Lecture 26

Note Title 4/16/2012

* LT code constuction:

① Start from k $\boxed{\text{info bits}}$, k is usually ≥ 1000.

② A prob distri $p_1 \cdots p_k$ is predefined $\sum_{i=1}^{k} p_i = 1$

③ Whenever we can transmit a bit (a packet)

choose independently & randomly a "d" value based on the distribution $\{p_i\}$.

④ Choose randomly d distinct bits.

$X_{i_1}, X_{i_2}, X_{i_3}, \cdots X_{i_d}$

& send $Y_t = X_{i_1} + X_{i_2} + \cdots + X_{i_d}$

$+$ ↳ binary XOR

Our goal is when receiving $Y_1, \cdots, Y_k$ we can decode $X_1, \cdots, X_k$ efficiently.

Q: How to decode ?

A: Once we receive k coded bits, we have

$$\boxed{\quad H \quad} \vec{x} = \vec{y}$$

Decoding is equivalent to solving the equation:

Solution 1: Inversion, too complicated.

☆ Solution 2: Message passing (look for the rows of deg 1)

$$\boxed{\quad H_1 \quad} \vec{x}_{erased} = \vec{y} + H_0 \cdot \vec{x}_{recovered}$$

An alternative view that is convenient for later analysis.

k info bits.

Consider two types of partitions that evolve during decoding

$\mapsto$ during decoding

* | recovered | Not recovered |

(also known as "covered")

$\mapsto$ during decoding

* | processed | unprocessed |

processed: those values of bits that have been used to derive other bits. (those bits that have been moved to the RHS of ⊕

* The intersection of recovered & unprocessed bits is called "ripple", which contains new useful bits that might help further decoding.

