REVIEW

January 20, 2025

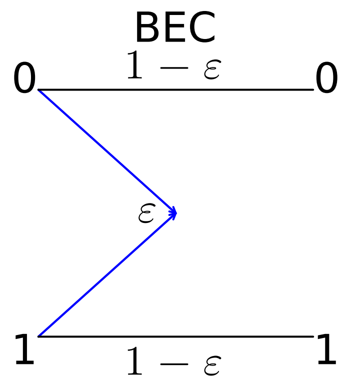
Quick Summary of P2P Channels

[1]:
```
using Plots, LaTeXStrings
```

[2]:
```
# Define coordinates for transmitted and received states
tx = [0, 1]
rx = [0, 1]

# Define probabilities as labels
p = [L"\epsilon", L"1 - \epsilon"];
```

[3]:
```
# Plot the BSC diagram
plot(grid=false
    , xaxis=false, yaxis=false
    , framestyle=:none, size = (200,200)
    , title = "BEC"
)

plot!([0, 0.5], [1, 0.5], arrow=:arrow, label="", color=:blue) # p line
plot!([0, 0.5], [0, 0.5], arrow=:arrow, label="", color=:blue) # p line
plot!([0, 1], [1, 1], label="", color=:black) # 1-p
plot!([0, 1], [0, 0], label="", color=:black)

# Annotate the graph
annotate!(-0.05, 1.05, tx[1]); annotate!(1.05, 1.05, rx[1])
annotate!(-0.05, -0.05, tx[2]); annotate!(1.05, -0.05, rx[2])
annotate!(0.4, 0.5, p[1]); annotate!(0.5, -0.1, p[2]); annotate!(0.5, 1.1, p[2])
```

[3]:



1 Binary Erasure Channel (BEC)

1. **Channel Model**:
   - Transmits binary symbols (0 or 1).
   - Each transmitted bit is either:
     - **Received correctly** with probability $1-\epsilon$, or
     - **Erased** with probability ϵ, represented as an erasure symbol (e).

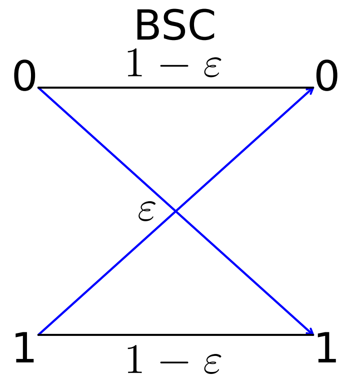
   Example:
   - $0 \to 0$ or e,
   - $1 \to 1$ or e.
2. **Capacity** (C): $\boxed{C = 1-\epsilon}$
   - 1: Maximum capacity with no erasures ($\epsilon = 0$).
   - ϵ: Fraction of bits erased by the channel, reducing capacity.
3. **Behavior**:
   - $\epsilon = 0$: Perfect channel, $C = 1$.
   - $\epsilon = 1$: Completely erasing channel, $C = 0$.
   - For $0 < \epsilon < 1$: Capacity decreases linearly as ϵ increases.

Compact Intuition: The **Binary Erasure Channel** (BEC) capacity is the fraction of bits successfully transmitted. Erasures (ϵ) reduce capacity by removing information from the channel.

[4]:
```
# Plot the BSC diagram
plot(grid=false
    , xaxis=false, yaxis=false
    , framestyle=:none, size = (200,200)
    , title = "BSC"
)

plot!([0, 1], [1, 0], arrow=:arrow, label="", color=:blue) # p line
plot!([0, 1], [0, 1], arrow=:arrow, label="", color=:blue) # p line
plot!([0, 1], [1, 1], label="", color=:black) # 1-p
```

```
plot!([0, 1], [0, 0], label="", color=:black)

# Annotate the graph
annotate!(-0.05, 1.05, tx[1]); annotate!(1.05, 1.05, rx[1])
annotate!(-0.05, -0.05, tx[2]); annotate!(1.05, -0.05, rx[2])
annotate!(0.4, 0.5, p[1]); annotate!(0.5, -0.1, p[2]); annotate!(0.5, 1.1, p[2])
```

