S. Egner, K. Jain and L. Tolhuizen, "Polynomial time algorithms for multicast network code construction," *IEEE Transactions on Information Theory*, vol. 51, no. 6, pp. 1973–1982, 2005.

[9] R. Koetter and M. Médard, "Beyond routing: an algebraic approach to network coding", *Proc. IEEE INFOCOM,* New York, NY, June 2002, vol. 1, pp. 122–130, 2002.

[10] Z. Li, B. Li, L. C. Lau, "On achieving optimal multicast throughput in undirected networks", *In Joint Special Issue on Networking and Information Theory, IEEE Transactions on Information Theory (IT) and IEEE/ACM Transactions on Networking (TON)*, vol. 52, no. 6, pp. 2184-2194, June 2006.

[11] D. S. Lun, N. Ratnakar, M. Médard, R. Koetter, D. R. Karger, T. Ho, E. Ahmed, and F. Zhao, "Minimum-cost multicast over coded packet networks", *IEEE Transactions on Information Theory*, vol. 52, no. 6 pp. 2608-2623, June 2006.

[12] C. Chekuri, C. Fragouli and E. Soljanin, "On average throughput benefits and alphabet size for network coding", *joint special issue of the IEEE Transactions on Information Theory and the IEEE/ACM Transactions on Networking*, vol. 52, no. 6, pp. 2410–2424, June 2006.

[13] A. Agarwal and M. Charikar, "On the advantage of network coding for improving network throughput", *IEEE Information Theory Workshop (ITW)*, pp. 247-249, San Antonio, Texas, 2004.

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

[15] L. R. Ford, Jr. and D. R. Fulkerson, "Maximal flow through a network", *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.

[16] P. Elias, A. Feinstein, and C. E. Shannon, "Note on maximum flow through a network", *IRE Transactions on Information Theory* IT-2, pp. 117–119, 1956.

[17] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. Allerton*, Oct. 2003.

[18] "TinyOs", http://www.tinyos.net/.

[19] R. Koetter and F. Kschischang, "Coding for errors and erasures in random network coding", *IEEE Transactions on Information Theory*, vol. 54, no 8, pp. 3579-3591 August 2008.

[20] D. Silva and F. R. Kschischang, "Using rank-metric codes for error correction in random network coding", *IEEE International Symposium on Information Theory (ISIT)*, pp. 796-800, Nice, France, June 2007.

[21] M. Jafari Siavoshani, C. Fragouli, and S. Diggavi, "Noncoherent multisource network coding", *IEEE International Symposium on Information Theory (ISIT)*, pp. 817–821, Canada, Toronto, July 2008.

[22] M. Jafari, S. Mohajer, C. Fragouli, and S.Diggavi, "On the capacity of noncoherent network coding", *IEEE International Symposium on Information Theory (ISIT)*, pp. 273-277, 2009.

[23] D. Silva, F. Kschischang, and R. Koetter, "A rank metric approach to error control in random network coding", *IEEE Transactions on Information Theory*, vol. 54, no. 9, pp. 3951–3967, 2008.

[24] A. Montanari and R. Urbanke, "Coding for network coding", 2007, Online Available: http://arxiv.org/abs/0711.3935/.

[25] D. Silva, F. R. Kschischang, and R. Kotter, "Communication over finite-field matrix channels", *IEEE Transactions on Information Theory*, vol. 56, no. 3, pp. 1296-1305, 2010.

[26] K. Price and R. Storn, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[27] M. Jafari Siavoshani, L. Keller, C. Fragouli, K. Argyraki and S. Diggavi, "Compressed network coding vectors", *IEEE International Symposium on Information Theory (ISIT)*, pp. 109-113, 2009.

[28] F. J. MacWilliams and N. J. A. Sloane, "The theory of error-correcting codes", North-Holland Mathematical Library, 1977.

[29] V. D. Goppa, "A new class of linear error-correcting codes", *Probl. Peredach. Inform*, vol. 6, no. 3, pp. 24–30, 1970.

[30] I. S. Reed and G. Solomon, "Polynomial codes over certain finite field", *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, vol. 8, pp. 300–304, 1960.

[31] V. D. Goppa, "Codes associated with divisors", *Probl. Peredachi Inform.*, vol. 13, pp. 33–39, 1977. Translation: *Probl. Inform. Transmission*, vol. 13, pp. 22–26, 1977.

[32] R. E. Blahut, "Algebraic codes for data transmission", Cambridge University Press, 2003.

[33] Edited by V. S. Pless and W. C. Huffman, "Handbook of coding theory", North-Holland Mathematical Library, 1998.

[34] E. R. Berlekamp, "Nonbinary BCH decoding", *IEEE Transactions on Information Theory*, vol. 14, no. 2, 1968.

[35] J. L. Massey, "Shift-Register synthesis and BCH decoding", *IEEE Transactions on Information Theory*, Volume 15, Issue 1, 1969.

[36] N. Patterson, "Algebraic Decoding of Goppa Codes", *IEEE Transactions on Information Theory*, vol. 21, no. 2, pp. 203–207, 1975.

[37] M. Grassl, "Bounds on the minimum distance of linear codes", *Online available at http://www.codetables.de*, Accessed on 2009-01-09.

[38] M. Jafarisiavoshani, C. Fragouli, and S Diggavi, "Subspace properties of randomized network coding", *IEEE Information Theory Workshop (ITW)*, pp. 17–21, Bergen, Norway, July 2007.

[39] M. Jafarisiavoshani, C. Fragouli, S Diggavi, and C. Gkantsidis "Bottleneck discovery and overlay management in network coded peer-to-peer systems", *ACM SIGCOMM Workshop on Internet Network Management*, Kyoto, Japan, August 2007.

[40] M. Jafari Siavoshani, C. Fragouli, S. Diggavi, "On locating Byzantine attackers", *Network Coding Workshop*, January 2008.

[41] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution", *IEEE Infocom*, pp. 2235-2245, vol.4, March 2005.

[42] C. Gkantsidis, J. Miller, P. Rodriguez, "Comprehensive view of a live network coding P2P system", *ACM SIGCOMM/USENIX IMC*, 2006.

[43] M. Wang and B. Li, "R^2: Random push with random network coding in live peer-to-peer streaming", *IEEE Journal on Selected Areas in Communications, Special ISsue on Advances in Peer-to-Peer Streaming Systems*, vol. 25, no. 9, pp. 1612-1666, Dec. 2007.

[44] P. Morandi, "Field and Galois Theory", *Springer*, 1996.

[45] J. Ebrahimi and C. Fragouli, "Multicasting algorithms for deterministic networks", *IEEE Information Theory Workshop (ITW)*, pp. 1-5, January 2010.

[46] J. Ebrahimi and C. Fragouli,, "Algorithms for vector network coding", *accepted to appear in IEEE Transactions on Information Theory, Special Issue on Facets of CODING THEORY: from ALGORITHMS to NETWORKS*, 2010.

[47] S. Jaggi, Y. Cassuto, M. Effros, "Low complexity encoding for network codes," *IEEE International Symposium on Information Theory (ISIT)*, pp. 40-44, 2006.

[48] J. Ebrahimi and C. Fragouli, "On the benefits of vector network coding", *Forty-Eighth Annual Allerton Conference on Communication, Control, and Computing*, 2010.