EECS 290S: Network Information Flow **Fall 2008**

Lecture 12 — October 9

Lecturer: Anant Sahai *Scribe: Gireeja Ranade*

12.1 Linear Deterministic Models of Relays

12.1.1 Recap

- We've looked at linear deterministic models of relays with one source and one destination.
- Last time we looked at non-layered models and unrolled them in time to have K stages and showed that the mincut of the resulting layered network was close to K times the mincut of the original graph.
- The proof for the layered case used block codes of length T and pipelining. The reason for having large T was to make the probability of error go to zero since we are using a random-coding argument over coding matrices and something needs to go to infinity to make the probability of error go to zero. However, notice that we didn't actually need the probability of error to go to zero. It just needs to drop below 1 and then since the problem is deterministic, we can conclude that a good code must exist.

  The reason for the pipelining had a physical interpretation. We had to make sure that all the nodes had something to do. So while one layer was working on transmitting one message, the previous layer could be transmitting the next message. Since nodes are assumed to be able to transmit and listen at the same time, they are listening to the block corresponding to one message while transmitting encoded information corresponding to the the previous one.

12.1.2 Details from the proof for cyclic/non-layered linear relay networks

Unroll the graph in time for a general network:

Let S be the source node, A, B, C be relays, and D be the destination node. The unrolled graph is shown in Figure 12.1.2. The horizontal infinite capacity links confer memory on each of the nodes. Let the infinite capacity links from all of the relays be LSB's below the noise floor, which ensures that only the S_i's will see the bits that the source is sending itself. Let's compare the layered and the general cases (Table 12.1.2). Note, T is the block length of time, K is the number of stages, $\tau(t)$ is the cyclic time function, and $m(t)$ is the message being transmitted at time t.

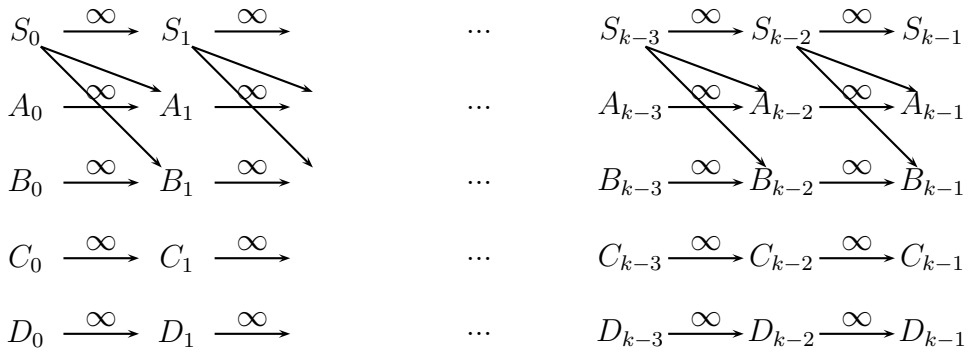


Figure 12.1. Unrolling a general network

	Layered Graph: T	**Cyclic/General Graph:** K, T
Messages	(linear time) 0, 1, 2, ...	(linear time) 0, 1, 2, ...
Cyclic time blocks (τ):	0, 1, 2 ... ,T-1, 0, 1, 2 (mod T)	0, 1, 2 ... ,T-1, 0, 1, 2 (mod T)
	Pipelining: Different layers talk about different messages at one time	Stages: 0, 1, 2, ... K-1

Table 12.1. Comparing the layered and general cases

Unrolling a general graph leads to the questions: Which happens first, T or K?

Notice that there is no need for pipelining messages in the general case since there is no problem of idling nodes. Every physical node is active at every "layer" in the unrolled graph. Since we are not pipelining our messages for the general case, a constraint that is imposed on the unrolled network is: Every node must be at the same stage $k(t)$ at a real time t. So at a time t, we can do the following:

$$\tau(t) = t \pmod{\text{T}} \tag{12.1}$$

$$k(t) = \lfloor \frac{t}{T} \rfloor \pmod{\text{K}} \tag{12.2}$$

$$m(t) = \lfloor \frac{t}{KT} \rfloor \tag{12.3}$$

$$\tag{12.4}$$

This ensures causality, and thus we can implement the scheme. We send one message every KT steps, and the size of our message set is thus 2^{RKT}. Do we really need two kinds of cyclic time? Clearly, unrolling the graph to K stages is essential to maintain the parallel to the layered case, but do we need T?