[4]:



2 Binary Symmetric Channel (BSC)

1. **Channel Model**:
   - Transmits binary symbols (0 or 1).
   - Each bit has a probability ϵ of being flipped.
   - Error probability: $P(0 \to 1) = P(1 \to 0) = \epsilon$.
   - Correct transmission probability: $P(0 \to 0) = P(1 \to 1) = 1 - \epsilon$.
2. **Capacity** (C_{BSC}): $\boxed{C_{BSC} = 1 - H_2(\epsilon)}$
   - 1: Maximum capacity without errors.
   - $H_2(\epsilon)$: Binary entropy function, representing uncertainty due to errors.
3. **Binary Entropy Function** ($H_2(\epsilon)$): $\boxed{H_2(\epsilon) = -\epsilon \cdot \log_2(\epsilon) - (1-\epsilon) \cdot \log_2(1-\epsilon)}$
   - $H_2(0) = 0$: No errors, full capacity.
   - $H_2(0.5) = 1$: Maximum uncertainty, no capacity.
4. **Behavior**:
   - $\epsilon = 0$: Perfect channel, $C_{BSC} = 1$.
   - $\epsilon = 0.5$: Completely noisy, $C_{BSC} = 0$.
   - For $0 < \epsilon < 0.5$: Capacity decreases as ϵ increases.

Compact Intuition: The BSC capacity is the theoretical maximum rate of reliable data transmission, reduced by the uncertainty caused by errors. Lower ϵ means higher capacity, while higher ϵ reduces it.

2.0.1 AWGN Channel Summary

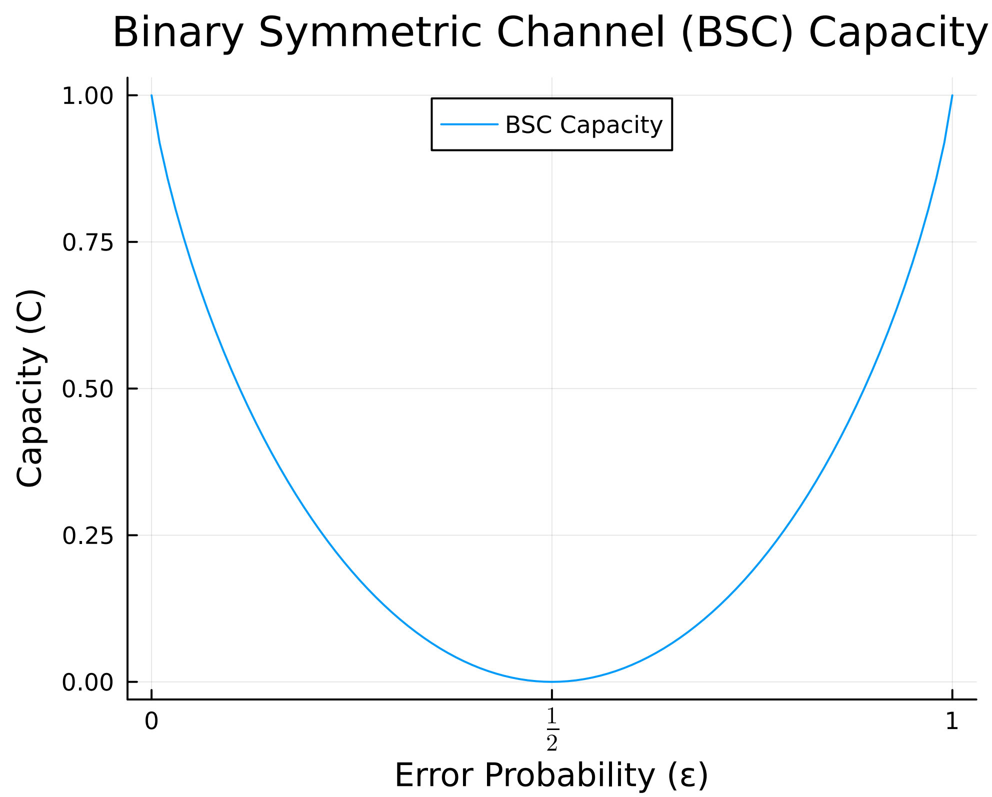
1. **Channel Model**: $\boxed{y = x + w, \quad w \sim N(0, N_0)}$
   - x: Transmitted signal.
   - y: Received signal.
   - w: Gaussian noise with zero mean and variance N_0 the noise power spectral density.
2. **Signal Power**: $\boxed{P = E[|x|^2]}$
3. **Signal-to-Noise Ratio (SNR)**: $\boxed{SNR = \frac{P}{N_0}}$
4. **Channel Capacity**: $\boxed{C = \log_2(1 + SNR) = \log_2\left(1 + \frac{P}{N_0}\right)}$
   - C: Maximum achievable data rate (in bps/Hz).
5. **Key Behavior**:
   - $P \uparrow$ (high signal power): $C \uparrow$ (more capacity).
   - $N_0 \uparrow$ (high noise): $C \downarrow$ (less capacity).

