Aula 18 - Camada de Enlace: Introdução e Correção de Erros

Diego Passos

Universidade Federal Fluminense

Redes de Computadores

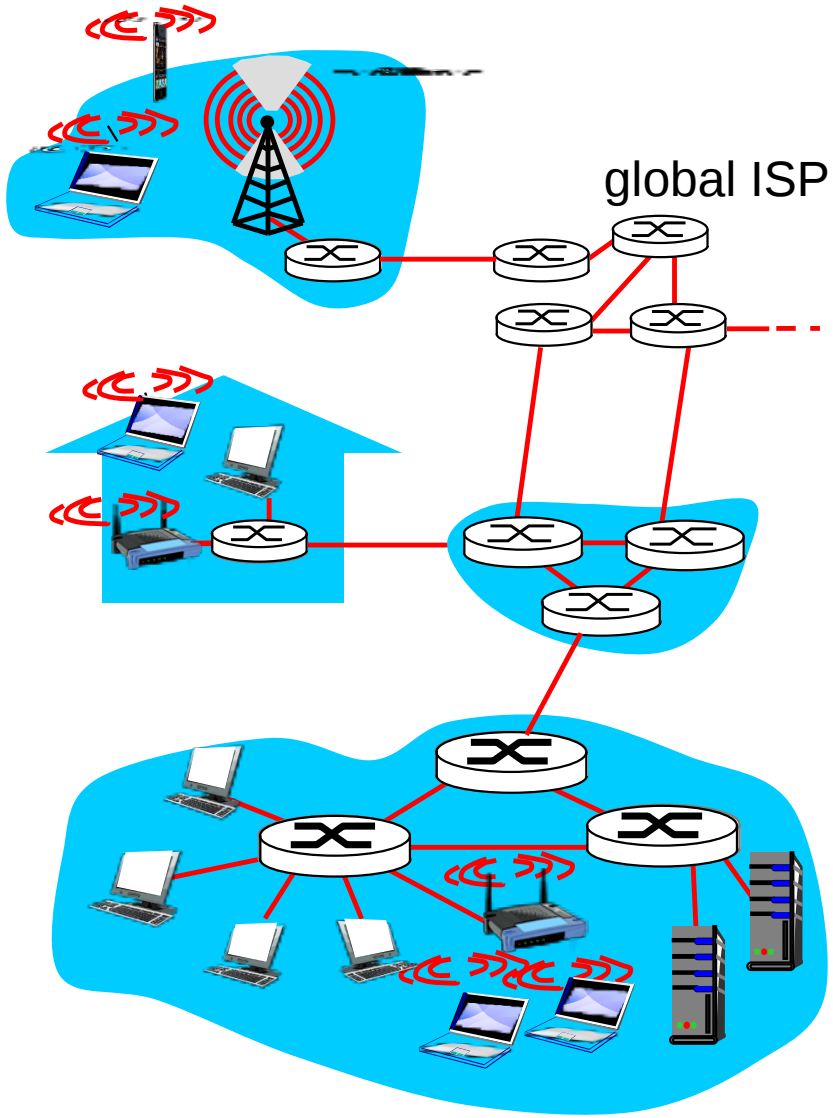
Introdução e Serviços

Introdução

- Terminologia:
  - Roteadores e *hosts*: **nós**.
  - Canais de comunicação conectando nós adjacentes: **enlaces ou links**.
    - Sem fio.
    - Cabeados.
  - Pacote do nível 2: **quadro**, encapsula o datagrama.

Responsabilidade

Transferência dos datagramas entre nós conectados fisicamente por um enlace.