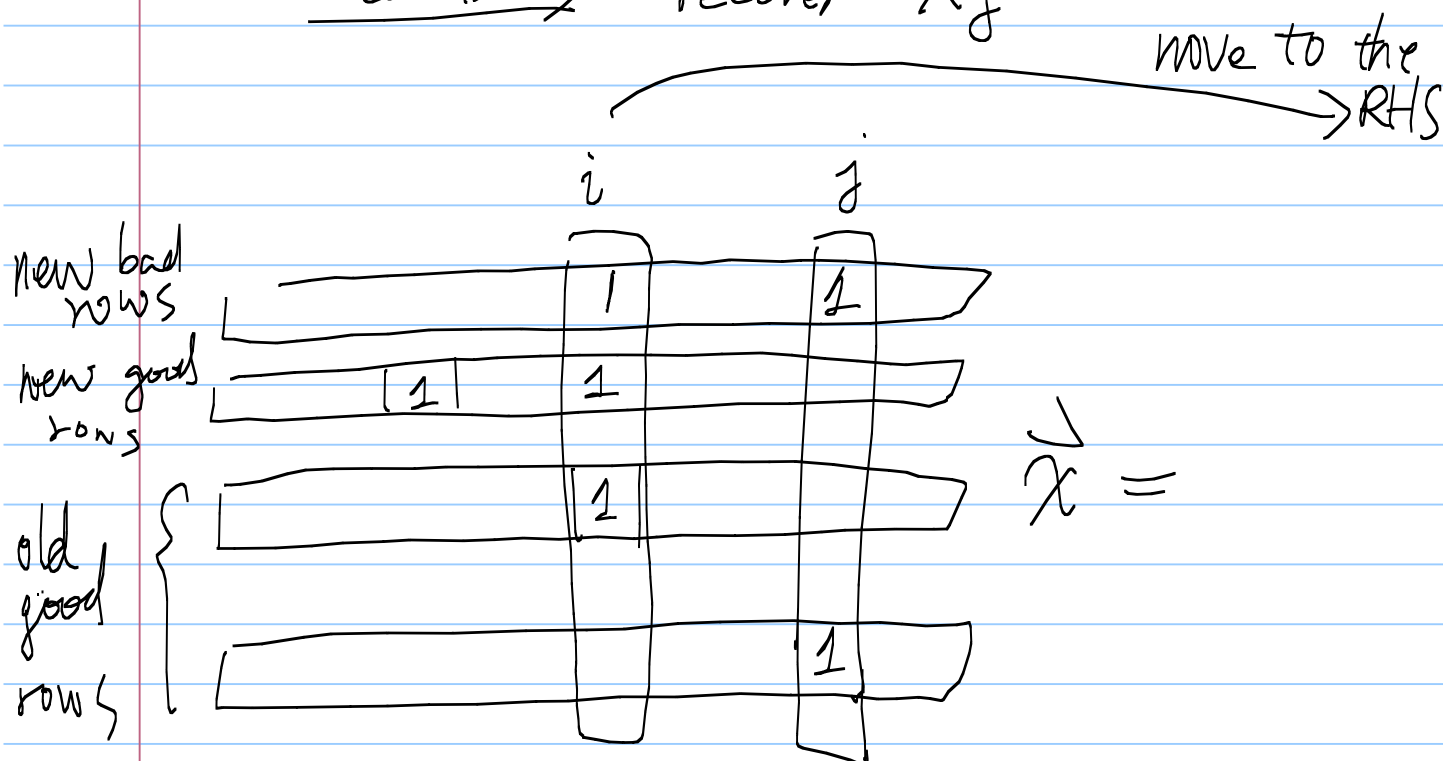
$\mapsto$ $\mapsto$

| | ripple | |

*Q: When can we guarantee decoding $X_1, \cdots, X_k$ from $Y_1, \cdots, Y_k$.

Ans: A heuristic observation:

Suppose the "ripple" contains two bits i, j. If we decide to process X_i & move it to the right. Then we will create some new good rows that recover an additional bit, But at the same time we may also create some new bad rows that <u>redundantly</u> recover X_j



$\Rightarrow$ We lose some valuable rows.

In sum,

$\Rightarrow$ We would like to keep the size of the ripple to be exactly one. So that no new bad rows will be created. (Very similar to the area theorem argument. so that the improvement is infinitesimal.

* On the other hand, we also like to keep the ripple from destroying itself. Namely when we move the only bit X_i in the ripple to the RHS, we want to replenish the ripple with exactly 1 bit

The two goals of

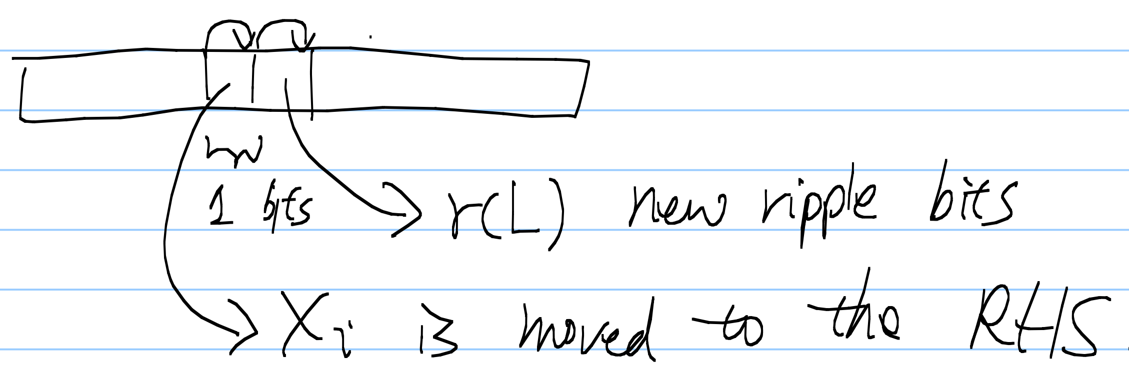
① ripple size $= 1$.

② ripple size is self sustainable

lead to the design guideline of p_i, $i=1, \cdots, k$

Def:
Let $r(L)$ denote the expected amount of new ripple bits that will replenish the old ripple bits when there are L unprocessed bits & when we move one old ripple bit to the RHS.

