Fountain Codes

Amin Shokrollahi
EPFL

Access high-quality content available anywhere in the world

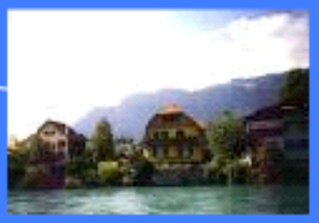
quickly

easily

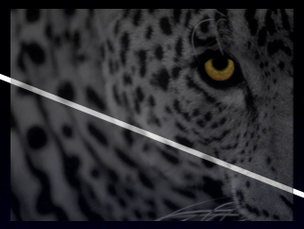
reliably

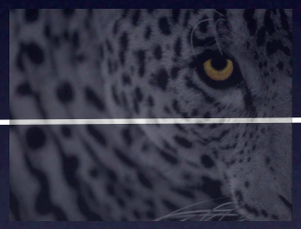
Point-to-Point Communication

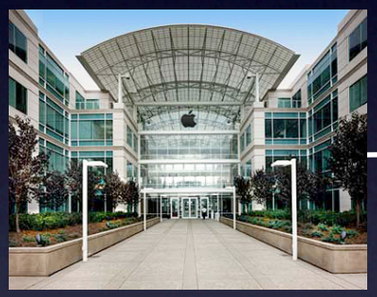
Point-to-Multipoint Communication



Multipoint-to-Point Communication

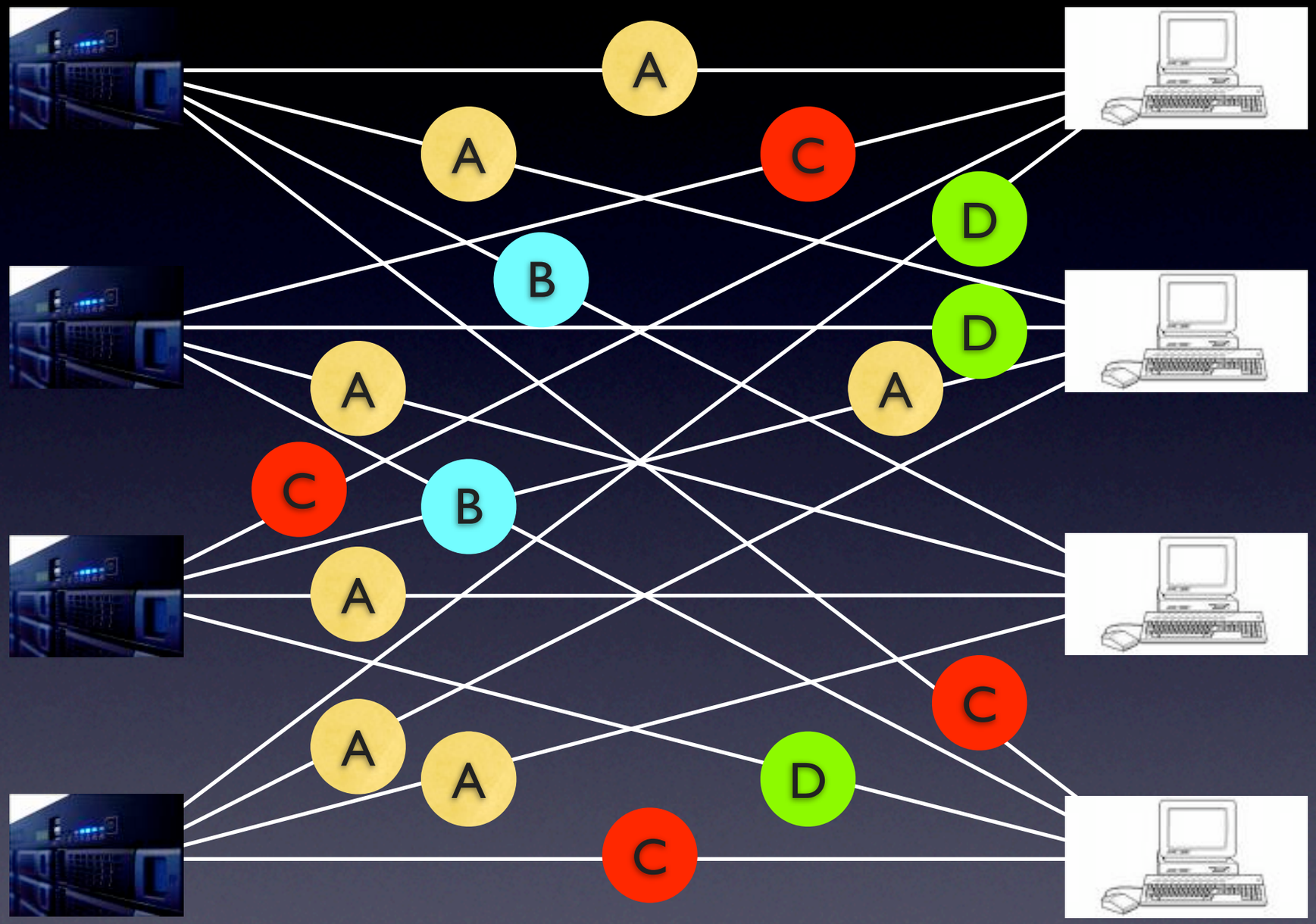



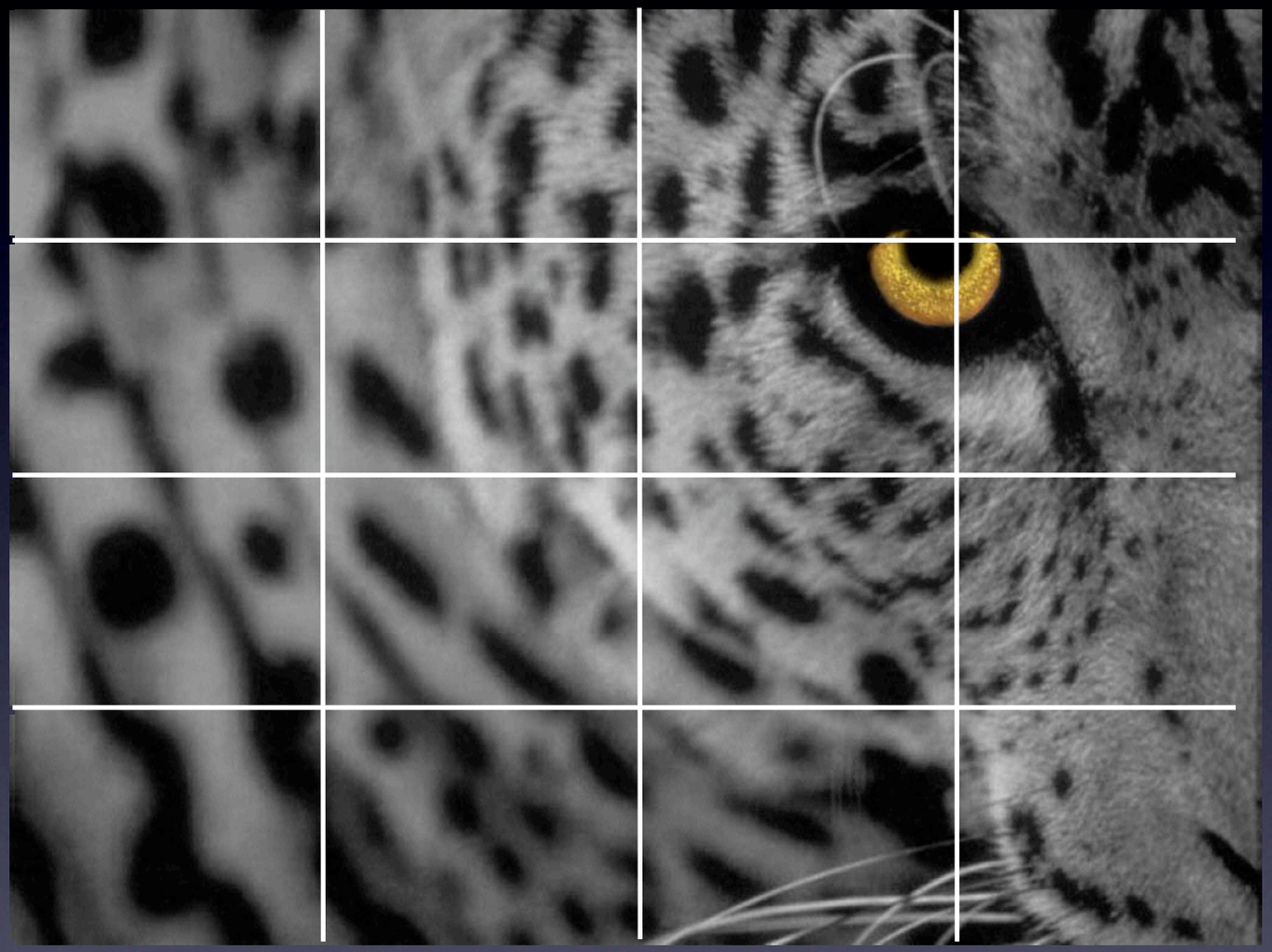


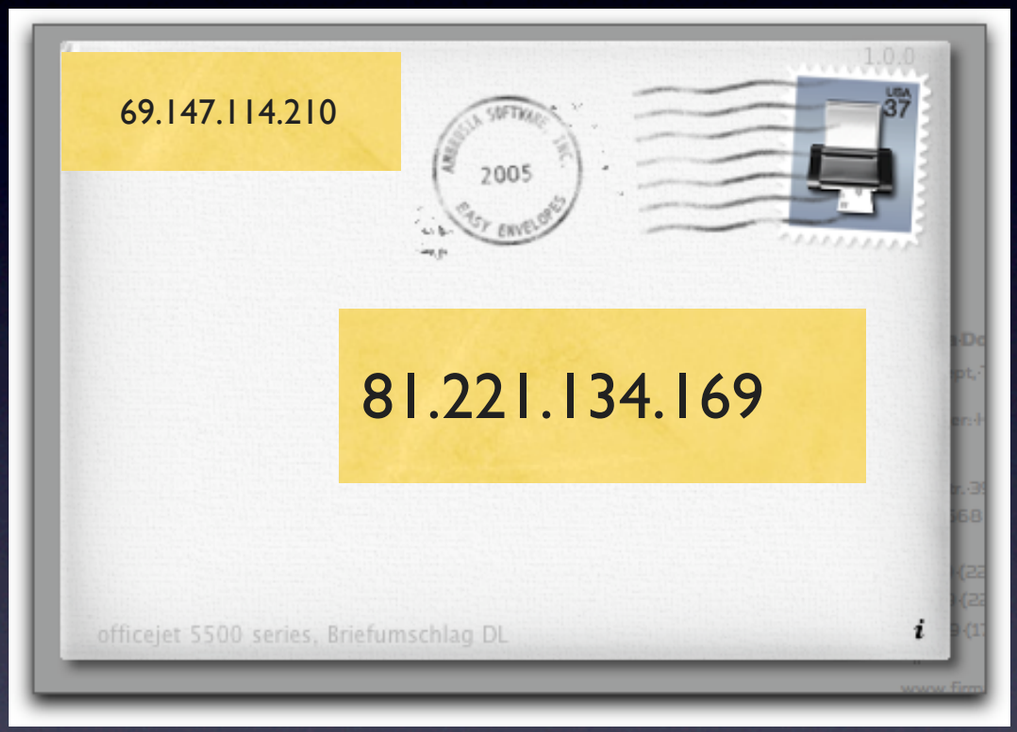


Multipoint-to-Multipoint Communication

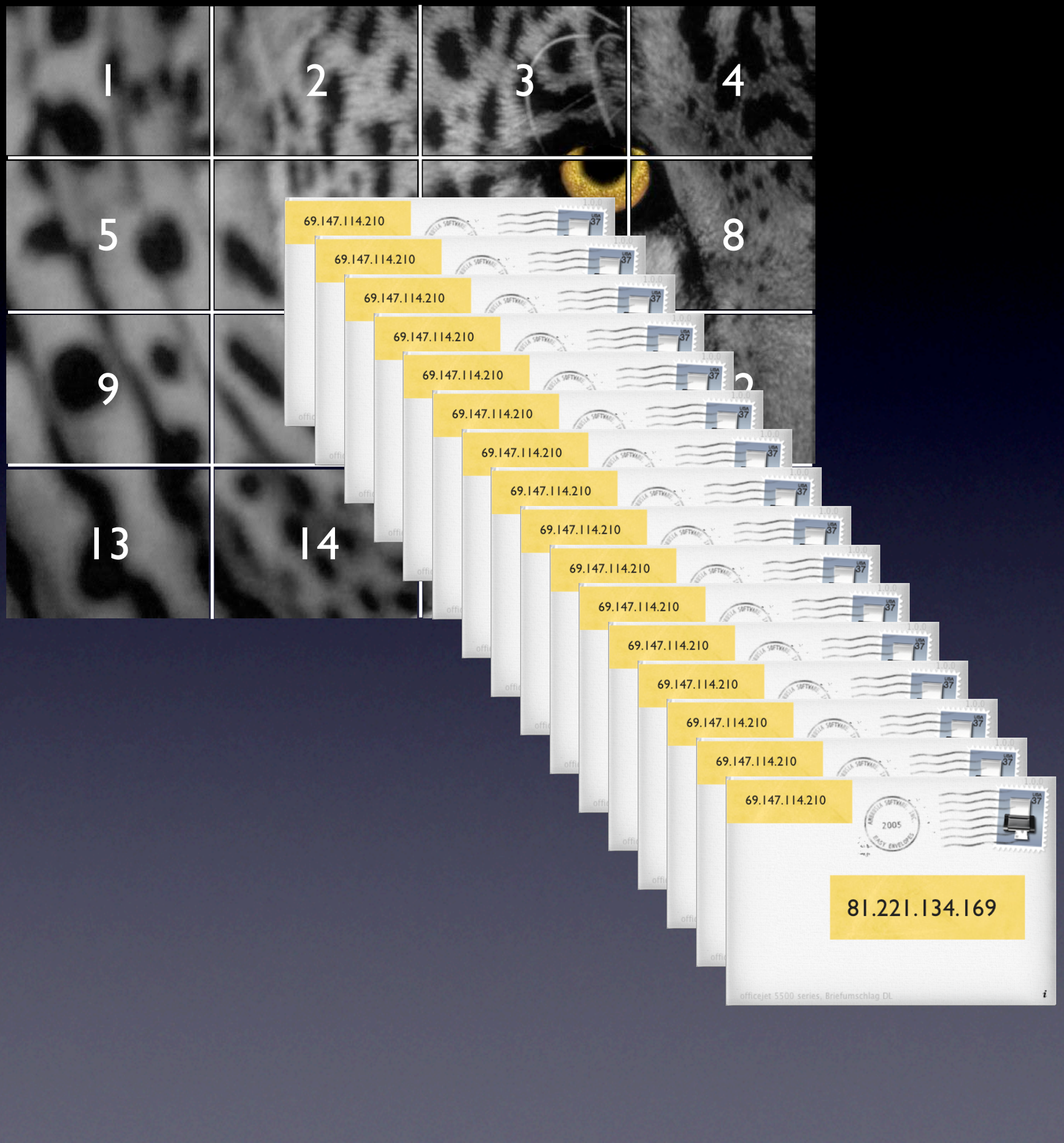


Source: www.blythewood.co.uk
www.chanserve.com




69.147.114.210
2005
81.221.134.169
officejet 5500 series, Briefumschlag DL

Packet


1
2
3
4
5
8
9
13
14
69.147.114.210
81.221.134.169
officejet 5500 series, Briefumschlag DL