Insight: The AWGN channel capacity quantifies the theoretical limit of reliable communication over a noisy channel.

[5]:
```
using Plots

# Define the binary entropy function
function H( )
    if  == 0 ||  == 1
        return 0.0
    end
    return - * log2( ) - (1 - ) * log2(1 - )
end

# Define the BSC capacity function
function C ( )
    return 1 - H( )
end

# Generate values of  from 0 to 1
_values = 0:0.01:1
capacities = C .( _values)

# Plot the capacity curve
plot( _values, capacities,
     label = "BSC Capacity"
    , xlabel = "Error Probability ( )", ylabel = "Capacity (C)"
    , title = "Binary Symmetric Channel (BSC) Capacity"
```

```
    , legend = :top, size = (500,400)
    , xticks = (0:0.5:1, ["0", L"\frac{1}{2}", "1"])
)
```

[5]:



2.1 Communication System Components:

$

$$\boxed{\text{Source}} \rightarrow \underline{x} = (x_1, \dots, x_n) \rightarrow \boxed{\text{Encoder}} \rightarrow \underline{c} = (c_1, \dots, c_n) \quad \rceil \tag{1}$$

$$\boxed{\text{Channel}} \tag{2}$$

$$\boxed{\text{Sink}} \leftarrow \hat{\underline{x}} = (\hat{x}_1, \dots, \hat{x}_n) \leftarrow \boxed{\text{Decoder}} \leftarrow \underline{y} = (y_1, \dots, y_n) \quad \rfloor \tag{3}$$

$

1. **Source**: Produces the information $\underline{x} = (x_1, \dots, x_n)$.
2. **Encoder**: Transforms $\underline{x}$ into a codeword $\underline{c} = (c_1, \dots, c_n)$, adding redundancy.
3. **Channel**: Transmits $\underline{c}$, introducing errors, resulting in $\underline{y} = (y_1, \dots, y_n)$.
4. **Decoder**: Processes $\underline{y}$ to estimate $\hat{\underline{x}} = (\hat{x}_1, \dots, \hat{x}_n)$, correcting errors.
5. **Sink**: Receives $\hat{\underline{x}}$, ideally matching $\underline{x}$.

2.1.1 Objective:

Ensure $\underline{\hat{x}} = \underline{x}$ despite channel errors.