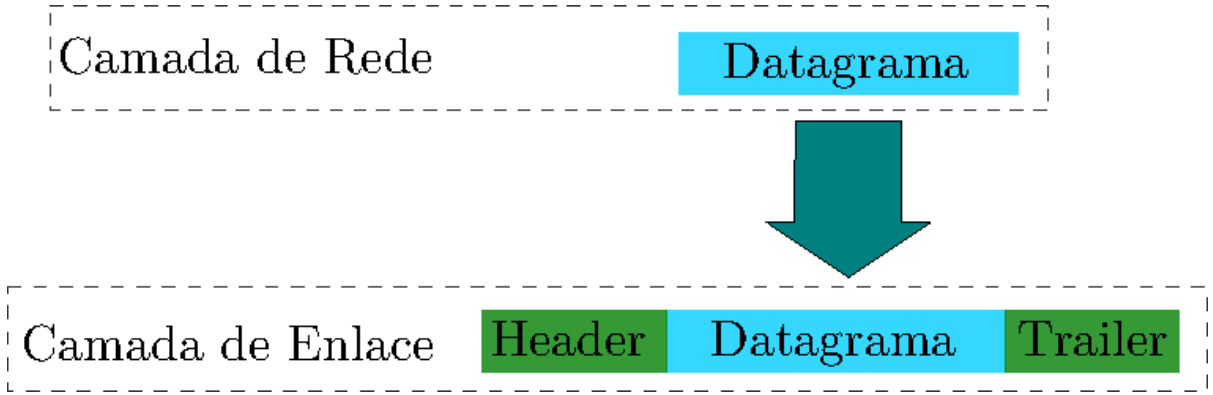
Contextualização

- Datagramas são transmitidos por diferentes enlaces e protocolos.
  - *e.g.*, Ethernet, ADSL, 802.11, *Frame Relay*, ...
- Protocolos têm suas diferenças.
  - Podem ou não prover certos serviços.
  - *e.g.*, transmissão confiável.

- Analogia de transporte de pessoas:
  - Viagem de Niterói para Gramado.
    - Taxi até o aeroporto.
    - Avião até Porto Alegre.
    - Ônibus até Gramado.
- Turista = datagrama.
- Trechos da viagem = enlaces.
- Meio de transporte = protocolo.
- Agência de viagem = algoritmo de roteamento.

Serviços da Camada de Enlace (I)

- **Encapsulamento**.
  - Encapsula datagrama em um **quadro**.
  - Adiciona informações relevantes à camada de enlace.
    - *Header* (cabeçalho), *trailer*.



- **Acesso ao enlace**.
  - Meio **dedicado** *vs.* **compartilhado**.
  - Se múltiplos nós competem, necessitam **coordenação**.

Serviços da Camada de Enlace (II)

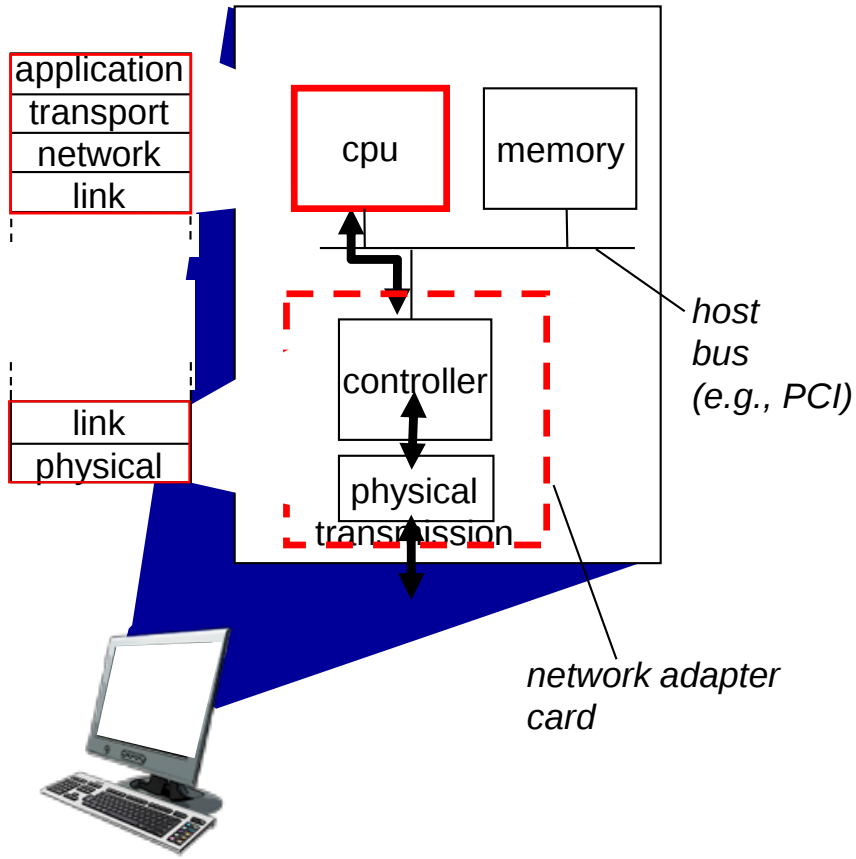
- Endereçamento.
  - **Endereço MAC**.
  - Identifica origem, destino no cabeçalho do quadro.
    - Em algumas redes, pode haver mais elementos endereçados.
  - **Diferente do endereçamento da camada de rede.**
    - *e.g.*, endereço IP.
- Entrega confiável de dados.
  - Já estudado em Redes I (camada de transporte, TCP).
  - Pouco usado em enlaces com poucos erros: fibras ópticas, alguns tipos de par-trançado.
  - Importante para enlaces sem fio: altas taxas de erro.
    - **Por que implementar no nível de enlace e fim a fim?**

