Network Coding
Exercise session

I. Network coding: basics

1. Recall what is the general principle of NC, and what is its primary goal.

Assume that a number of original packets $M^1, ..., M^n$ are generated by one or several sources. In linear network coding, each packet in the network is associated with a sequence of coefficients $g_1, ..., g_n$ in $\mathbb{F}_{2^s}$ and is equal to $X = \sum_{i=1}^{n} g_i M^i$. The summation has to occur for every symbol position, i.e., $X_k = \sum_{i=1}^{n} g_i M_k^i$, where M_k^i and X_k is the kth symbol of M^i and X respectively. In the example of Figure 1, the field is $\mathbb{F}_2 = \{0, 1\}$, a symbol is a bit, and the linear combination sent by S after receiving $M^1 = a$ and $M^2 = b$ is $M^1 + M^2$ (the + sign here is addition in $\mathbb{F}_2$, i.e., bitwise `xor`).

For simplicity, we assume that a packet contains both the coefficients $g = (g_1, ..., g_n)$, called *encoding vector*, and the encoded data $X = \sum_{i=1}^{n} g_i M^i$, called *information vector* [6]. The encoding vector is used by recipients to decode the data, as explained later. For example, the encoding vector $\mathbf{e}_i = (0, ..., 0, 1, 0, ...0)$, where the 1 is at the ith position, means that the information vector is equal to M^i (i.e., is not encoded).

Encoding can be performed recursively, namely, to already encoded packets. Consider a node that has received and stored a set $(g^1, X^1), ..., (g^m, X^m)$ of encoded packets, where g^j [resp. X^j] is the encoding [resp. information] vector of the jth packet. This node may generate a new encoded packet (g', X') by picking a set of coefficients $h_1, ..., h_m$ and computing the linear combination $X' = \sum_{j=1}^{m} h_j X^j$. The corresponding encoding vector g' is not simply equal to h, since the coefficients are with respect to the original packets $M^1, ..., M^n$; in contrast, straightforward algebra shows that it is given by $g'_i = \sum_{j=1}^{m} h_j g_i^j$. This operation may be repeated at several nodes in the network.

Using the notation and description above:

2. What is the linear system to solve in order to get the original packets in terms of the received encoded (possibly several times) packets? ***(2pts)***

3. What is the condition for this system to be solvable? What is the connection with the min-cut max-flow theorem? ***(2pts)***

4. What are the advantages of random linear network coding? ***(1pt)***

II. Intra- and Inter-session network coding

Figures 1 and 2 below represent 2 wired networks, where each arc is directed and capable of carrying a single packet per unit of time. As in the lecture, the network is considered delay-free, meaning that a packet can go from any node to any other node within one time slot, whatever the number of hops, provided that the packet is not queued. Queuing only occurs by the capacity constraint that each edge can carry only one packet each time slot. There are 2 unicast sessions with source-destination pairs (S_1,D_1) and (S_2,D_2) wanting to transmit a single packet each. The rate obtained by a given destination node is the goodput, i.e., the number of original packets it is interested in (so only packets from its source) it is able to recover per unit of time.

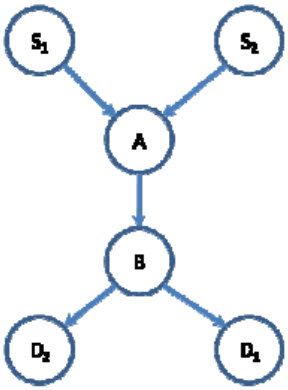

Figure 1

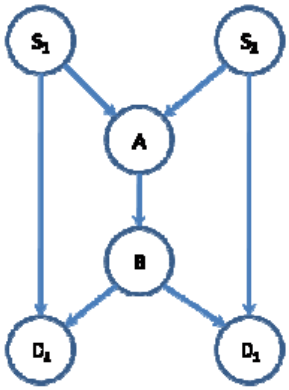

Figure 2

1. Considering the network of Figure 1: What is the average rate (between the two destinations) you can get with single routing?
2. Considering the network of Figure 1: Would **intra**-session NC be possible and useful, and why?
3. Considering the network of Figure 1: Would **inter**-session NC be useful? Why, how much and how?
4. Considering the network of Figure 2: To improve performance, would you implement intra- or inter-session NC? Where? And what is the rate you would get doing so?
5. Considering the network of Figure 2: What additional signaling would such NC require compared with simple routing?
6. Considering the network of Figure 2.2: If the side links are congested, what must we pay attention to when allowing NC?

III. Network coding for DSS

1. Given a certain file retrievable from ***n*** servers, the amount of data being stored at each server being set, what is the most efficient, in terms of failure resilience, to generate the data stored at the servers?
2. We consider a certain file of size ***M*** bits, split into ***k*** fragments of size ***M/k***, and retrievable from any ***k*** of ***n*** servers. When a storing node fails, a new server must be set up so that the number of servers keeps constant. To make a new server fill in the role of the failed one, which of the following statements is true:
   - it is optimal in terms of storage requirement only to send the newcomer ***k(M/k)*** bits so that it can forge a new set of ***M/k*** bits to store.
   - to be able to send the newcomer less than ***M*** bits to forge a new fragment to store, we necessarily need to increase the storage requirement.
   - it is optimal in terms of storage requirement and bandwidth consumption to send the newcomer ***k(M/k)*** bits so that it can forge a new set of ***M/k*** bits to store.
   - if each server stores one fragment of size ***M/k***, then it is sufficient to send the newcomer ***M/k*** bits so that it creates a new fragment to store.

Tip to sum up the intuition behind NC DSS:

Roughly, we can say that if each of the nodes stores more than the minimum ***M/k***, then it has somehow access to more information on the whole file, and within *1* packet sent, this packet contains more info and hence less packets than the total file are necessary to build redundancy at the new node.
More in detail:

- file of size ***M***, made of ***k*** pkts of size ***M/k***
- 1pkt stored at each ***n*** of the storage nodes (***M/k*** bits)
- if proper code, optimal because the file can be recovered from any ***k*** of the ***n*** nodes
- BUT, to set up another storage node, all ***k*** pkts (***M*** bits) must be received just to create ***M/k*** bits (the new pkt). This is the overhead bw.
- to allow to set up a new node with less than the whole file transmitted, each node can store more than 1 pkt (***M/k***), but the total number of pkts required to be sent to the new node is less in total than ***M*** bits: example s29

  -> M=1MB, 4 pkts of size 0.5MB. 2 pkts stored at each node, but only 1.5MB necessary to transfer to set up a new node.

3. We consider a DSS with ***k***=3 fragments and ***n***=6 servers. Such a DSS is partly represented in the figure below. What is the minimum value of β (number of bits transmitted from each server to the newcomer) required to set up a new server ensuring the same DSS properties?