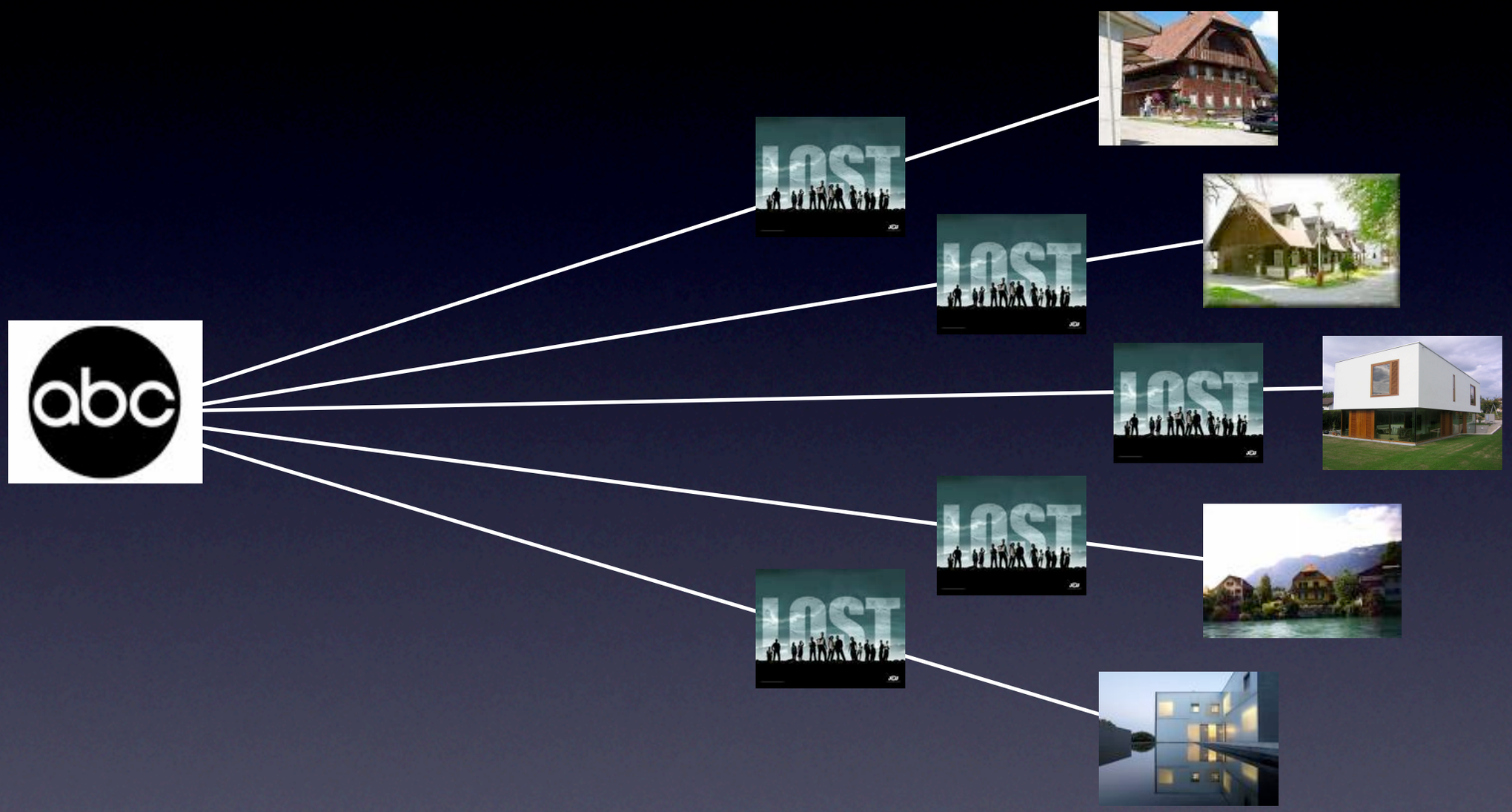




Point-to-Point Communication: TCP

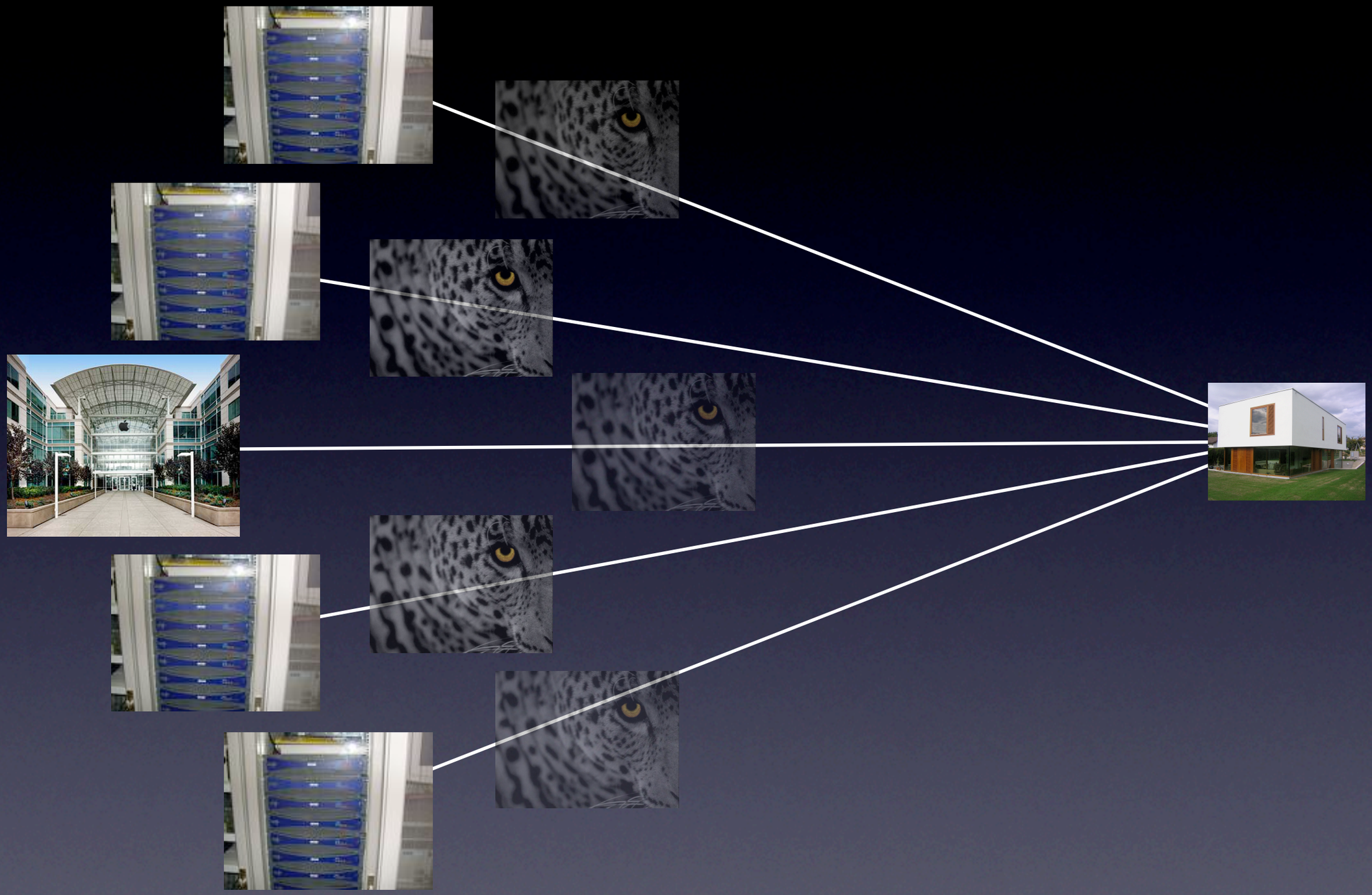
Inefficient if distance is large

Point-to-Multipoint Communication: TCP



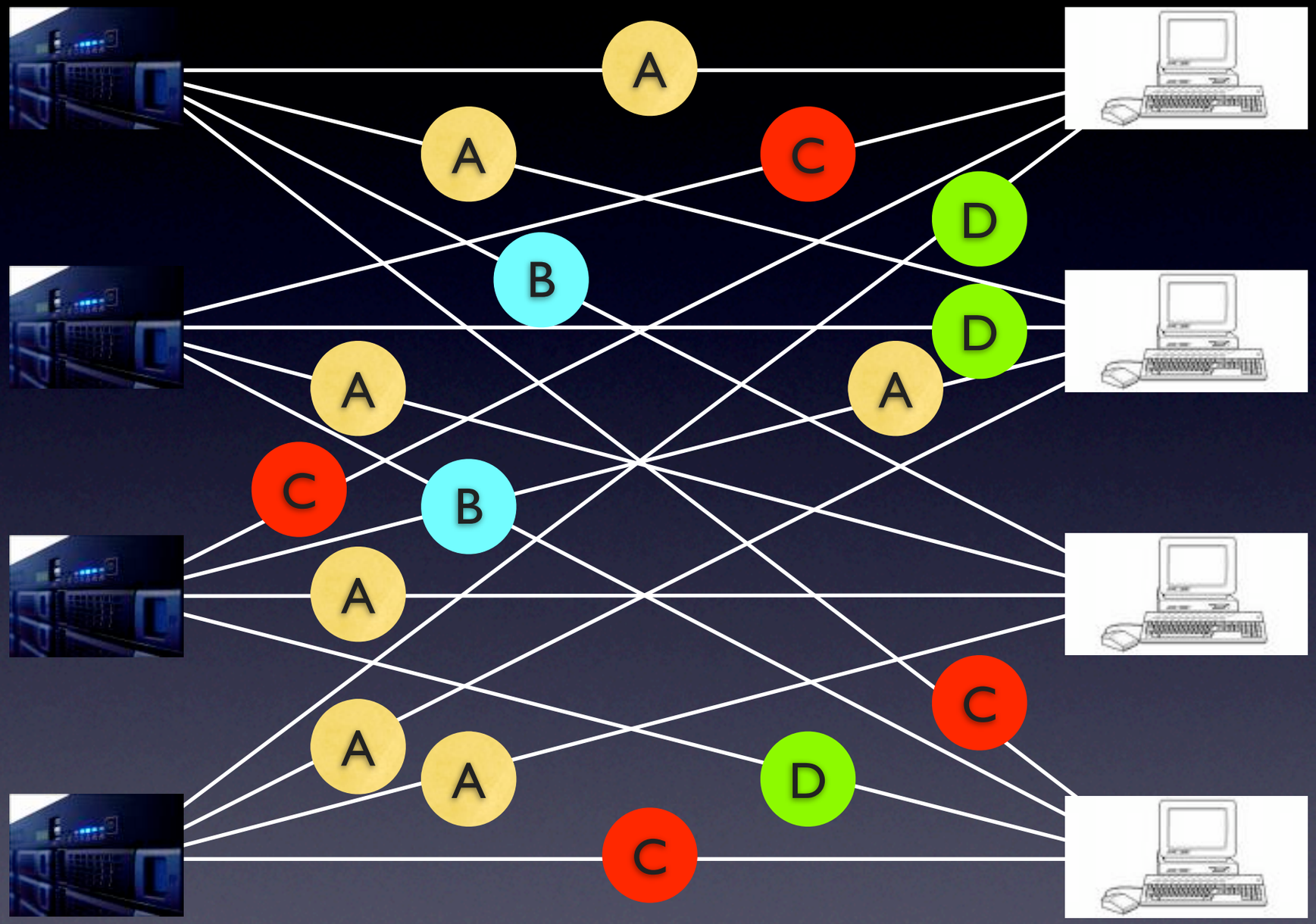
Is not scalable

Multipoint-to-Point Communication: TCP



Not scalable, and needs management

Multipoint-to-Multipoint Communication: TCP



Not scalable, needs management

Fountain

Generates for a given piece of data a potentially limitless stream of packets such that:

1. Each packet is generated independently of any other packet.

2. It is sufficient to collect any set of packets that is in aggregate of the same size as the original piece of data for recovery.