Remember, in the layered proof we had:

$$P_e \leq (\text{no. of cuts}) \cdot 2^{-(\text{mincut}-R)T} \tag{12.5}$$

This comes from calculating the probability of a code having an error as being (by the union bound and linearity) no more than 2^{RT} times the probability of any individual non-zero message showing up at the destination as all zeros. This event was further broken up as the disjoint union of all the ways that zeros could be received at the intermediate relay nodes. The probability of each of these events could then be written out as a product: the probability of getting a zero at the zero nodes conditioned on getting something nonzero at the nonzero nodes times the probability of getting something nonzero at the nonzero nodes. The second probability could then be upper-bounded by 1. The conditional probability for a random linear code was then just exponentially small in the value of the cut represented by this particular partition between zero and nonzero nodes. All these exponentials are dominated by the slowest ones and this gives the above expression.

So, we needed to make T large to counter the large number of possible cuts and make P_e smaller than 1. In the general case with a K-unrolled network we have:

$$P_e \leq (\text{no. of non-infinite cuts}) \cdot 2^{-(\frac{(K-2)\text{mincut}}{K} - R)KT} \tag{12.6}$$

We don't have to worry about the infinite-valued cuts since those are never a source of error. To answer our question, we will count the number of non-infinite cuts to see how large it is. Follow these steps:

1. Unroll

2. Prune: Remove those nodes that are never affected by the source, as well as those that never talk to the destination. The maximum delay between any node and the source is $|V|$, the number of vertices in the graph. So after $|V|$ stages in the unrolling, everyone will have information from the source (note: the graph is connected). Similarly, only at most the last $|V|$ stages can contain nodes that are irrelevant to the destination.

3. Count: The number of cuts of the original graph is $2^{|V|-2}$. These correspond to straight cuts in the unrolled graph (i.e. cuts that don't cross any of the new infinite horizontal edges). Remember, that we are allowed to have dipping cuts in the unrolled graph, but any rising cut would have an infinite value.

   A dipping cut can be thought of as a list of straight cuts with a number of downward transitions between them. But there are at most $3|V|$ places where such transitions could occur. Thus, we can specify each non-infinite cut by an ordered list of length $3|V|$ of the straight cuts that comprise it and then a list of transition points. Thus the number of cuts is:

$$\begin{aligned} \text{number of cuts} &= (2^{|V|-2})^{3|V|} \cdot K^{3|V|} && (12.7) \\ &= O(K^{3|V|} \cdot 2^{3|V|^2}) && (12.8) \end{aligned}$$

   But this is only polynomial in K and the dying exponential term in the error probability will kill it as K grows larger. Thus we can choose $T = 1$ and don't need to have T grow larger.

Observe that to make the exponential be negative requires a K that grows like the number of nodes times the reciprocal to the gap between the rate R and the mincut. However, K must be even bigger to overpower the $2^{3|V|^2}$ possible cuts in the union bound. So a K that is roughly proportional to something greater than the square of the number of nodes is enough. So the memory requirement only scales polynomially in the size of the network.

Contrast this to what is required in the original proof of Avestimehr and Tse. In that proof, making the exponential be negative requires K to be proportional to $2^{|V|}$ times the reciprocal to the gap between R and mincut. The number of possible cuts is then exponential in K and hence double-exponential in $|V|$. Overcoming this requires a T that is therefore at least exponential in $|V|$ as well. Thus, even though the original construction attempts to restrict the need for nodes to have memory, the implicit memory requirements from the proof are much larger than what we get if we allow ourselves to take explicit advantage of memory.

12.2 Generalized Models of Relay Networks

Here we consider some generalizations of the linear deterministic single source single destination case.

12.2.1 Multiple Destinations: Multicast

We now generalize the problem to include multiple destinations that want to receive the same message. We would like to achieve a rate equal to the $\min_i \text{mincut} D_i$ - can we do this?