Serviços da Camada de Enlace (III)

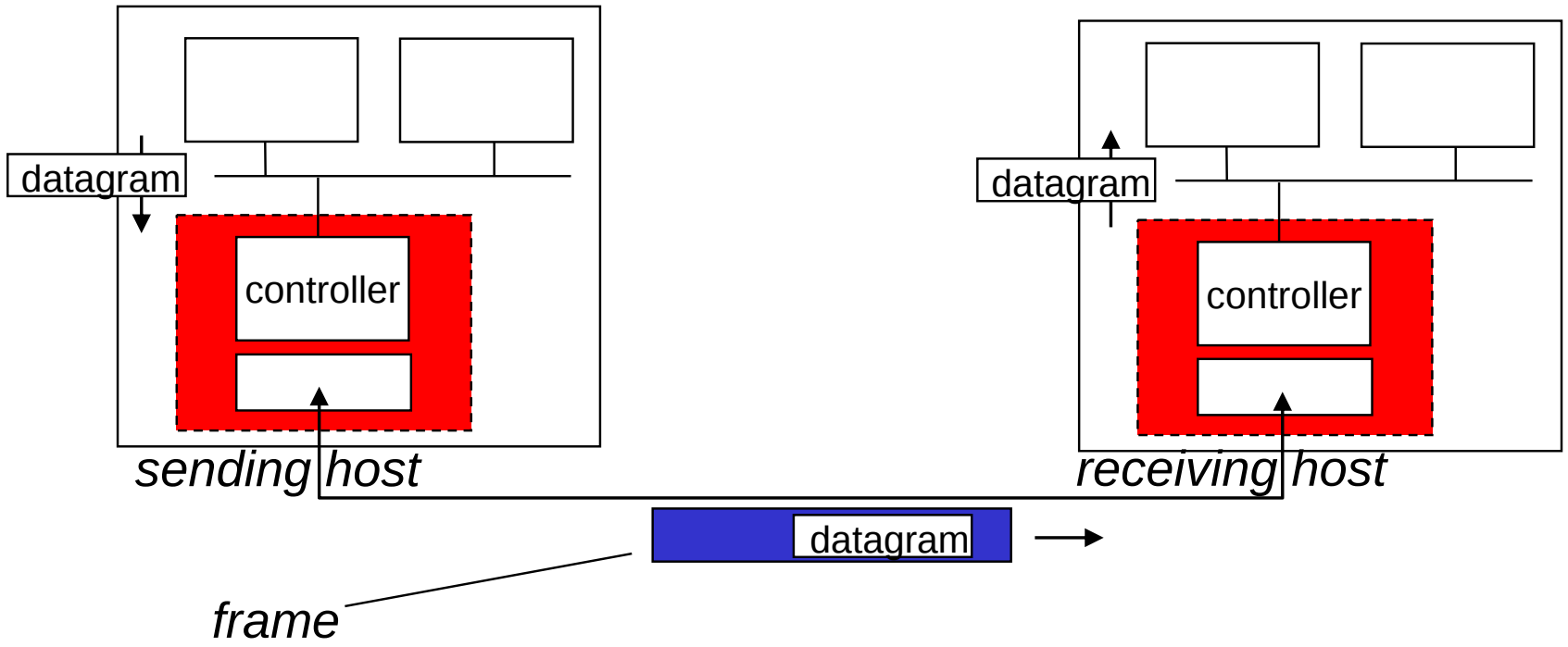
- Detecção de erros.
  - Erros causados por atenuação de sinal, ruído.
  - Receptor deve ser capaz de detectá-los.
    - Pede retransmissão ou descarta o quadro.
- Correção de erros.
  - Passo além do anterior.
  - Receptor identifica **e corrige** erros.
    - **Sem necessitar de retransmissões**.
- Controle de fluxo.
  - Evitar afogar o receptor.
  - Também discutido no contexto do TCP.
  - Mas aqui, entre dois nós adjacentes.
- Transmissão **full-duplex** ou **half-duplex**.
  - *Duplex* = transmissões em ambos os sentidos.
  - *Half-duplex* = um de cada vez.
  - *Full-duplex* = simultaneamente.