Goal: $r(L) = 1$ for $L = 1, \dots, k$



$\ast$ When $L = k$. $\boxed{\text{unprocessed}}$

$\because r(k) = 1 \Rightarrow \underline{\underline{P_1}} \times \underline{\underline{k}} = 1$

Totall k coded bits available at decoding

P_1: the prob of choosing 1 input bit & generate $Y_t = X_i$ for each Y_t

$$\Rightarrow p_1 = \frac{1}{k}$$

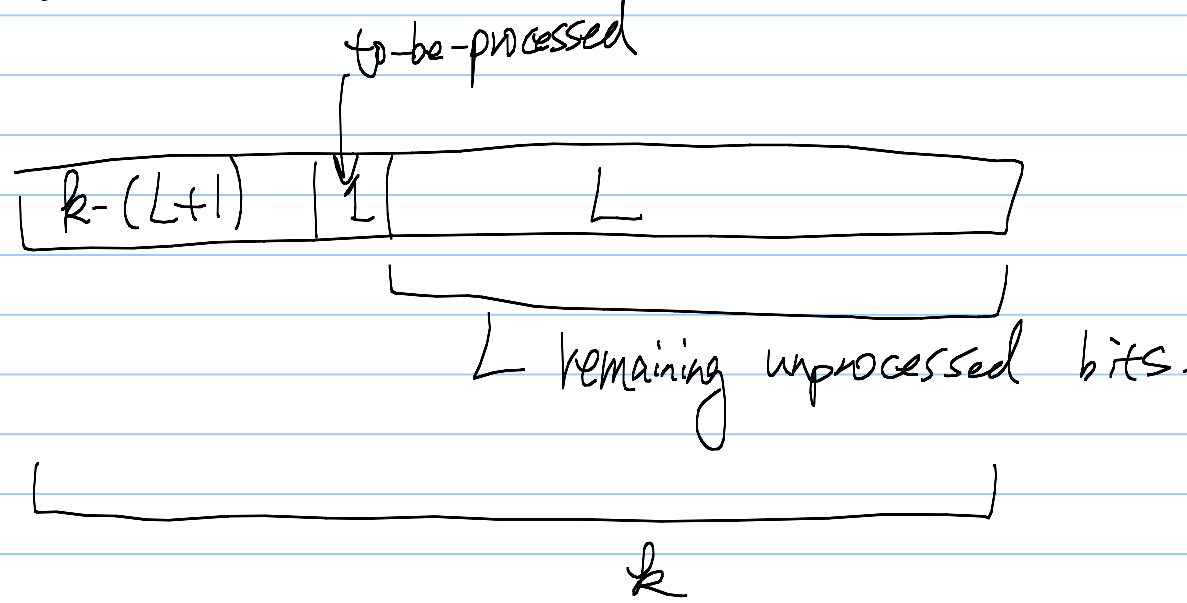
for $L = k-1, \cdots, 1$

* $r(L) = \underbrace{k}_{\text{\# of coded symbols}} \sum_{d=1}^{k} \underbrace{P_d}_{\text{prob that 1 coded symbol is of degree } i} \cdot \Big(P(\text{a degree } d \text{ coded symbol contributes to a new ripple bit when there are } \underbrace{}_{\text{remaining}} \text{ unprocessed bits.}) \Big)$

* $P($a degree d coded symbol contributes to a $\underline{\text{new}}$ ripple bit when there are L remaining unprocessed bits$)$

$$= \begin{cases} \dfrac{L \times \dfrac{(k-(L+1))!}{(k-(L+1)-(d-2))!\,(d-2)!}}{\dfrac{k!}{(k-d)!\,d!}} & \text{if } 0 \le d-2 \le k-(L+1) \\ 0 & \text{otherwise} \end{cases}$$

$(i-2)$ of the connections are in the processed portion



By solving the linear equations

$$r(L) = 1 \quad \text{for } L = k-1, \cdots, 1$$

We have

$$P_1 = \frac{1}{k}.$$

$$P_d = \frac{1}{d} \times \frac{1}{d-1} \quad \text{for } d = 2, \cdots, k$$

Verification:

$$k \cdot \sum_{d=2}^{k-L+1} P_d \cdot \frac{L \times \dfrac{(k-(L+1))!}{(k-(L+1)-(d-2))!\,(d-2)!}}{\dfrac{k!}{(k-d)!\,d!}}$$

$$= \sum_{d=2}^{k-L+1} \frac{\dfrac{(k-d)!}{(k-d-(L-1))!\,(L-1)!}}{\dfrac{(k-1)!}{(k-1-L)!\,L!}}$$

$$= \sum_{d=2}^{k-L+1} \frac{\binom{k-d}{L-1}}{\binom{k-1}{L}}$$

Using $\binom{a}{b} + \binom{a}{b+1} = \binom{a+1}{b+1}$

we have $= 1$

* Nonetheless, when $E(r(L)) = 1$, the ripple may easily vanish due to the randomness of the system.