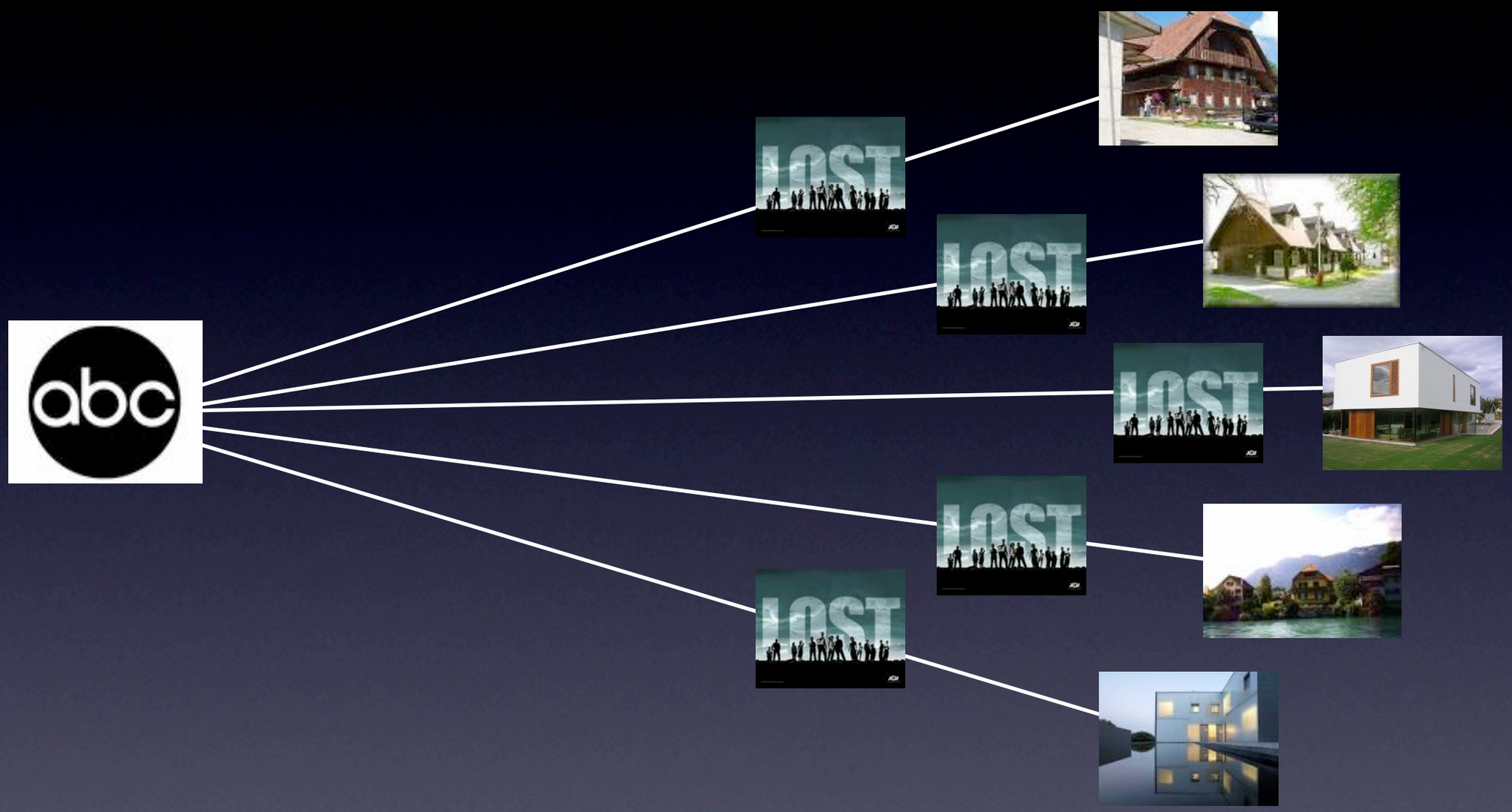
3. Generation and recovery are very efficient.

Source: http://lisa.innereyes.com/wp-photos

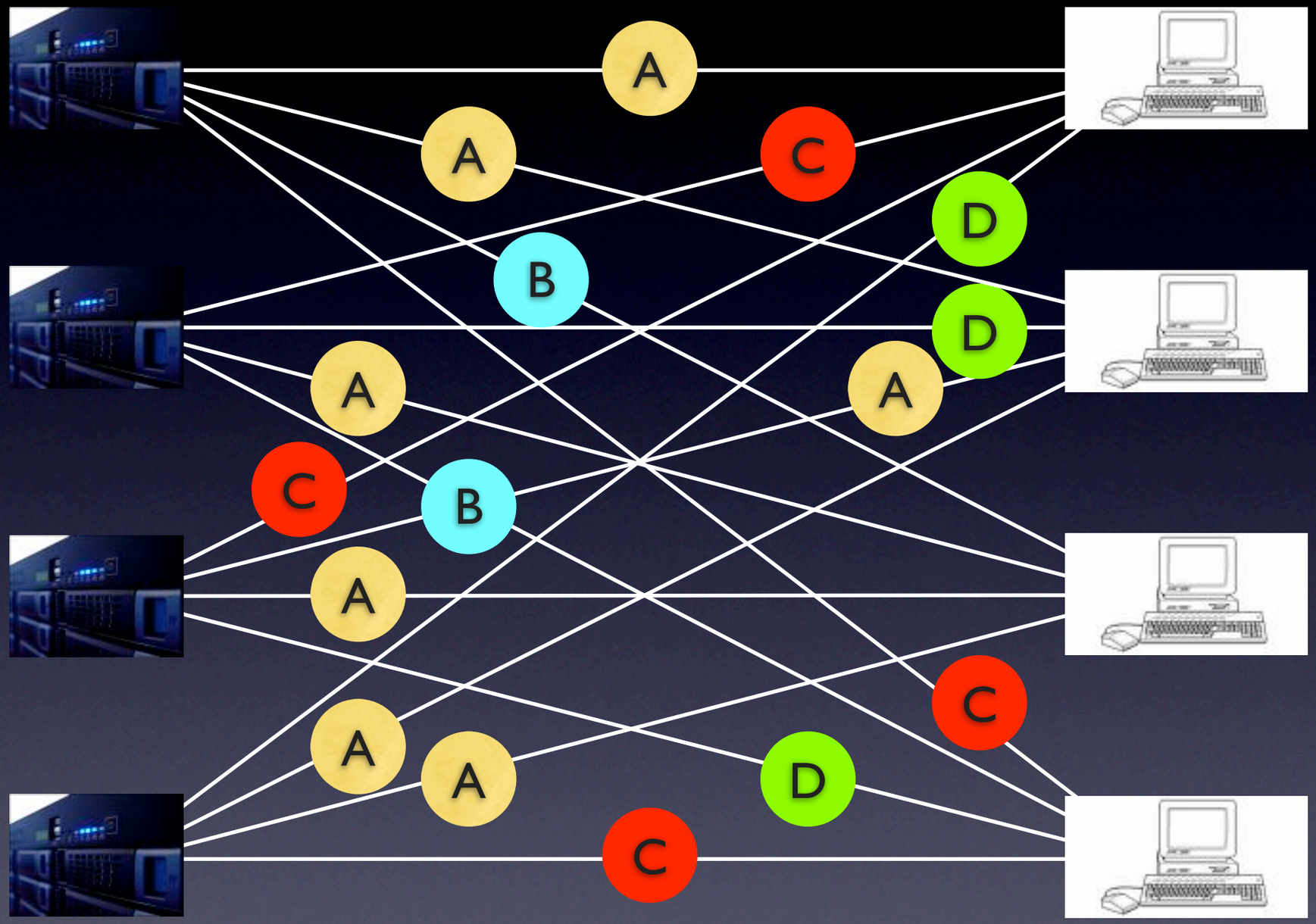




Point-to-Multipoint Communication: TCP



Multipoint-to-Multipoint Communication: TCP



Summary and History

Fountain codes are a class of codes designed for solving various data transmission problems, at the same time.

Fountain codes with fast encoding and decoding algorithms, and (arbitrarily) small overhead are particularly interesting for solving these problems.

Fountain codes were stipulated by Byers et al in 1998, and their applications discussed. A construction was, however, not given.

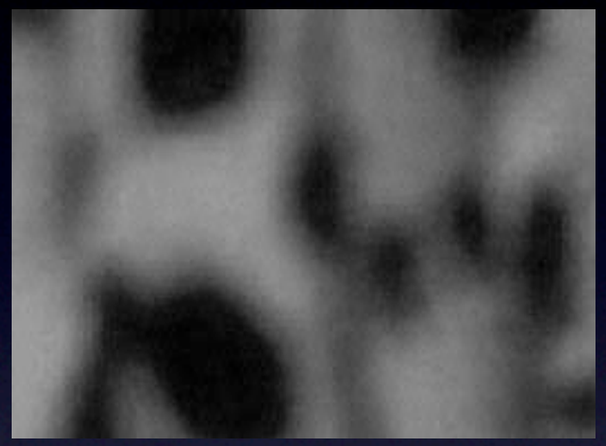
First construction of efficient Fountain codes was given by Luby (1998, published 2002).

How can we design them?

XOR

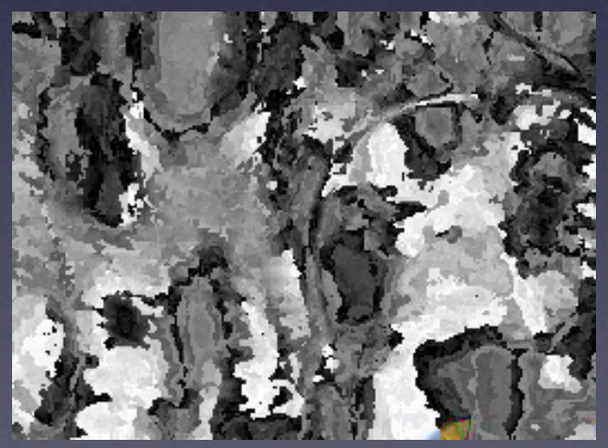
(aka addition in GF(2))

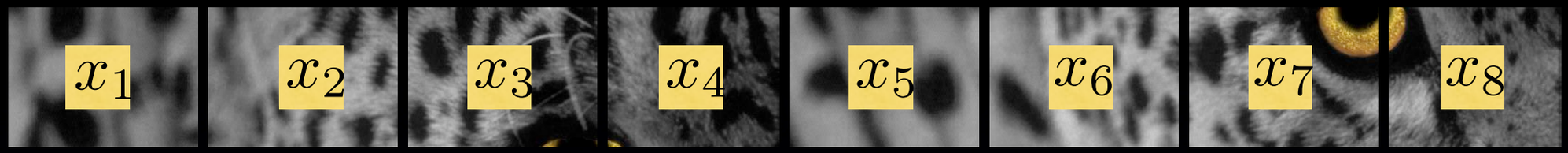
$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 0$$

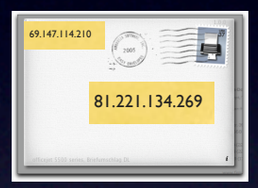

+

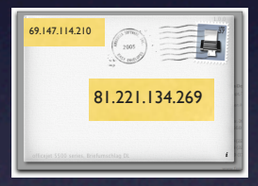
+

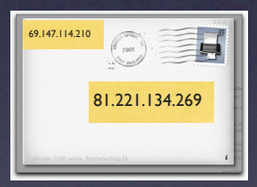

=

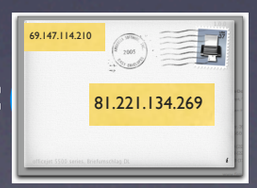




$x_1 + x_4 + x_7 =$ 

$x_4 + x_6 =$ 

$x_7 + x_3 + x_8 + x_2 =$ 

$x_5 + x_1 =$ 

$$
\begin{array}{cccccccccccccccc}
x_1 & & & + & x_3 & & & & & & & & & + & x_8 & = z_1 \\
 & x_2 & + & x_3 & & & + & x_5 & + & x_6 & + & x_7 & + & x_8 & = z_2 \\
x_1 & & & & & & & & & & + & x_7 & & & = z_3 \\
 & & & x_3 & + & x_4 & & & & & + & x_7 & + & x_8 & = z_4 \\
 & & & & & x_4 & & & & & + & x_6 & & & = z_5 \\
x_1 & & & + & x_4 & + & x_5 & & & & & & + & x_8 & = z_6 \\
 & & x_3 & + & x_4 & & & & + & x_6 & & & + & x_8 & = z_7 \\
x_1 & & & + & x_4 & + & x_5 & + & x_6 & & & & & & = z_8
\end{array}
$$

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \\ z_8 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \\ z_8 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix}$$

A *random* *k* x *k-matrix* with binary entries is invertible only with probability ~ 30%.

Probability tends to 1 only when the number of rows is around

$$k + log(k).$$

Example:

File size = 50MB, packet size = 1KB, Number of packets = 50000

If we receive *any set of* 50030 packets, the probability that recovery is not possible is about 0.0000000001.

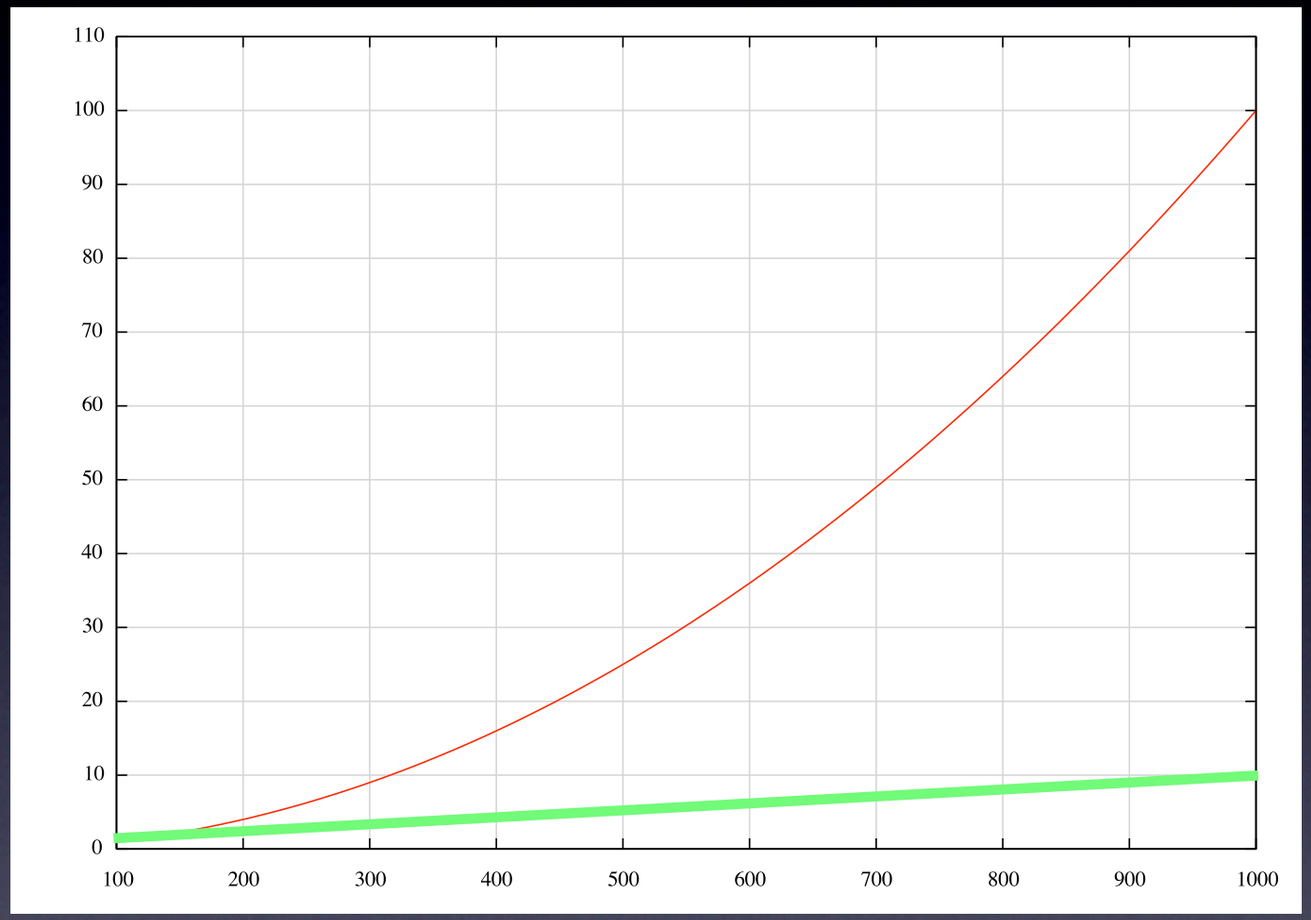
Complexity of creation/recovery is large with random selections!