2.1.2 Properties of Linear Block Code

1. **Linearity**:
    - The set of codewords $\mathcal{X}$ forms a **linear subspace** of $\mathbb{F}_2^n$.
    - Any linear combination of codewords is also a valid codeword: $\underline{v}_1 + \underline{v}_2 \in \mathcal{X},\ \forall \underline{v}_1, \underline{v}_2 \in \mathcal{X}$.
2. **Generator Matrix (G)**:
    - The $k \times n$ generator matrix G maps k-bit input vectors ($\underline{u} \in \mathbb{F}_2^k$) to n-bit codewords ($\underline{v} = \underline{u}^\top G$).
    - Defines the structure of the codebook $\mathcal{X}$.
3. **Code Rate (R)**:
    - The ratio of information bits to total bits: $R = \frac{k}{n}$
    - Indicates the efficiency of the code.
4. **Code Size ($|\mathcal{X}|$)**:
    - The number of unique codewords: $|\mathcal{X}| = 2^k$
5. **Minimum Hamming Distance ($d_{\mathbf{min}}$)**:
    - The smallest Hamming distance between any two distinct codewords.
    - Determines the error-detecting and error-correcting capability: $t = \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$
        - t: Maximum correctable errors.
        - $d_{\min} - 1$: Maximum detectable errors.
6. **Parity Check Matrix (H)**:
    - The H matrix defines the null space of G: $HG^\top = 0$
    - Used to verify codewords: $H\underline{v}^\top = 0 \implies \underline{v} \in \mathcal{X}$.
7. **Error Detection and Correction**:
    - Error detection: Capable of detecting up to $d_{\min} - 1$ errors.
    - Error correction: Can correct up to $\lfloor \frac{d_{\min}-1}{2} \rfloor$ errors.
8. **Redundancy**:
    - The number of redundant bits added for error correction is $n-k$, where n is the codeword length.

2.1.3 Compact Summary:

- **Linear subspace**: Codewords form a subspace of $\mathbb{F}_2^n$.
- **Size**: $|\mathcal{X}| = 2^k$.
- **Rate**: $R = \frac{k}{n}$.
- **Error capabilities**: Based on $d_{\min}$.
- Defined by:
    - **Generator matrix (G)**.
    - **Parity check matrix (H)**.

Error Correction Error Correction **Code** $\mathcal{X}$
Linear error corr. **Code** $\mathcal{X}$

Code Rate: $\boxed{R = \frac{k}{n}}$

Explanation:

- **Code Rate** (R) is the fraction of a codeword used for information:
  - k: Number of information bits.
  - n: Total number of bits in the codeword (information + redundancy).
- **Trade-off**:
  - Higher R ($R \to 1$): More efficient but less error protection.
  - Lower R ($R \to 0$): Less efficient but better error correction.

Linear Block Code Definition: $\boxed{\mathcal{X} = \{\underline{v} = \underline{u}^\top G,\ \underline{u} \in \mathbb{F}_2^k\}}$

Explanation:

- $\mathcal{X}$: The **set of all codewords** in the code (codebook).
- $\underline{u}$: A binary **message vector** of length k ($\underline{u} \in \mathbb{F}_2^k$).
- G: The **generator matrix** ($k \times n$) used to map $\underline{u}$ to a codeword.
- $\underline{v}$: A binary **codeword** of length n ($\underline{v} \in \mathbb{F}_2^n$).

Key Points:

1. Each $\underline{u}$ maps to a unique $\underline{v}$ via $\underline{v} = \underline{u}^\top G$.
2. The total number of codewords is 2^k (one for each $\underline{u}$).
3. $\mathcal{X}$ is a **linear subspace** of dimension k in $\mathbb{F}_2^n$.

Compact Summary: $\mathcal{X}$ is the **codebook** of a linear block code, where each codeword $\underline{v}$ is generated by multiplying a k-bit message vector $\underline{u}$ with the generator matrix G.