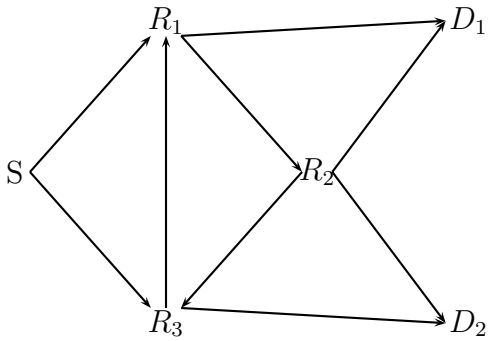


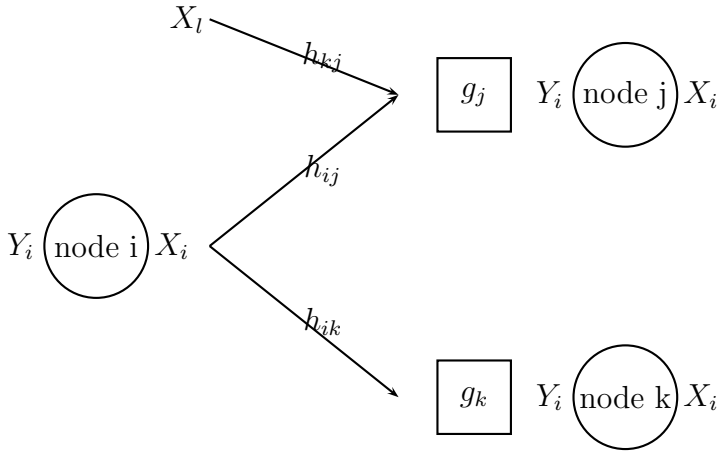
Figure 12.2. Multicast network

Our previous proof for the single destination case was based on generating random coding matrices for each relay. However, we never used any knowledge about the destination in the proof. Thus, we expect that generating random coding matrices for each of the relays should also work in the multicast problem. Consider a setup with two destinations as in Figure 12.2.1. We know that the probability of bad coding matrices (i.e. coding matrices that take

a nonzero-message to zero) for $D_1 \to 0$, as does the probability of bad matrices for D_2. Thus, the probability of bad matrices for either of D_1 or D_2 tends to 0 and thus the probability of simultaneously good matrices tends to 1.

12.2.2 Losing Linearity

We would also like to lift our linearity assumption. Our linear model employs both a linear coding strategy and assumes linear superposition. To make our models more realistic, could we allow for non-linear-superposition, or just superposition with carry's? Let each node have input Y_i and output X_i, and allow for both nonlinear transformations by the edges between two nodes (call these h_{ij}) and for non-linear superposition at the nodes (represented by g_j). We can easily just absorb the h_{ij}'s into the g_j's and there is thus no reason to explicitly consider them.



While the g_i's (and the h_{ij}'s) are fixed, we have the freedom to choose an encoding function f_i, at each node. Let us restrict our attention to random f_i's, with codewords drawn iid across time with probability $p_i(x)$ that might vary from node to node.

How do we define a cut and capacity for such a network? It is tempting to define the value of a cut as $H(\vec{Y_{cut,R}})$. However, we care about the information flowing across a cut from the source side to the destination side and don't want to be confused by the information going in cycles on the destination side of the cut.

Hence, we define the value of a cut as: $H(\vec{Y}_{cut,R}|\vec{X}_{cut,R})$. We can see this is the same as:

$$\begin{aligned} I(\vec{X}_{cut,R}; \vec{Y}_{cut,R}|\vec{X}_{cut,R}) &= H(\vec{Y}_{cut,R}|\vec{X}_{cut,R}) - H(\vec{Y}_{cut,R}|\vec{X}\text{'s from whole system}) \quad (12.9) \\ &= H(\vec{Y}_{cut,R}|\vec{X}_{cut,R}) \quad (12.10) \end{aligned}$$

since $(\vec{Y}_{cut,R}|\vec{X}$'s from the whole system) is deterministic and hence its entropy is zero.