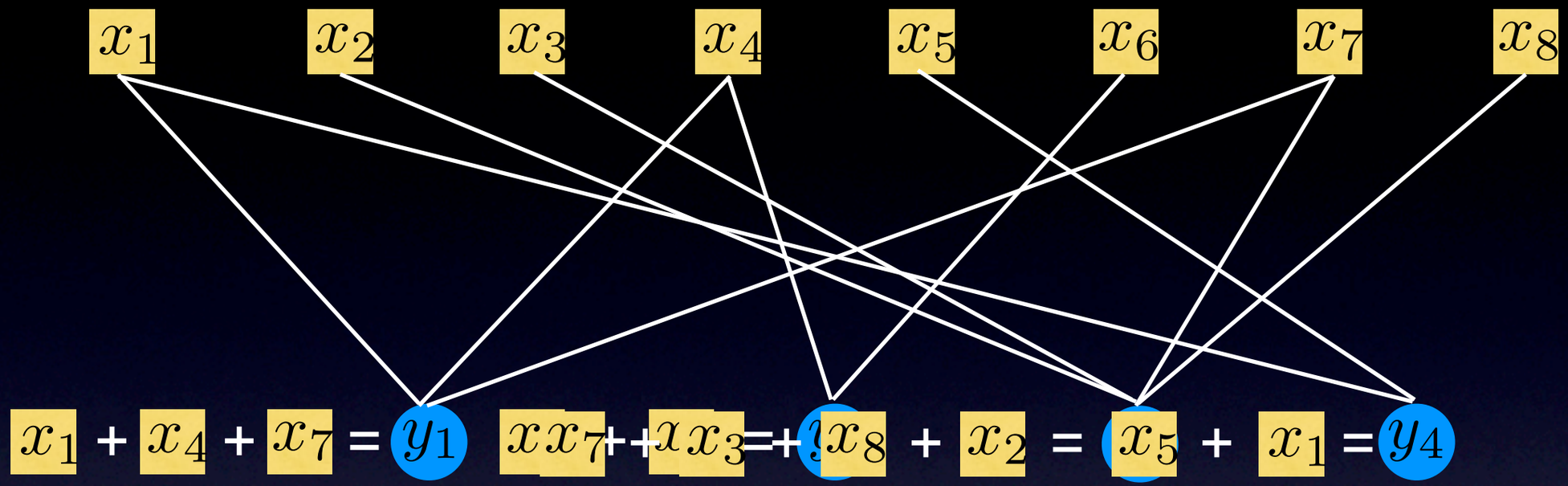
A

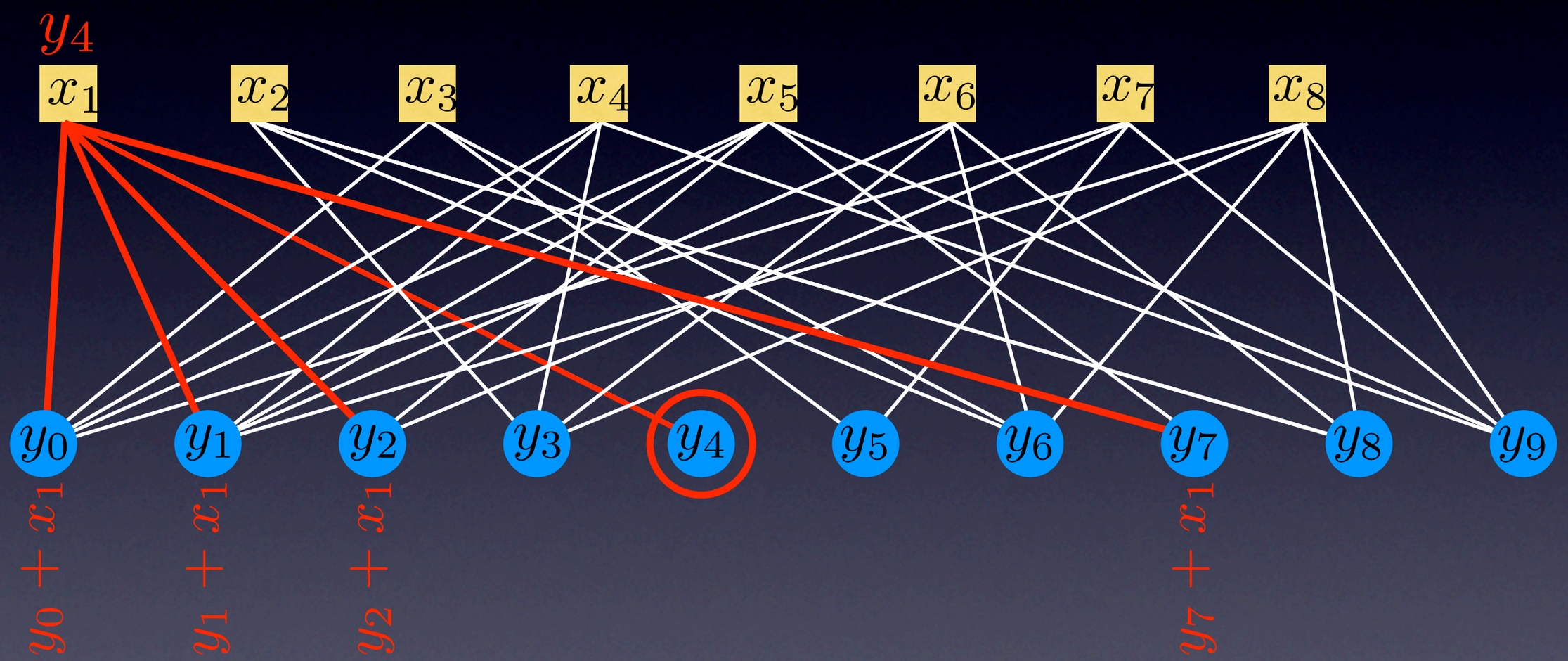
A^{-1}

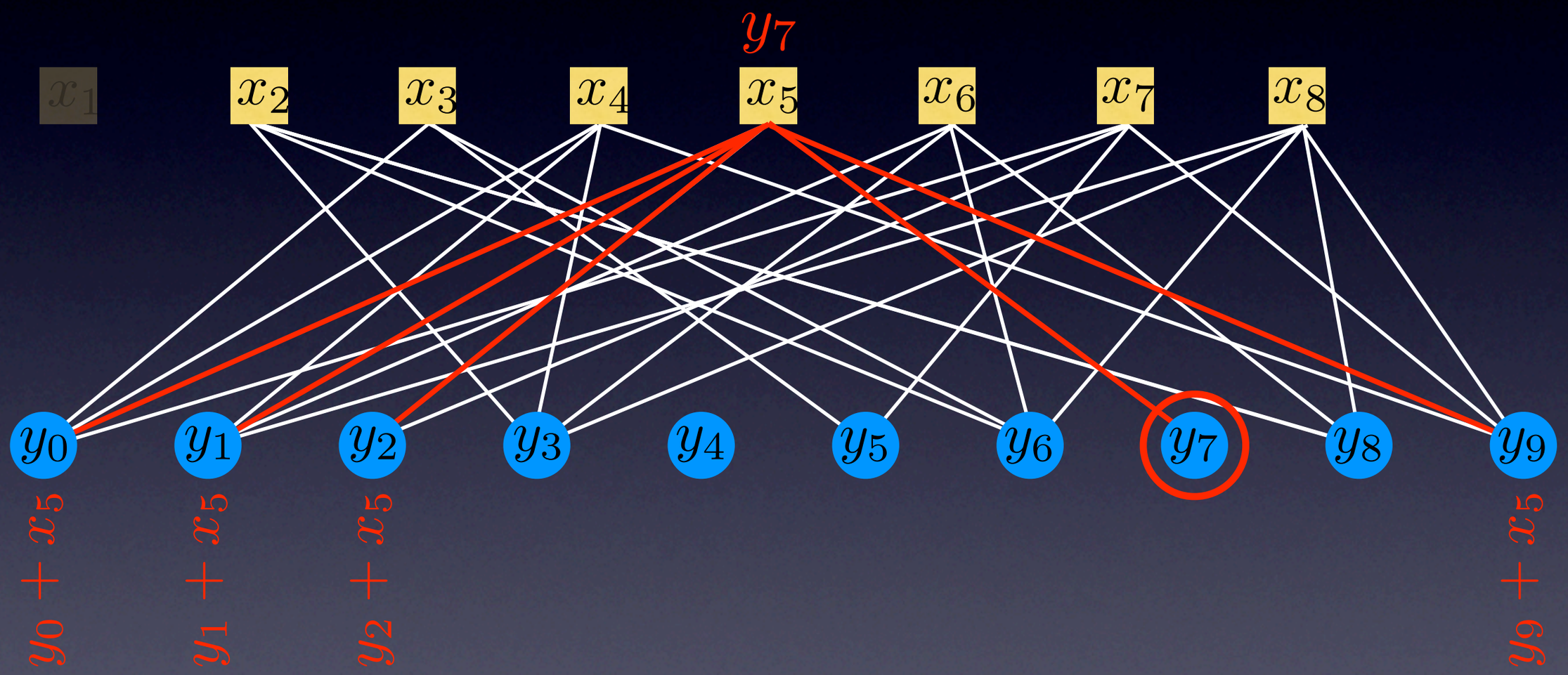
50000 packets, roughly 1250000000 operations.

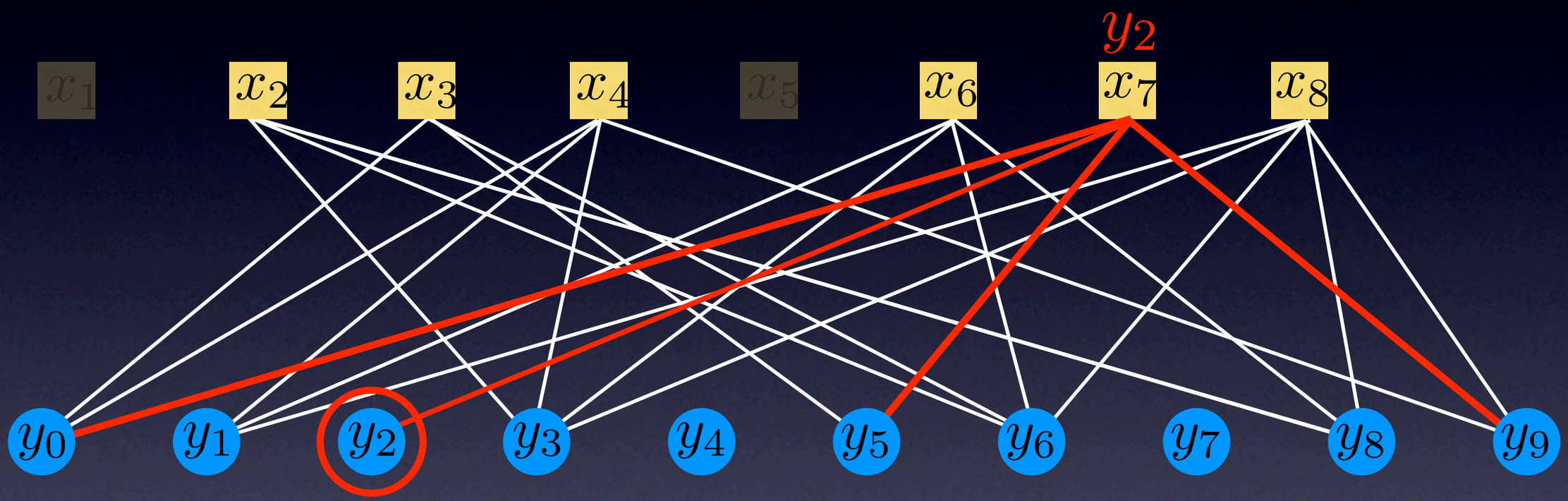
Would take 4.4 hours for recovery on a set-op box.