Onde a Camada de Enlace é Implementada?

- Presente em todos os nós.
- Implementada no **adaptador de rede**.
  - Ou placa de rede.
  - Ou NIC.
- Exemplos de adaptadores de rede:
  - Placa Ethernet, placa 802.11.
  - Implementam tanto a camada de enlace, quanto camada física.
- Adaptadores se conectam a barramentos.
  - PCI, USB, ...
- Implementação pode também envolver *software*, *firmware*.



Comunicação entre Adaptadores

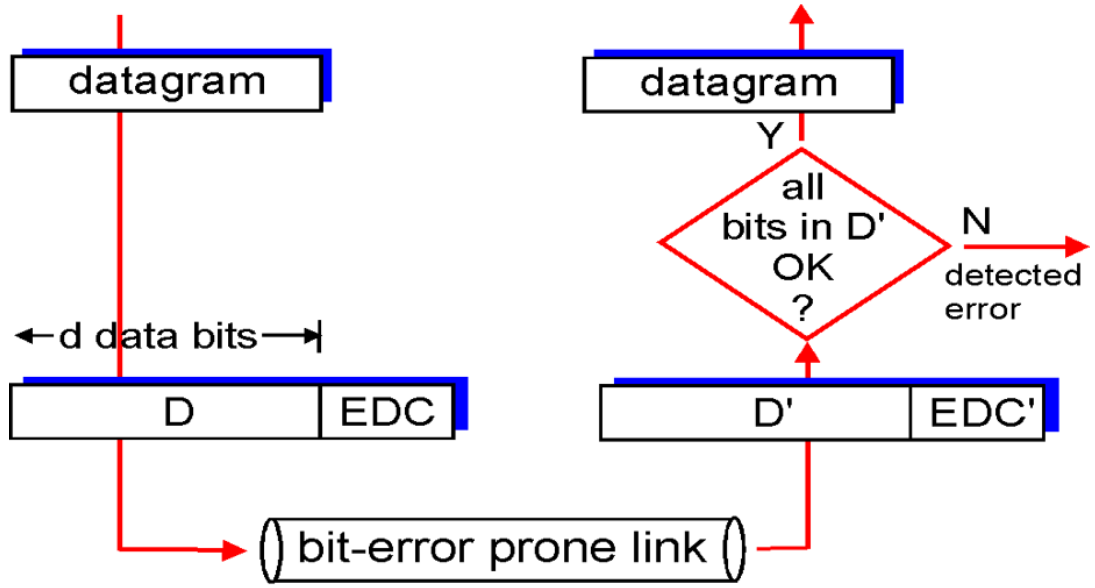


- Transmissor:
  - Encapsula datagrama em quadro.
  - Adiciona informações para verificação/correção de erros, controle de fluxo, transmissão confiável, ...
- Receptor:
  - Verifica/recupera erros, transmissão confiável de dados, ...
  - Extrai datagrama, repassa para camadas superiores.

Detecção e Correção de Erros

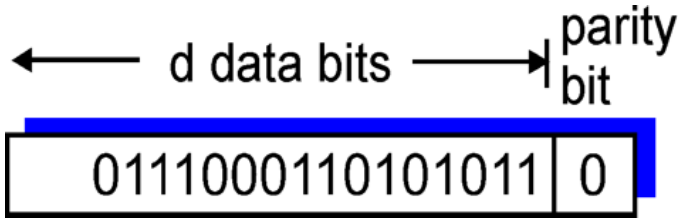
Detecção de Erros

- EDC: bits de detecção e correção de erros.
  - Redundância.
- D: dados protegidos pela redundância.
  - Pode incluir campos de cabeçalho.
- Detecção de erros é probabilística.
  - Pode falhar, mas geralmente com probabilidade baixa.
  - Quanto mais bits no EDC, menor a probabilidade.