Example: Even the 1st ripple bit is hard to generate. For $k = 1000$, with prob $= (1 - \frac{1}{k})^k \approx 37\%$ we don't even have the first ripple bit.

* In pratice, we choose

$$r(k) = (\log k) \cdot \sqrt{k}$$ to ensure

& $r(L) = 1$ for $L = k-1 \ldots 1$

a large enough (although suboptimal) ripple.

to accommodate the randomness, but at the same time having $\frac{r(k)}{k} \to 0$

so that the loss of efficiency is under control

Q: Why square root? The variance of the random walk is $\sqrt{k}$. So that the variance won't destroy the ripple.

* New $P_1, \ldots, P_k$ can be solved by ② (Note the previous computation needs to be significantly revised as how we need to take care of the "new" ripple bits.

* Network coding: A similar concept but for a setting that is quite different from the fountain codes.

* Q: Why do we need a large # of $k \simeq 1000$ info bits?

A1: To achieve the capacity efficiently. (the averaging effect kicks in at a much faster rate.

$k = 30 - 100$ should be sufficient

A2: To reduce the # of ACK packets.
(A more important reason why focusing on Long-range, large-file transmission.)

However for a shorter range communication say a WMN, we do not need to use a large k. We can use a smaller $k = 32$.

* The immediate benefit is that for small k we can now use the optimal erasure decoder

$$\boxed{H} \; X_{info} = Y_{received}$$

that inverts the matrix equation

* Nonetheless when k is small, say $k=2$, then we may not generate a long stream of coded packets that has the desired property that any k pkts are sufficient for decoding

Example: $k=2$, binary symbols

$X_1 \quad X_2$

$Y_t = X_1, \quad X_2, \quad X_1+X_2,$

must repeat one of the previous three codes symbol. Say we send

$$\underbrace{X_1, \; X_2, \; X_1+X_2, \; X_1}, \; ,$$

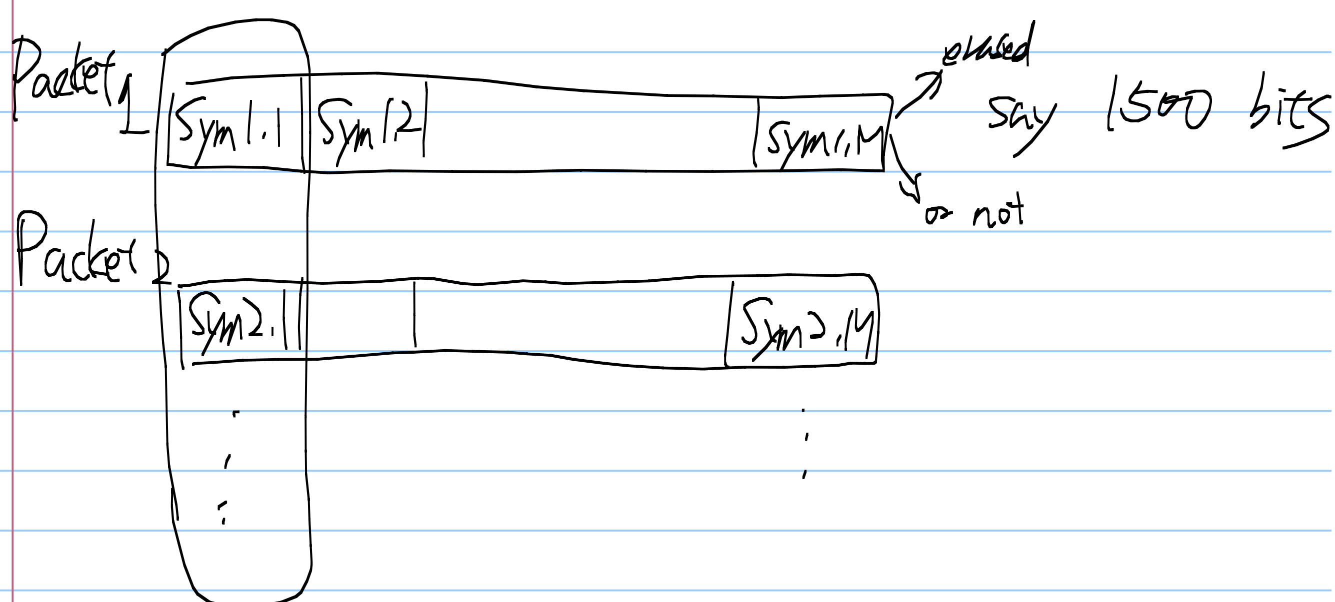
Then receiving $k=2$ coded pkts X_1, X_1 does not guarantee decodability.