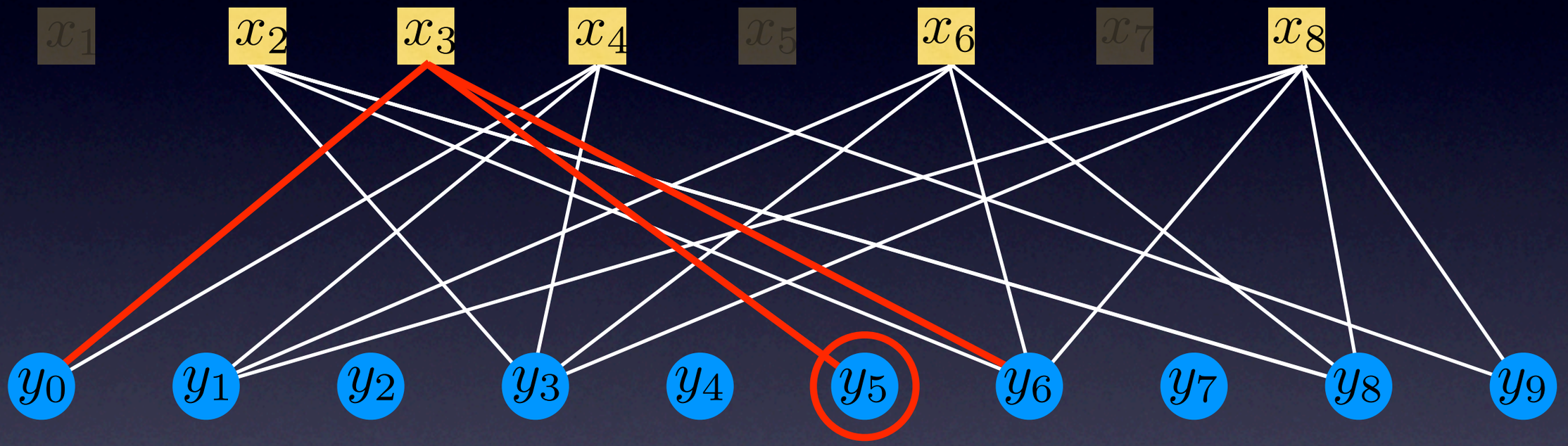

110
100
90
80
70
60
50
40
30
20
10
0
100
200
300
400
500
600
700
800
900
1000

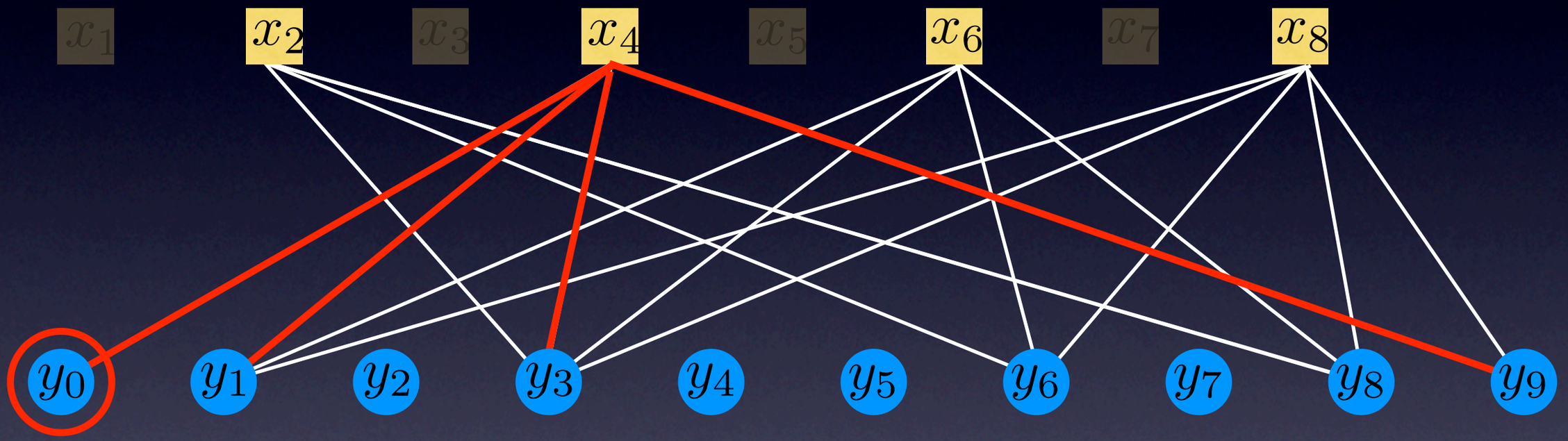

x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
$x_1 + x_4 + x_7 = y_1$
$x_7 + x_3 = y_8$
$x_8 + x_2 = x_5 + x_1 = y_4$

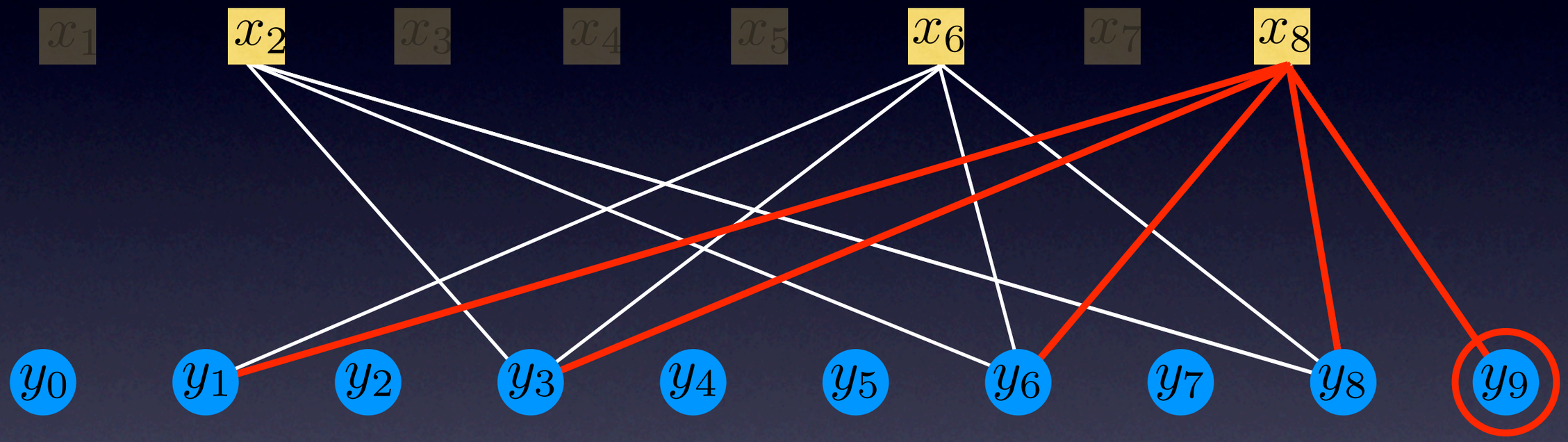


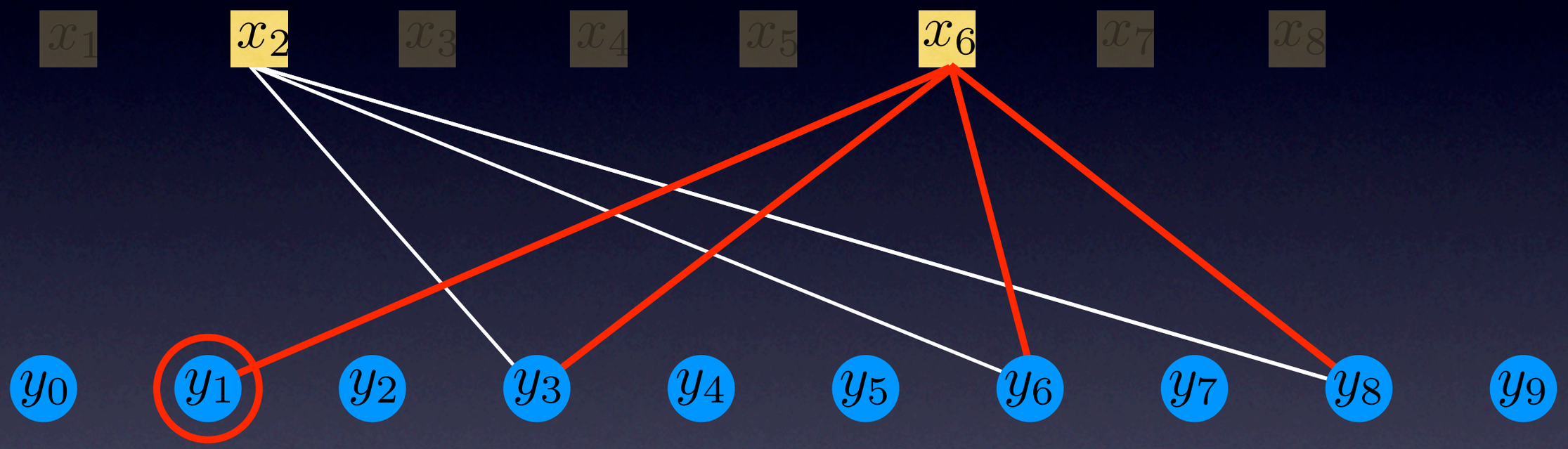












x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8

y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9

x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8

y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9

Successful recovery

How can we make sure that recovery is successful?

Generate the packets according to a probabilistic process, independently.

Any set of recovered packets has with high probability the same statistics.

Design the statistics such that recovery is possible, with high probability.

Resulting codes are called “Fountain Codes”

Fountain Codes

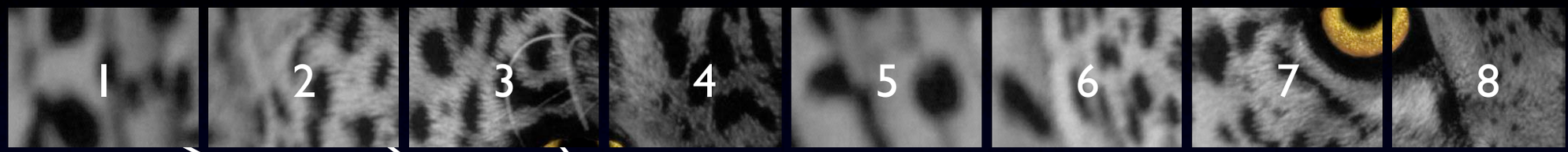
$\mathcal{D}$ Distribution on $\mathbb{F}_2^k$

Encoding of vector $(x_1, \ldots, x_k)$:

1. Sample from $\mathcal{D}$ a vector $(a_1, \ldots, a_k) \in \mathbb{F}_2^k$
2. Output $a_1 x_1 + \cdots + a_k x_k$
3. Goto 1

Example: $\mathcal{D}$ is the uniform distribution.

LT Codes



3

deg	probability
1	0.001
2	0.5
3	0.16
4	0.083
5	0.05
6	0.033
7	0.024
8	0.018
9	0.011
.....	.....

deg	probability
1	0.001
2	0.5
3	0.16
4	0.083
5	0.05
6	0.033
7	0.024
8	0.018
9	0.011
.....	.....

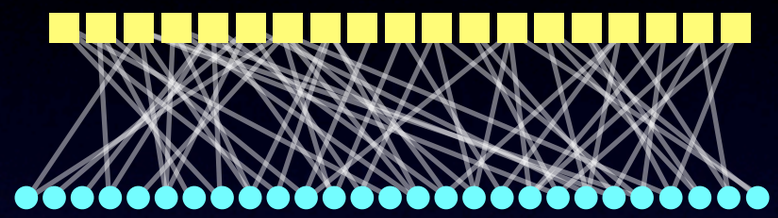
Degree Distribution

deg	probability
1	Ω_1
2	Ω_2
3	Ω_3
4	Ω_4
5	Ω_5
6	Ω_6
7	Ω_7
8	Ω_8
9	Ω_9
.....	.....

$$\Omega(x) = \Omega_1 x + \Omega_2 x^2 + \Omega_3 x^3 + \cdots$$

Notation

k input symbols

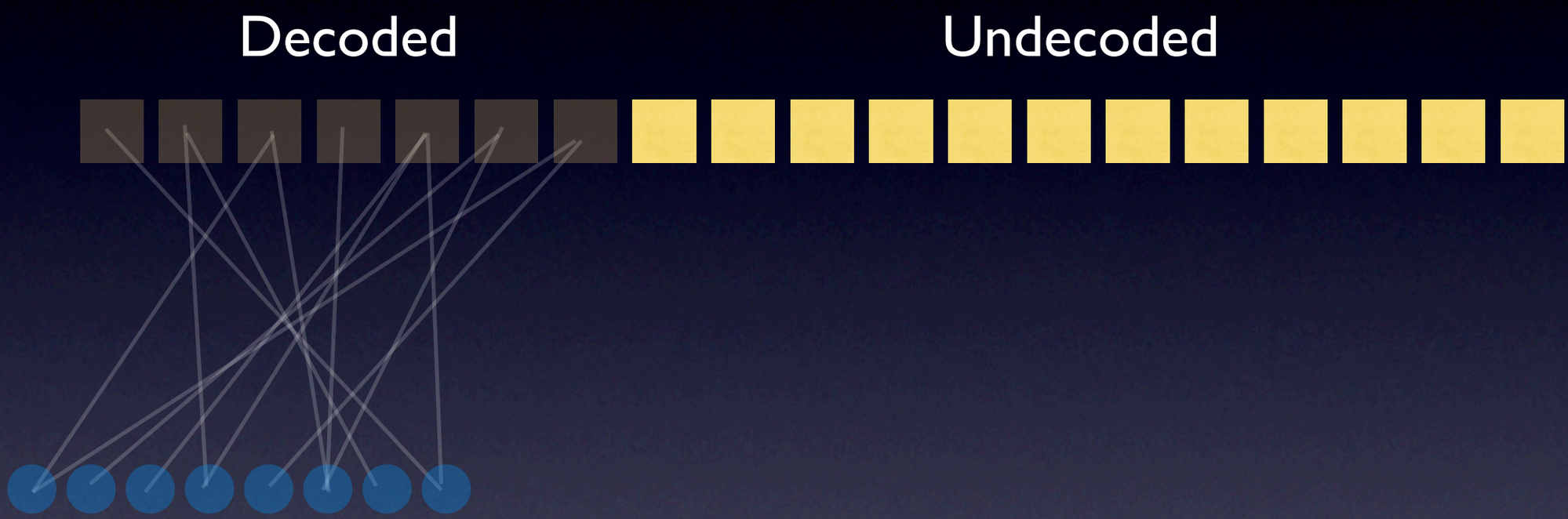


n output symbols

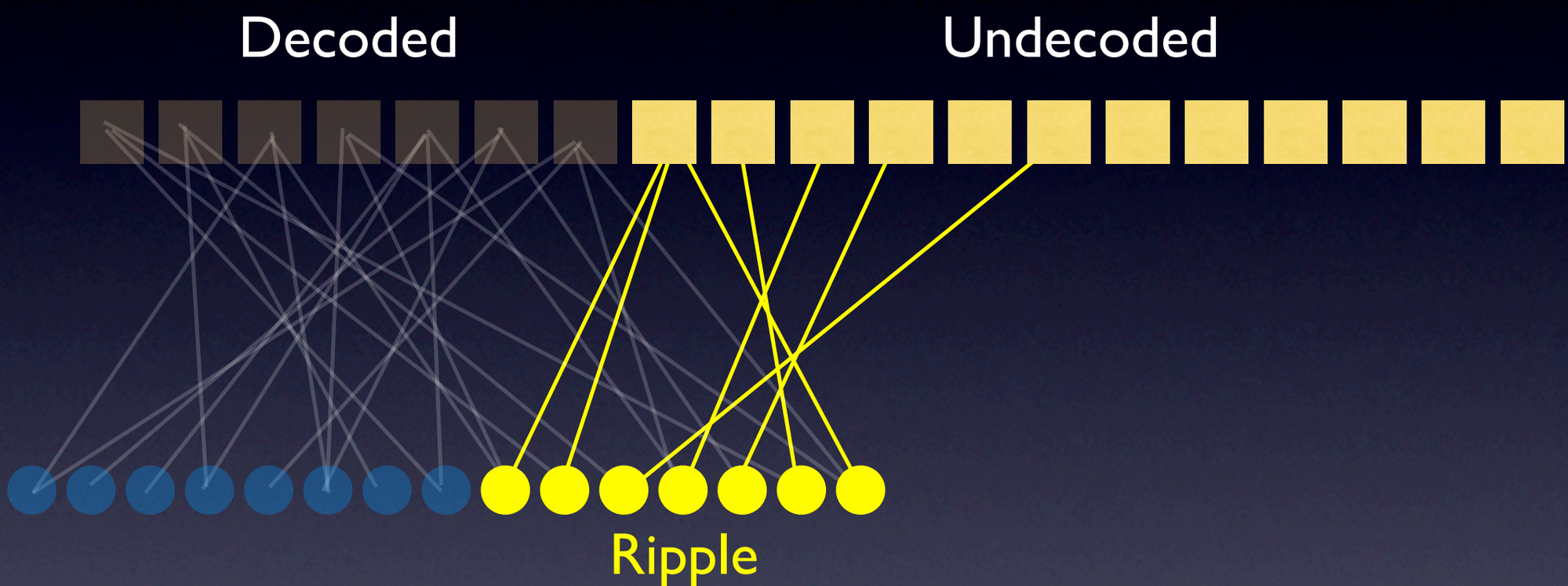
n/*k*-1 is called the *overhead*: *n* = (1+overhead)*k*