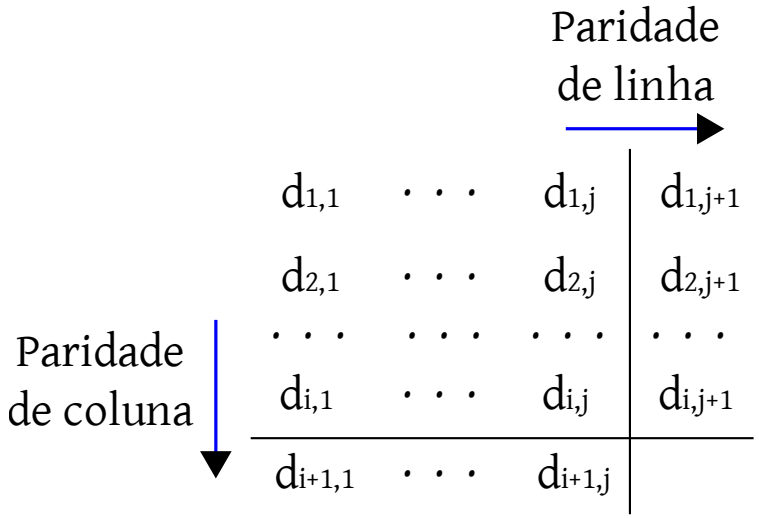
Paridade

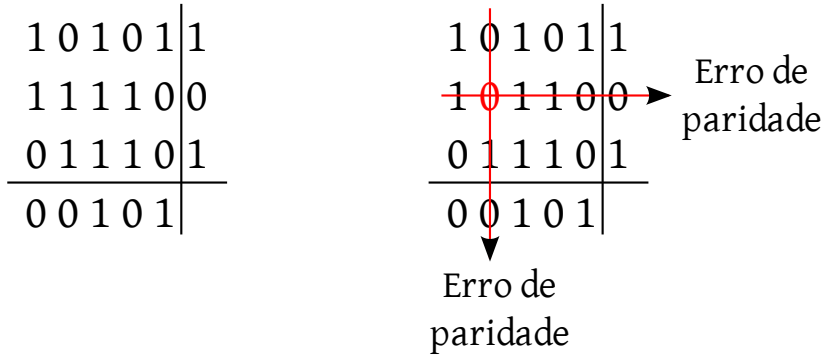
- Um único bit de paridade:



- Paridade par *vs.* paridade ímpar.

- Paridade bi-dimensional:





Paridade: Detecção *vs.* Correção

- Ambas a paridade simples e a bi-dimensional garantem a **detecção** de certos tipos de erro.
  - Paridade simples detecta erros quando há um número **ímpar** de bits errados.
  - Paridade bi-dimensional consegue, **adicionalmente**, detectar alguns casos de erros com número **par** de bits errados.
- Mas funcionalidade da paridade simples para neste ponto.
  - *i.e.*, não é possível saber qual ou quais bits estão errados.
- Já a paridade bi-dimensional tem a capacidade de **corrigir** certos tipos de erro.
  - Quais?
- Se há um único bit errado, receptor **sabe** exatamente **onde** está o erro e **pode corrigi-lo**.
  - Sem necessidade de retransmissão.
  - Capacidade conhecida como **FEC** (*Forward Error Correction*).

Internet Checksum (Revisão)

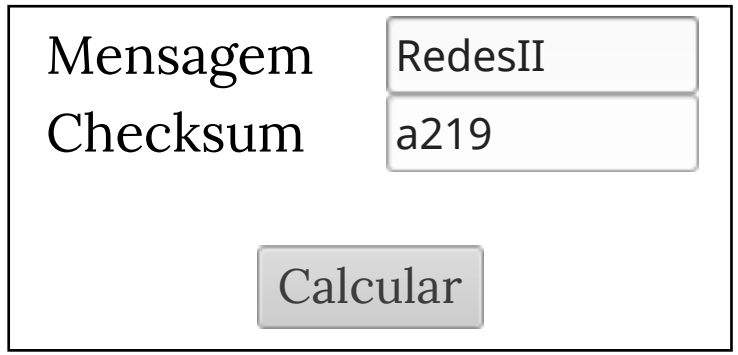
- **Objetivo:** detectar "erros" (*e.g.*, bits com valor trocado) no pacote transmitido.
  - Usado por protocolos de transporte.