Solution: Increase the symbol size.

$$X_1, \; X_2, \; X_1+X_2, \; X_1+2X_2, \; X_1+3X_2, \ldots$$

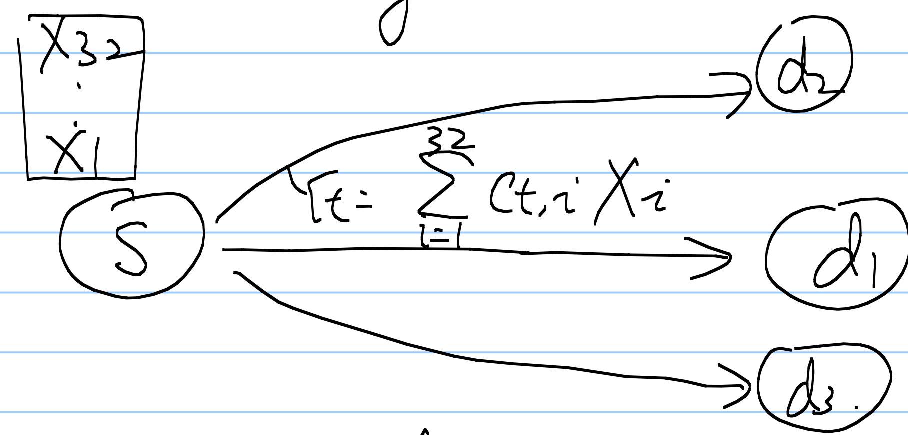
* Network coding focuses on medium $k = 30-100$ & large symbol size $GF(2^8)$

Recall



$\hookrightarrow$ Symbol erasure channel

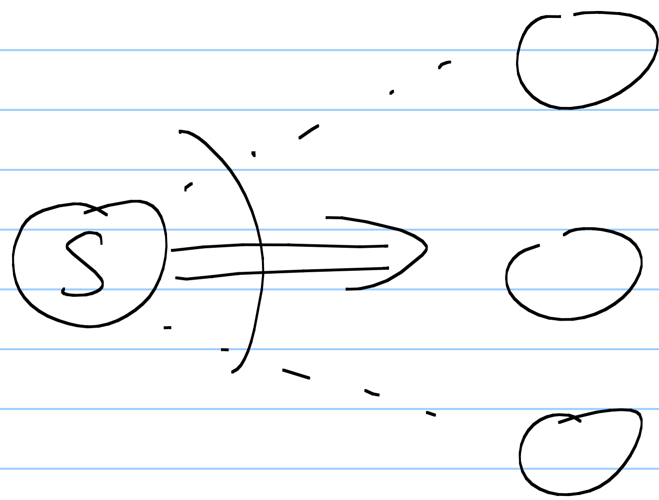
* Network coding-based fountain codes



by randomly choosing the coeff $C_{t,i}$.