Release Probability

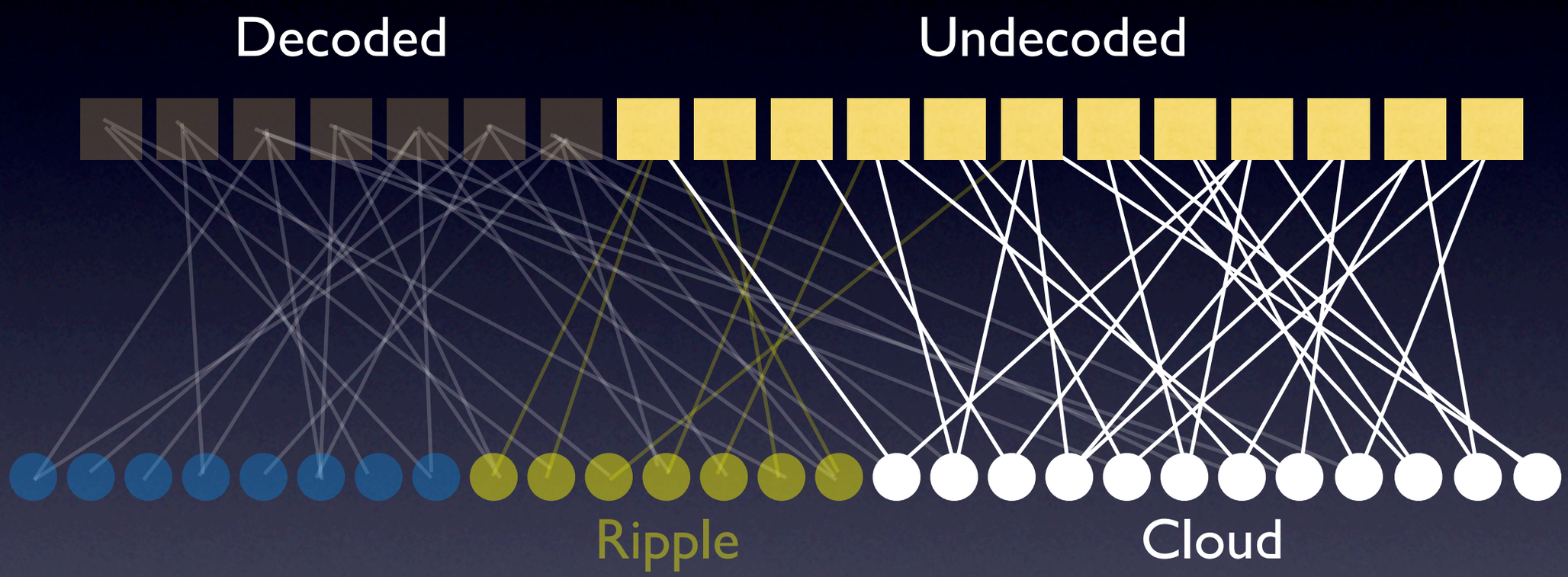
Release Probability



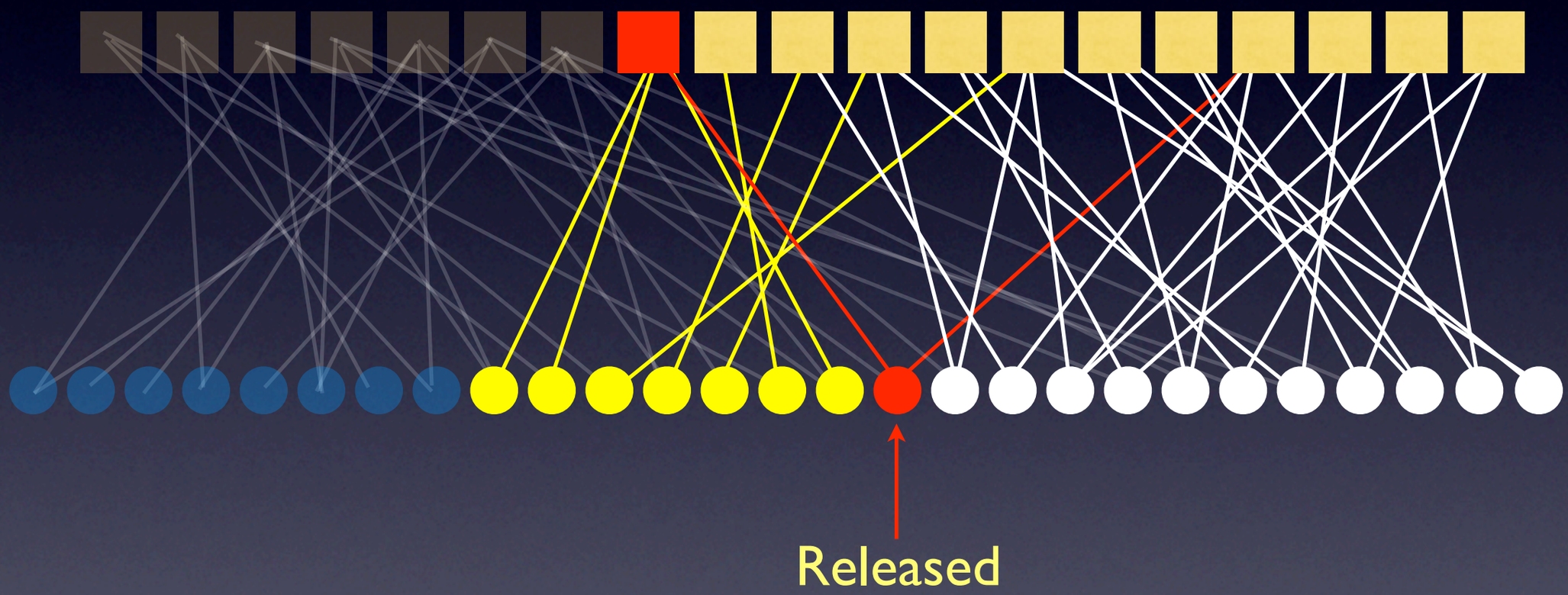
Release Probability



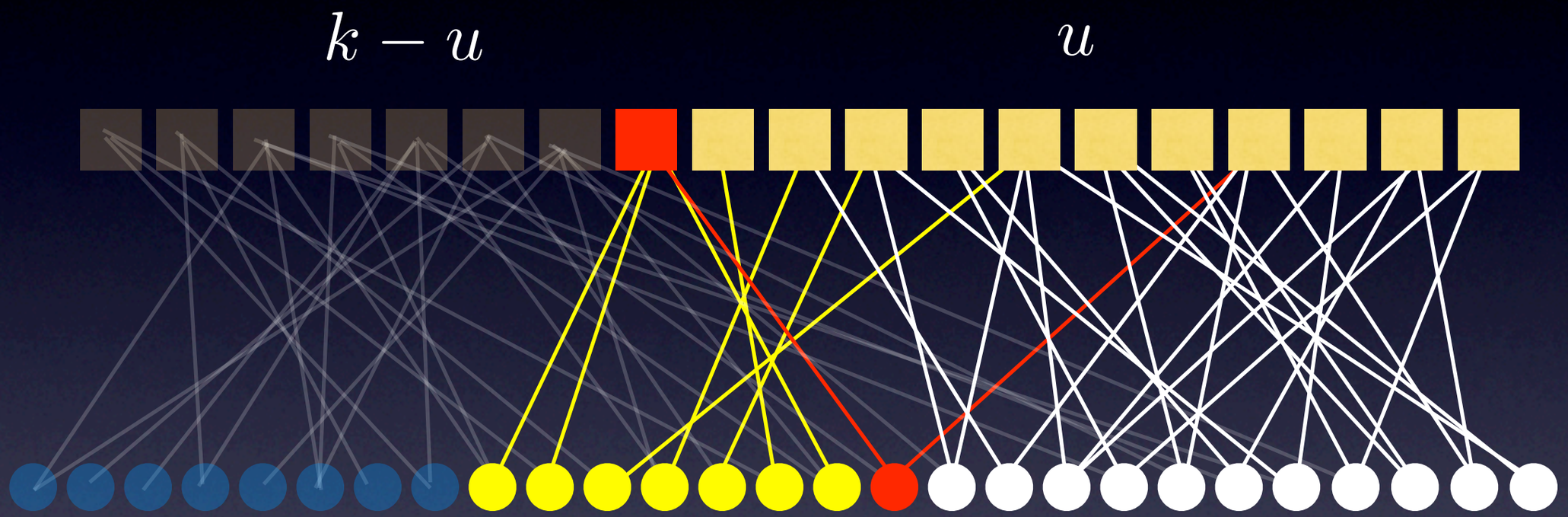
Release Probability



Release Probability

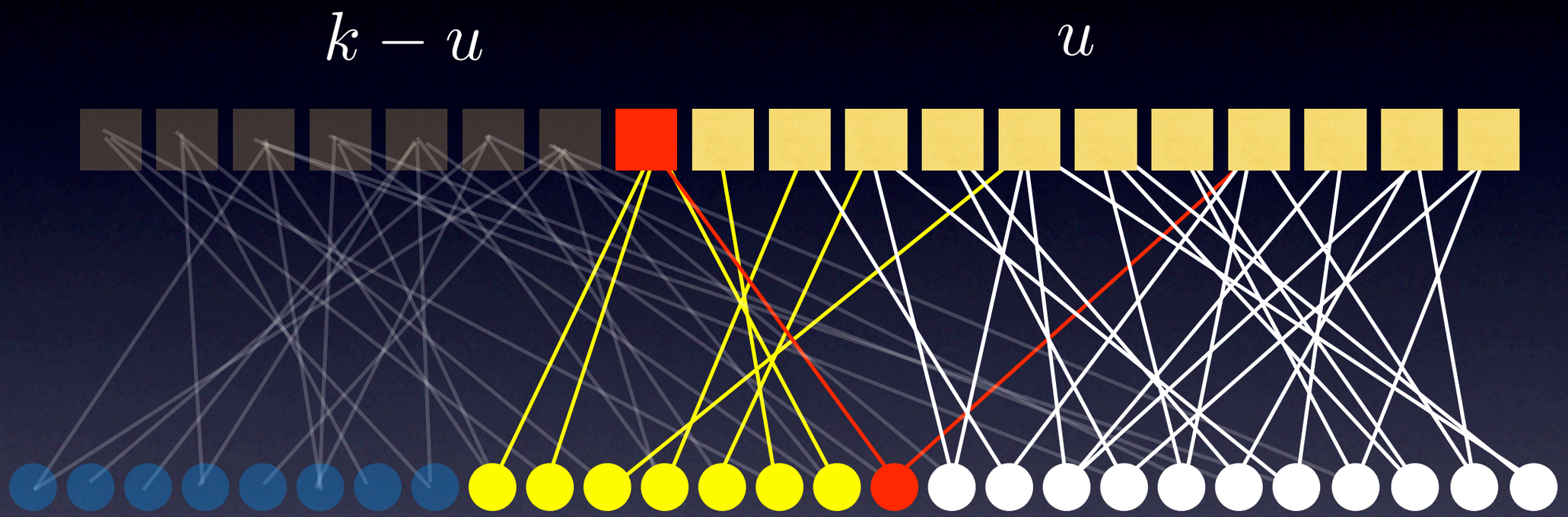


Release Probability



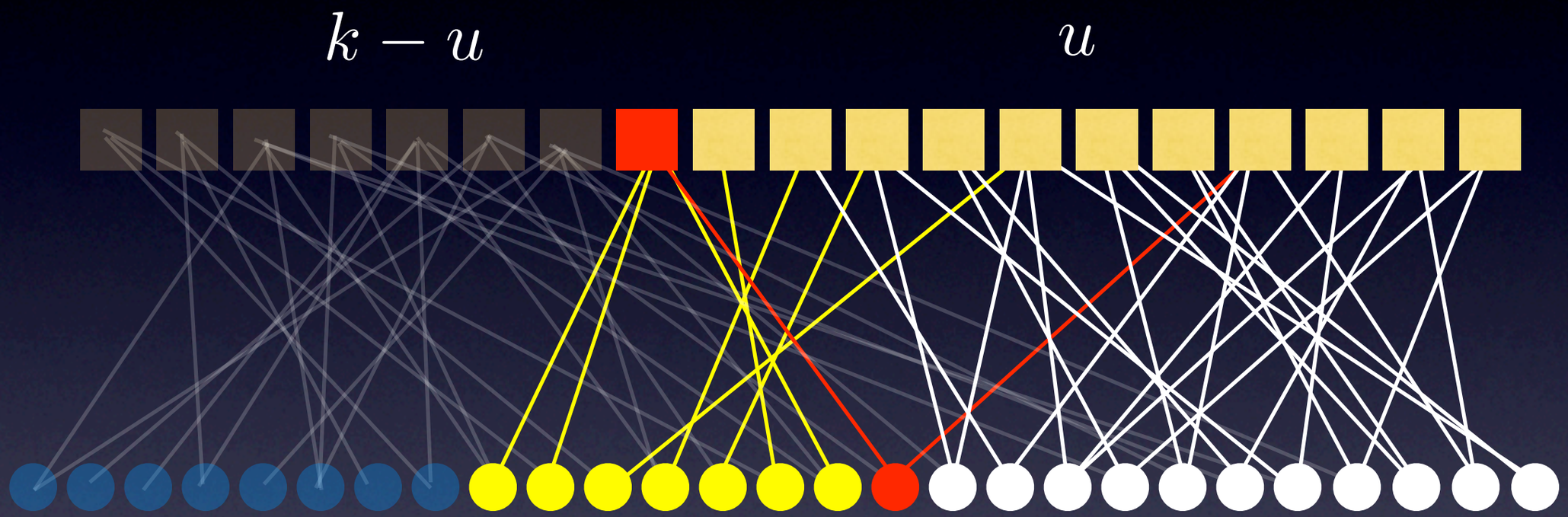
$$\Pr[\text{release}|\text{degree} = d] = d(d-1)\frac{1}{k}\frac{u-1}{k}\left(1-\frac{u}{k}\right)^{d-2}$$