- **Transmissor**
  - Trata mensagem como sequência de números de 16 bits.
  - Números são somados em complemento a 1.
    - Vai-um é somado de volta ao número.
    - Ao final, bits são invertidos.
  - Resultado é armazenado na mensagem.

- **Receptor**
  - Computa o checksum da mensagem recebida.
  - Compara o valor computado com o valor encontrado na mensagem.
    - Diferentes? Erro detectado.
    - Iguais? Nenhum erro **detectado** (mas não há mesmo erros?).

Internet Checksum: Exemplos

- Experimente o cálculo do *checksum* de algumas mensagens (*strings*):



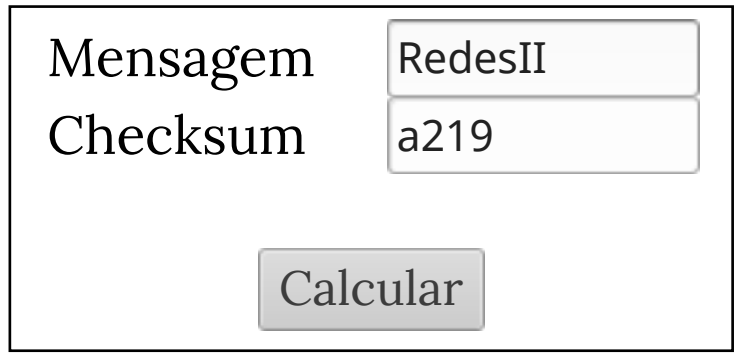
- Sugestão: calcule o *checksum* de “casa”.
  - Resultado: 0x3d29.
  - Em ASCII: 0x3D → “=”.
  - Em ASCII: 0x29 → “)”.
- **Pergunta:** qual é o *checksum* de “casa)=”?

Internet Checksum: Detecção *vs.* Correção

- Checksum só é capaz de **detectar** erros.
- Mas não é capaz de **corrigir** erros.
- Caso típico:
  - Mensagem chega ao receptor.
  - Receptor calcula o checksum.
  - Valor calculado é comparado ao recebido.
  - Valores são diferentes ⇒ mensagem é **completamente descartada**.

Internet Checksum: Falso Negativo

- Checksum pode não detectar certos erros.
  - *i.e.*, pode considerar certa uma mensagem com erros.
- Em outras palavras: mensagem errada pode ter o mesmo checksum da mensagem certa!
- Exemplo (com pequenas strings):
  - “testar” - 0xb3b6
  - “tfssar” - 0xb3b6
  - “settar” - 0xb3b6
  - “reutar” - 0xb3b6
  - ...



Internet Checksum: Falso Positivo

- Valor do checksum é transmitido junto do resto da mensagem.
- **Pergunta:** se a mensagem pode ser corrompida, por que o checksum não pode?
- **Resposta:** ele pode!
- O que acontece quando o valor do checksum é corrompido?

- Primeira possibilidade: apenas o valor do checksum muda.
  - Exemplo: mensagem “testar” e checksum 0xb3b7.
  - Dados estão **corretos**, mas checksum não bate: mensagem correta é descartada!

- Segunda possibilidade: tanto o valor do checksum, quanto mensagem são corrompidos.
  - Mais provável: checksum não irá bater, mensagem será (corretamente) descartada.
  - Menos provável: checksum irá bater, mensagem será (erroneamente) aceita!
    - Exemplo: mensagem “tertar”, checksum 0xb3b7.

Cyclic redundancy check (CRC)

- Método mais poderoso de detecção de erro.
  - Resultado de uma "conta", como o Checksum.
  - Mas posição dos bits tem maior influência no resultado final.
    - **Menos provável** que certos erros comuns não sejam detectados.