- **Generator Matrix**: G

Code Size: $|\mathcal{X}| = 2^k$

- k-bit message vectors ($\underline{u} \in \mathbb{F}_2^k$) are mapped to n-bit codewords ($\underline{v} = \underline{u}^\top G$) via the generator matrix G.
- The codebook $\mathcal{X}$ forms a linear subspace with 2^k unique codewords, corresponding to the k-bit input combinations.
- **Minimum Hamming Distance**:
  $d_{\min} = \min\{d_H(x_i, x_j)\},\ \forall x_i, x_j \in \mathcal{X},\ x_i \neq x_j$ (Hamming distance)
- **Linear Code Minimum Distance**:
  $d_{\min} = \min\left(\substack{x_i, x_j \in \mathcal{X} \\ x_i \neq x_j}\right)\{d_H(x_i, x_j)\} = \min x \in \mathcal{X}\{W_H(\underline{x})\}$

- **Parity Check Relation**:
  $\underline{v}H^\top = 0 \implies \underline{v} \in \mathcal{X} \subseteq \mathbb{F}_2^n$

2.1.4 Parity Check Matrix

- $H \in \mathbb{F}_2^{(n-k)\times n}$
- $\dim(\mathrm{Im}(G)) = k$
- $H = \mathrm{null}(G^\top),\ \dim = n - k$

[6]:
```
using LinearAlgebra

# Define the parity check matrix H (size 3x7 for (7, 4) code)
H = [
    1 0 0 1 1 1 0;
    0 1 0 1 1 0 1;
    0 0 1 1 0 1 1
]

# Define the generator matrix G
G = [
    1 0 0 0 1 1 0;
    0 1 0 0 1 0 1;
    0 0 1 0 0 1 1;
    0 0 0 1 1 1 1
];
```

[7]:
```
# Message vector
u̲ = [1 0 1 0]

# Generate codeword
v̲ = mod.(u̲ * G, 2)
println("Generated codeword: ", v̲)  # Should be a valid codeword
```

```
Generated codeword: [1 0 1 0 1 0 1]
```

[8]:
```
# Parity check validation
parity_check = mod.(v̲ * H', 2)
println("Parity check result: ", parity_check)  # Should be [0, 0, 0]
```

```
Parity check result: [0 0 0]
```

[9]:
```
using LinearAlgebra

# Define the generator matrix G (4x7 for (7,4) code)
G = [
    1 0 0 0 1 1 0;
    0 1 0 0 1 0 1;
    0 0 1 0 0 1 1;
    0 0 0 1 1 1 1
```

```
]

# Generate all possible messages (4-bit binary combinations)
messages = [bitstring(i)[end-3:end] for i in 0:2^4-1]  # 4-bit binary strings
u̲ = [parse.(Int, split(m, ""))' for m in messages]; @show u̲;  # Convert to row vectors (1x4) u\underbar

# Generate all codewords using G
 = [mod.(u * G, 2) for u in u̲ ]; @show

# Define a function to compute Hamming distance
function hamming_distance(v  , v )
    sum(v  .!= v )  # Count differing elements
end

# Compute the minimum Hamming distance using a comprehension
d   = minimum(
    hamming_distance( [i],  [j]) for (i, j)
        in Iterators.product(1:length( ), 1:length( )) if i < j
)

# Output the result
println("Minimum distance of the code: ", d  )
```

```
u̲ = Adjoint{Int64, Vector{Int64}}[[0 0 0 0], [0 0 0 1], [0 0 1 0], [0 0 1 1],
[0 1 0 0], [0 1 0 1], [0 1 1 0], [0 1 1 1], [1 0 0 0], [1 0 0 1], [1 0 1 0], [1
0 1 1], [1 1 0 0], [1 1 0 1], [1 1 1 0], [1 1 1 1]]
 = [[0 0 0 0 0 0 0], [0 0 0 1 1 1 1], [0 0 1 0 0 1 1], [0 0 1 1 1 0 0], [0 1 0
0 1 0 1], [0 1 0 1 0 1 0], [0 1 1 0 1 1 0], [0 1 1 1 0 0 1], [1 0 0 0 1 1 0], [1
0 0 1 0 0 1], [1 0 1 0 1 0 1], [1 0 1 1 0 1 0], [1 1 0 0 0 1 1], [1 1 0 1 1 0
0], [1 1 1 0 0 0 0], [1 1 1 1 1 1 1]]
Minimum distance of the code: 3
```