Release Probability



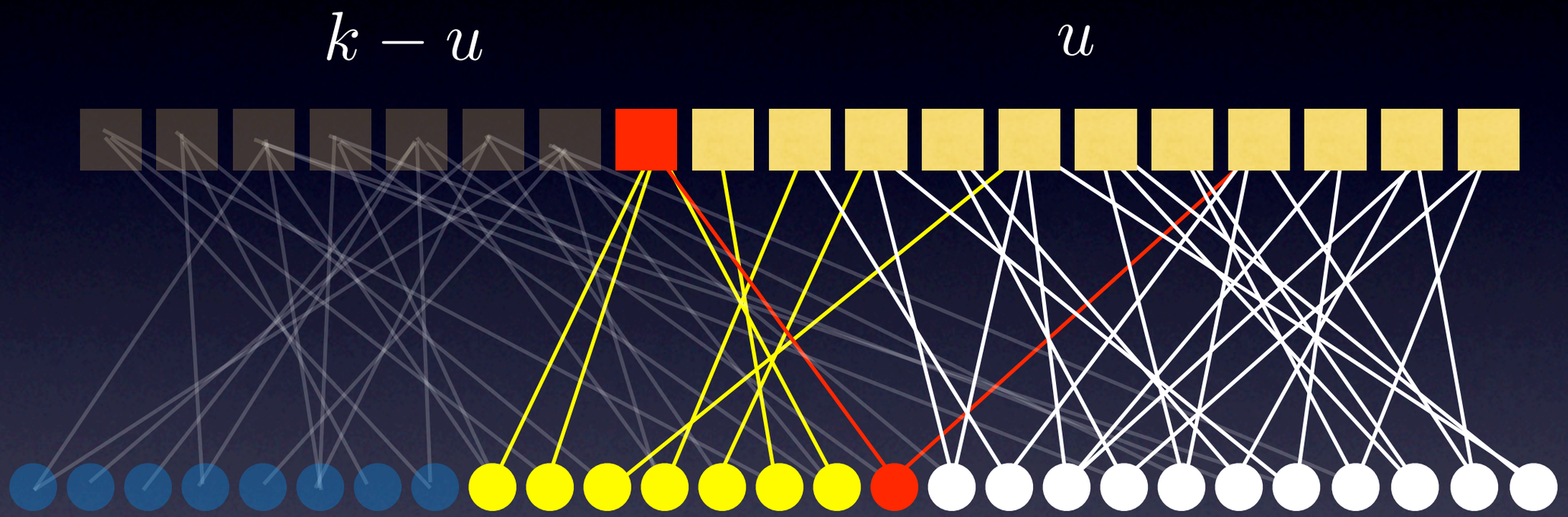
$$\Pr[\text{release}] = \sum_d \Omega_d d(d-1)\frac{1}{k}\frac{u-1}{k}\left(1-\frac{u}{k}\right)^{d-2}$$

Release Probability



$$\Pr[\text{release}] = \frac{1}{k}\frac{u-1}{k}\Omega''\left(1-\frac{u}{k}\right)$$

Release Probability



$$\mathrm{Exp}[\mathrm{release}] = \frac{n}{k}\frac{u-1}{k}\Omega''\left(1-\frac{u}{k}\right)$$

Asymptotic Degree Distribution

$$n \sim k$$

$$\frac{u-1}{k}\Omega''\left(1-\frac{u}{k}\right) = 1$$

$$x\Omega''\left(1-x\right) = 1$$

$$\Omega(x) = \sum_{i\geq 2} \frac{1}{i(i-1)} x^i$$

Soliton Distribution

$$\Omega(x) = \frac{x}{k} + \sum_{d=2}^{k} \frac{x^d}{d(d-1)}$$

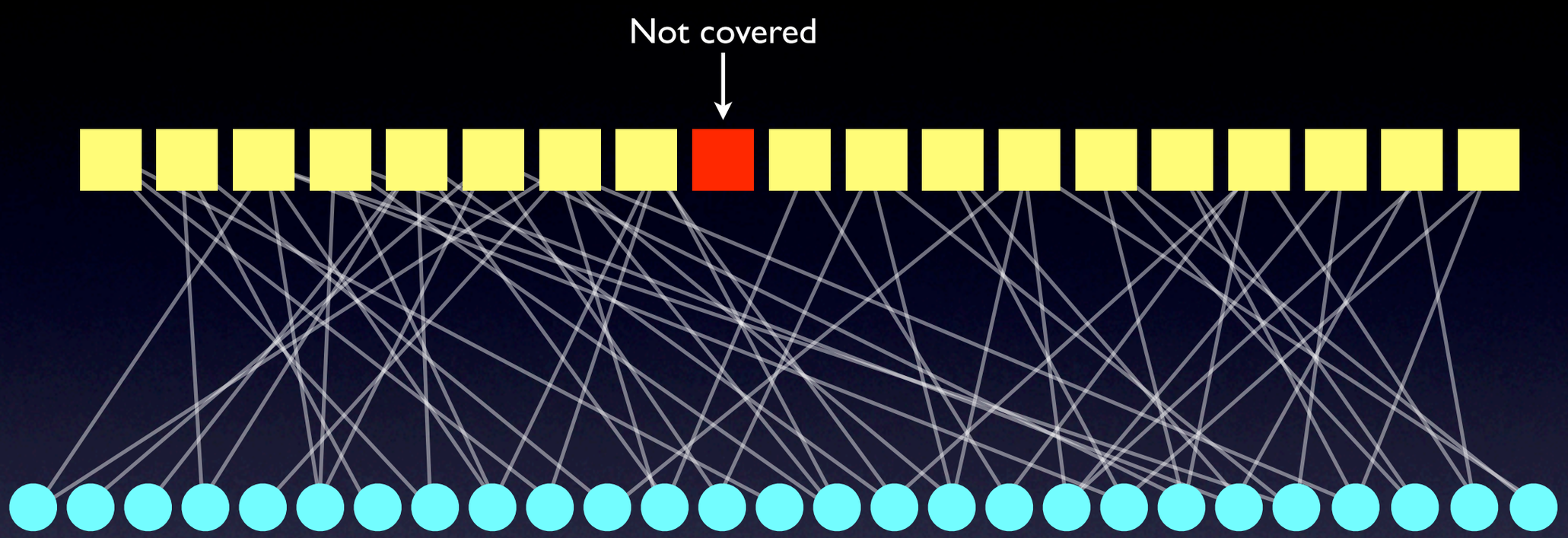
Makes sure that the *average* number of released output symbols is 1

Average Degree

$$\begin{aligned}
\Omega'(1) &= \sum_{d=1}^{k} \Omega_d d \\
&= \frac{1}{k} + 1 + \frac{1}{2} + \cdots + \frac{1}{k-1} \\
&\sim \ln(k)
\end{aligned}$$

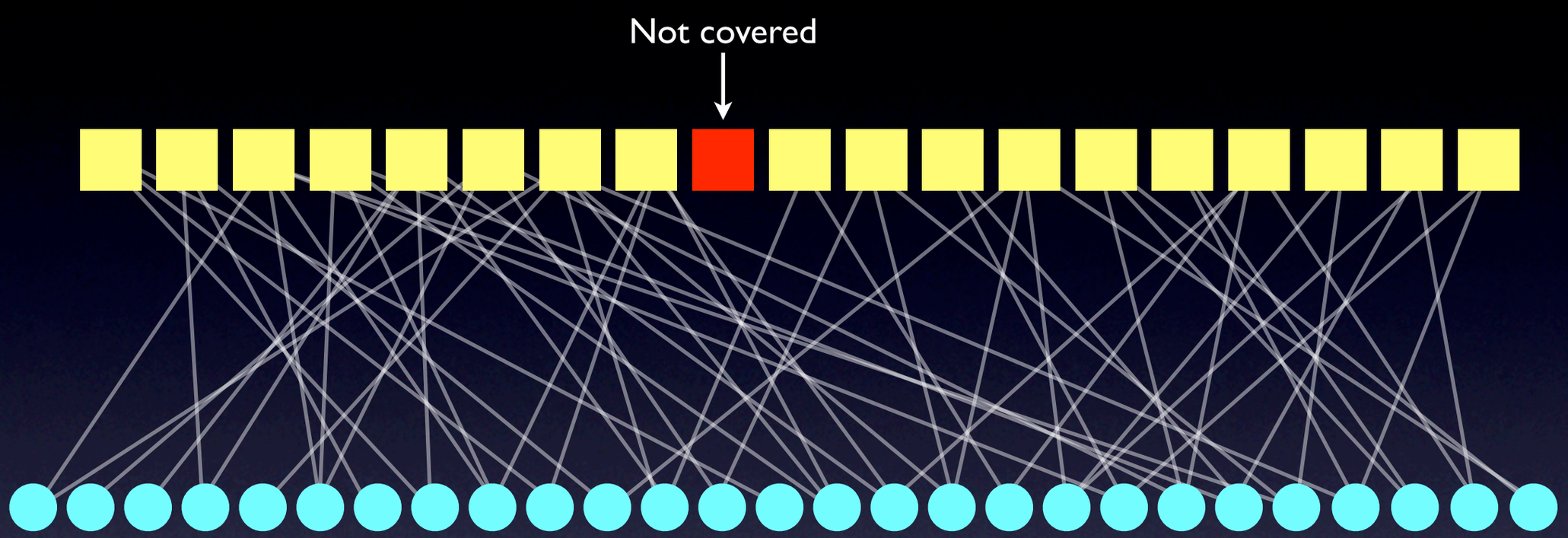
Encoding and decoding run in time O(k log(k)).
Is this necessary?

Average Degree



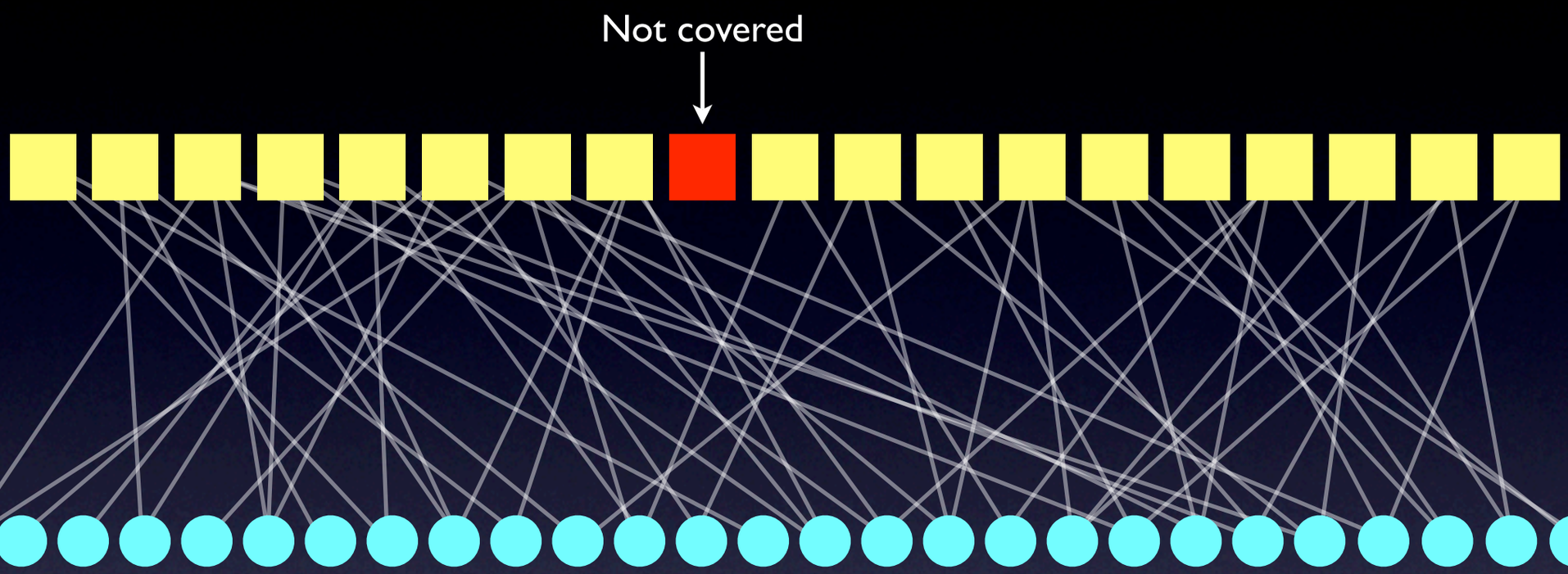
Pr[● does not cover ■ | deg(●) = d] = $1 - \frac{d}{k}$

Average Degree



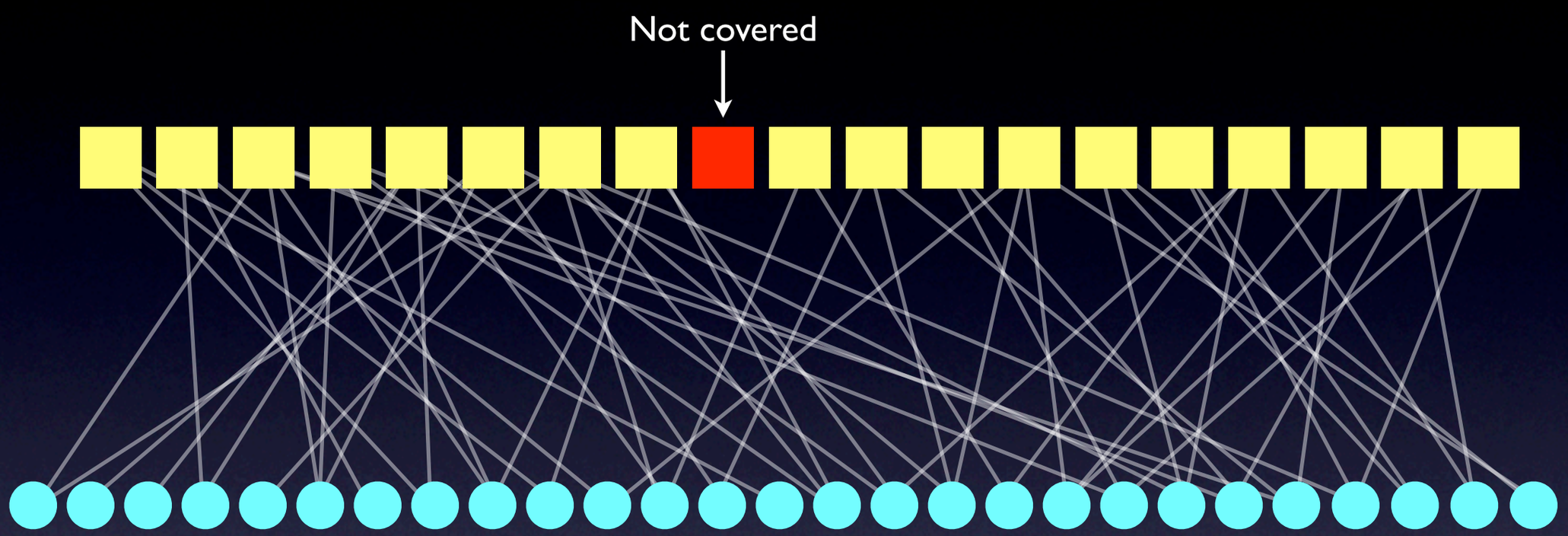
Pr[● does not cover ■] = $\sum_d \Omega_d \left(1 - \frac{d}{k}\right) = 1 - \frac{a}{k}$