- **Ideia**
  - Trata msg **M** como um polinômio **D**.
  - Bits são coeficientes:

    M = 1 1 0 1 0

    $D = x^4 + x^3 + x^1$
  - Escolhe um *polinômio gerador* **G** (grau **r**).
  - Encontra um polinômio **R** de grau menor que r, tal que:

    $G \mid (D \cdot x^r + R)$

- **Aspectos práticos**
  - Durante a divisão, operações sobre os coeficientes são feitas em módulo 2.
    - $1 - 1 = 0$
    - $1 - 0 = 1$
    - $0 - 0 = 0$
    - $0 - 1 = 1$
  - Coeficientes de R são adicionados à mensagem como o CRC (**sempre r bits**).
  - Receptor trata msg recebida (incluindo CRC) como polinômio e testa divisibilidade.

CRC: Exemplo

- Considere:
  - $M = 101110$.
  - $G = x^3 + 1$.
    - $D = x^5 + x^3 + x^2 + x \implies D \cdot x^3 = x^8 + x^6 + x^5 + x^4$.

- **Pensando em polinômios:**

$$
\begin{array}{r|l}
x^8 + x^6 + x^5 + x^4 & x^3 + 1 \\
\cline{2-2}
-x^8 \quad\quad - x^5 & x^5 + x^3 + x + 1 \\
\cline{1-1}
x^6 + x^4 & \\
-x^6 \quad\quad - x^3 & \\
\cline{1-1}
x^4 + x^3 & \\
-x^4 \quad\quad - x & \\
\cline{1-1}
x^3 + x & \\
-x^3 \quad\quad - 1 & \\
\cline{1-1}
x + 1 &
\end{array}
$$

- $R = x + 1 \implies \text{CRC} = 011$.

CRC: Exemplo (II)

- Alternativa: **pensando diretamente nos bits**

- **Transmissor**

	Msg						CRC		
	1	0	1	1	1	0	0	0	0
⊕	1	0	0	1					
	0	0	1	0	1	0	0	0	0
⊕			1	0	0	1			
	0	0	0	0	1	1	0	0	0
⊕					1	0	0	1	
	0	0	0	0	0	1	0	1	0
⊕						1	0	0	1
	0	0	0	0	0	0	**0**	**1**	**1**

- **Receptor**

	Msg						CRC		
	1	0	1	1	1	0	0	1	1
⊕	1	0	0	1					
	0	0	1	0	1	0	0	1	1
⊕			1	0	0	1			
	0	0	0	0	1	1	0	1	1
⊕					1	0	0	1	
	0	0	0	0	0	1	0	0	1
⊕						1	0	0	1
	0	0	0	0	0	0	**0**	**0**	**0**

CRC: Uso

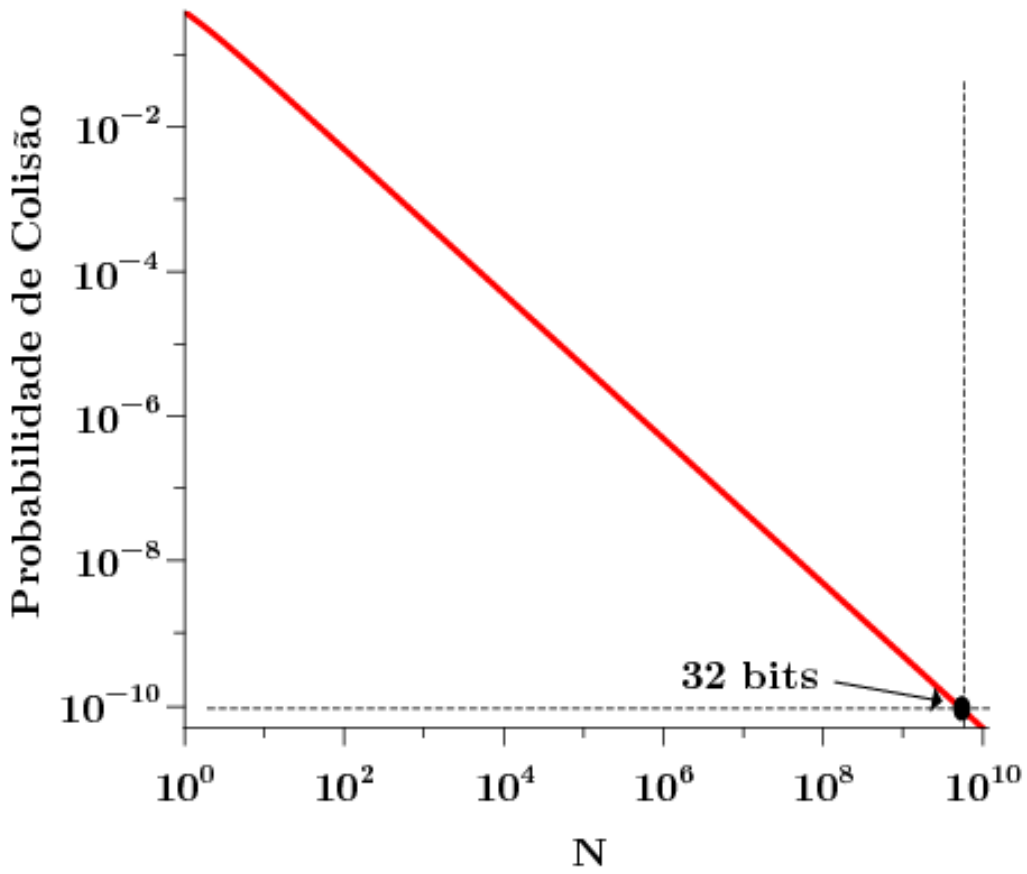
- Computacionalmente simples.
- Efetivo em detectar erros comuns.
- Parametrizável.
  - Número de bits (grau do polinômio gerador).
  - Polinômio gerador em si.
- **Ampla adotação** em protocolos da camada de enlace.
  - Ethernet, 802.11 (Wi-Fi), ...
- **Escolha do polinômio** gerador é importante.
  - Alguns são melhores que outros.
- Para evitar escolhas ruins, há **CRCs padronizados**.
  - CRC16—IBM, **CRC32**, CRC32—C, CRC40—GSM, ...