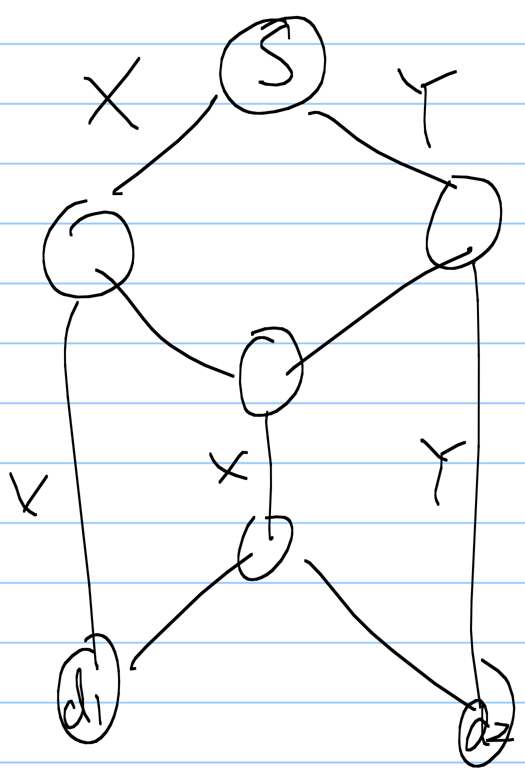
We can decode X_i from Y_t

* And we can focus on wireless broadcast channels.



* The throughput advantage of network coding & its main distinction from fountain codes.

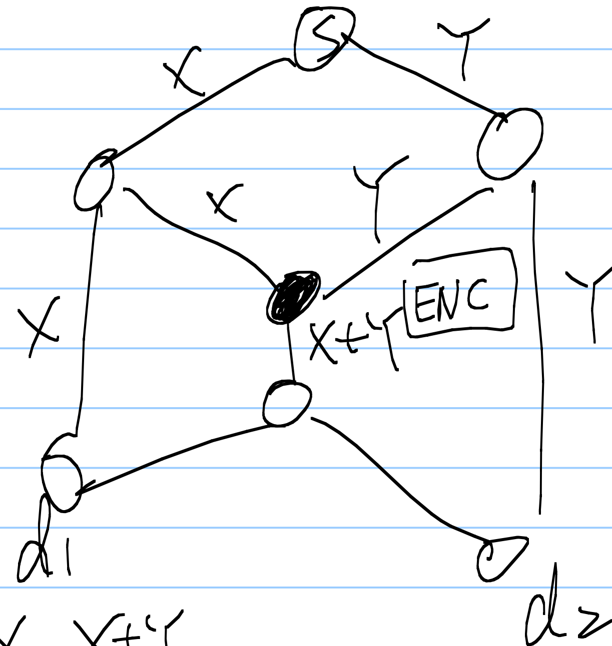
The butterfly example



We would like to send two packets X & Y to both d_1 & d_2. A multicast session $(s, \{d_1, d_2\})$

* Pure routing each time only 1 receiver can receive 2 pkts & the other receiver can only receive 1 pkt. ⟹ The overall rate is 1.5 pkt/slot.

* Network coding



by performing XOR at ● the rate is enhanced to 2 pkt/slot.

X, X+Y

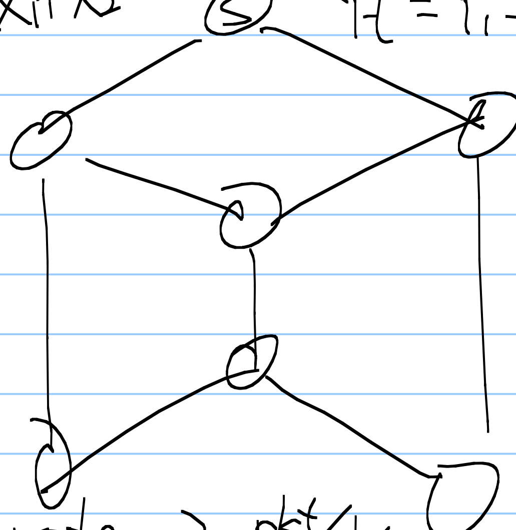
DEC X, Y

X+Y, Y

X, Y

* Can we use fountain codes at the sources to achieve the same throughput

$X'_t = X_1 + X_2 \cdots$ S $Y'_t = Y_1 + Y_2 + \ldots$



Ans: No, it does not increase the throughput

To achieve rate 2 pkt/slot coding needs to be performed at the intermediate node ● which coins the term "network coding"