Average Degree



Pr[●●●●●●●●●● does not cover ■] = $\left(1 - \frac{a}{k}\right)^n \sim \mathrm{e}^{-an/k} \sim \mathrm{e}^{-a}$

Average Degree



$$\mathrm{e}^{-a} < \frac{1}{k} \implies a > \ln(k)$$

Conclusion

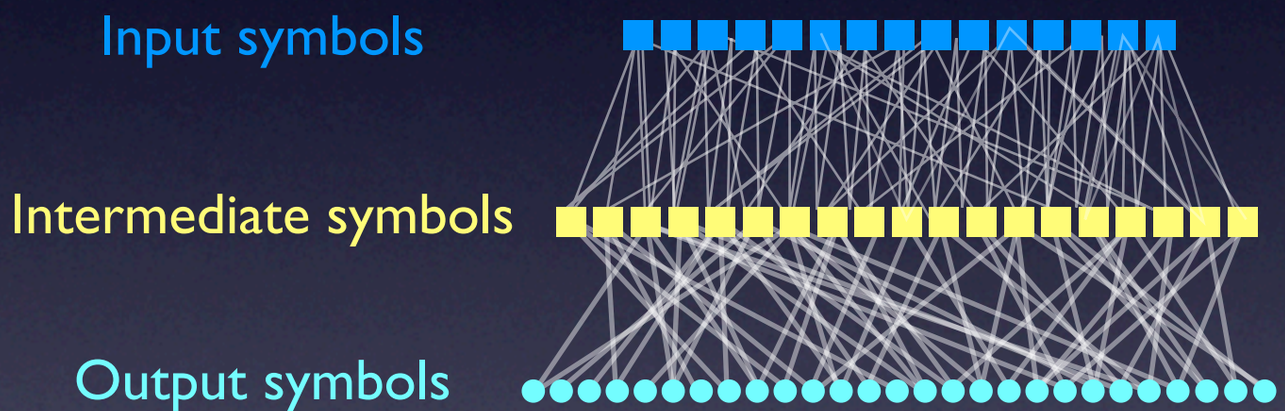
If the overhead is small, then the average degree has to be logarithmic in k.

Encoding and decoding cannot run in linear time.

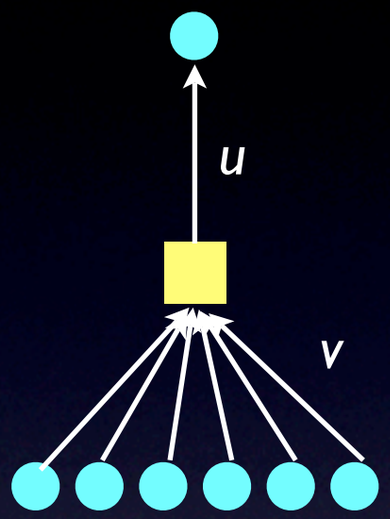
Modification?

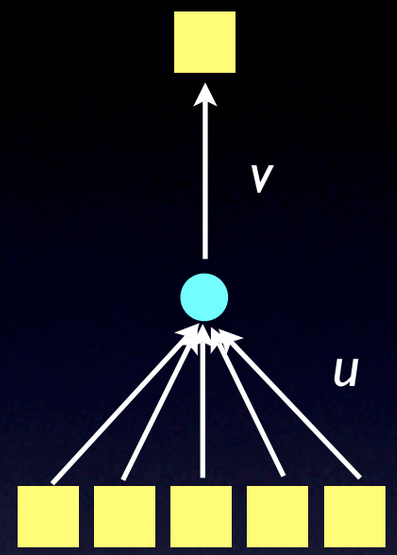
Raptor Codes

Idea: Precode the input symbols. Allows using "light" LT-code.



Tracking Probabilities





Pr[u=0 | deg(■) = d] =

(Pr[v=0])$^{d-1}$

Pr[v=0 | deg(●) = d] =

1-(1-Pr[u=0])$^{d-1}$

$$1 - x - \mathrm{e}^{-(1+\varepsilon)\Omega'(x)} > 0, \qquad 1 + \varepsilon = \frac{n}{k}$$

Designing Degree Distributions

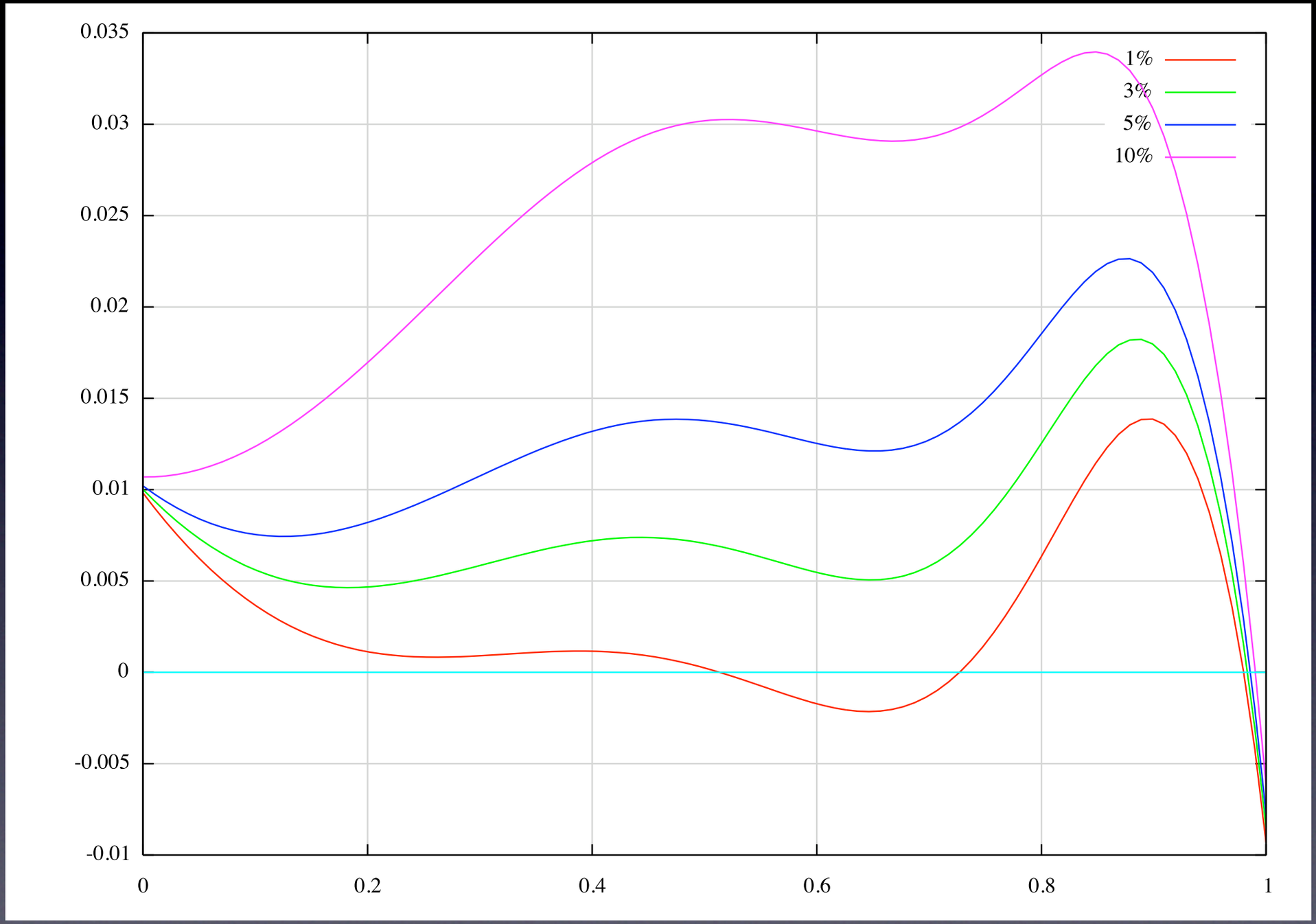
$$1 - x - \mathrm{e}^{-(1+\varepsilon)\Omega'(x)} > c\sqrt{\frac{1-x}{k}}$$

Motivated by assuming that the variations of the ripple follow a random walk.

Standardized Raptor

$$\Omega(x) = 0.00977x + 0.45904x^2 + 0.21096x^3 + 0.1134x^4 + 0.11134x^{10} + 0.0799x^{11} + 0.01559x^{40}.$$

Standardized Raptor



What Else Can be Done?

- Error probability of decoder can be calculated for finite lengths
- Variance of the size of the ripple can be tracked for finite length
- Raptor codes can be designed for which the ML decoder has low complexity on the erasure channel
- Relationship between the decoder and the size of the giant component in random graphs
- Design and analysis on other memoryless symmetric channels
- The codes are being used in practice.

Main Open Problem

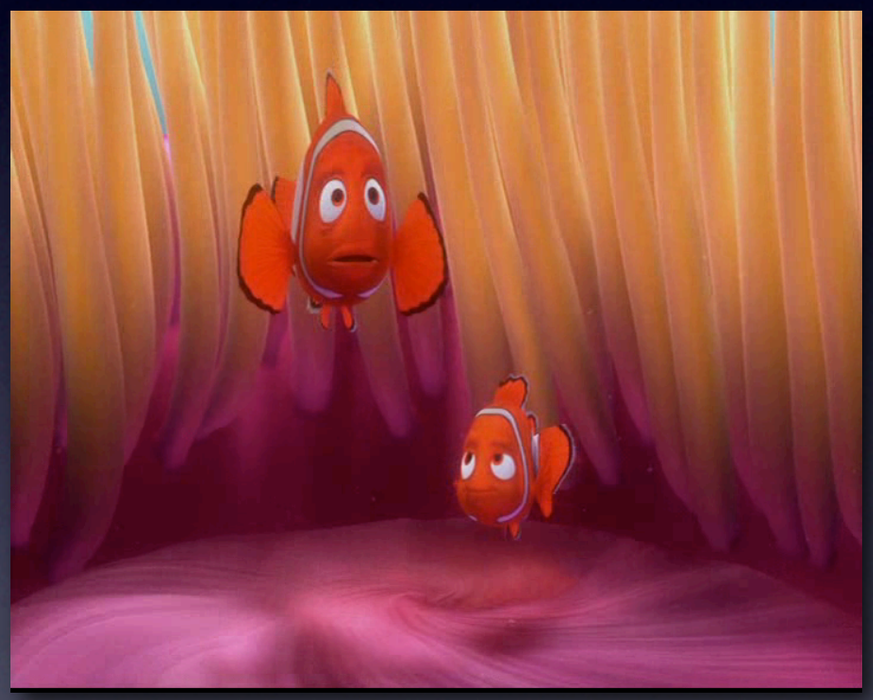
What is the equivalent of the Soliton distribution for general binary input memoryless symmetric channels?

Conjecture (Luby-S):

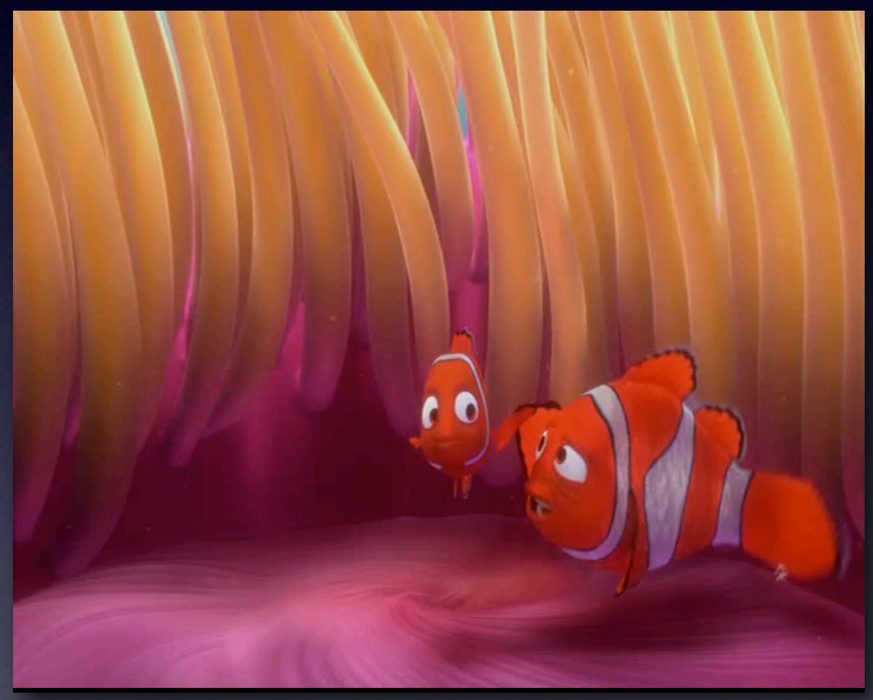
$g(x)$ pdf of the LLR of the channel

$$\Pi(\mathcal{C}) = \int_{-\infty}^{\infty} \tanh\left(\frac{x}{2}\right) g(x)\mathrm{d}x$$

$$\Omega_{\mathcal{C}}(x) = \frac{1-(1-x)^{\Pi(\mathcal{C})}-\Pi(\mathcal{C})}{1-\Pi(\mathcal{C})}$$



With Raptor



Without Raptor

Thank You!