CRC: Probabilidade de Falha

- CRC pode falhar?
  - Sim! Mesmos casos vistos para o Checksum, se aplicam.
  - **Em particular, duas mensagens diferentes podem ter o mesmo CRC.**
  - Estamos "resumindo" mensagem em poucos (*e.g.* 32) bits.
  - Há mais combinações de mensagens que valores de CRC.
- O quão provável é isso?
  - Assumindo uma "boa" função de espalhamento, probabilidade de **colisão**:

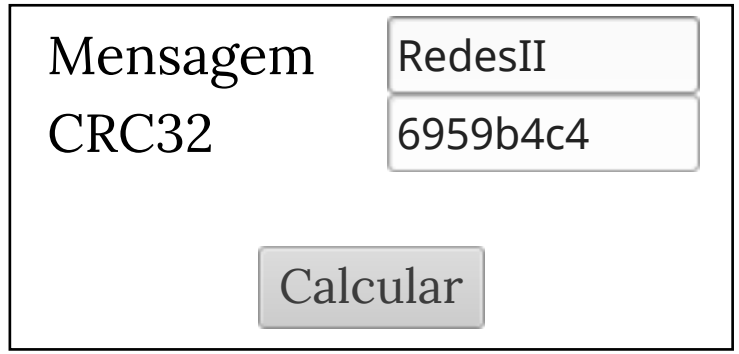
$$\approx 1 - e^{\frac{-2}{2N}}$$

  - N: # de diferentes valores de CRC.



CRC: Exemplo de Colisão

- Alguns pequenos exemplos de colisão do CRC32:
  - “plumless” - 0x4ddb0c25.
  - “buckeroo” - 0x4ddb0c25.



CRC *vs.* Checksum (I)

- CRC é mais eficiente em detectar erros.
  - Justifica seu emprego comum em protocolos da camada de enlace.
- Mas por que, então, o checksum é usado nas camadas superiores?

CRC *vs.* Checksum (II)

- CRC é mais eficiente em detectar erros.
  - Justifica seu emprego comum em protocolos da camada de enlace.
- Mas por que, então, o checksum é usado nas camadas superiores?
  - Cálculo do CRC é mais complexo computacionalmente.
  - Já o checksum é rápido.
  - Lembre-se:
    - Camada de enlace é normalmente implementada em *hardware* especializado.
    - Camadas de rede, transporte são geralmente implementadas em *software*.
  - Conclusão: implementação do CRC é "mais viável" na camada de enlace.
    - Ainda relevante hoje?