

Aula 2 - Camada de Enlace: Introdução e Correção de Erros

Diego Passos

Universidade Federal Fluminense

Redes de Computadores II

Camada de Enlace

- Tema do capítulo 5 do livro (Kurose).
- Objeto das próximas 7 aulas da disciplina.
- Objetivos:
 - Entender os princípios dos serviços da camada de enlace (ou MAC).
 - Correção e detecção de erros.
 - Uso de canais compartilhados: acesso múltiplo.
 - Endereçamento na camada de enlace.
 - Redes locais: Ethernet e VLANs.
 - Implementação de diversas tecnologias da camada de enlace.

Agenda do Capítulo

- Introdução e serviços.
- Detecção e correção de erros.
- Protocolos de acesso múltiplo.
- Redes locais:
 - Endereçamento, ARP.
 - Ethernet.
 - *Switches*.
 - VLANs.
- Redes de *Data Centers*.
- Exemplo: uma requisição web.

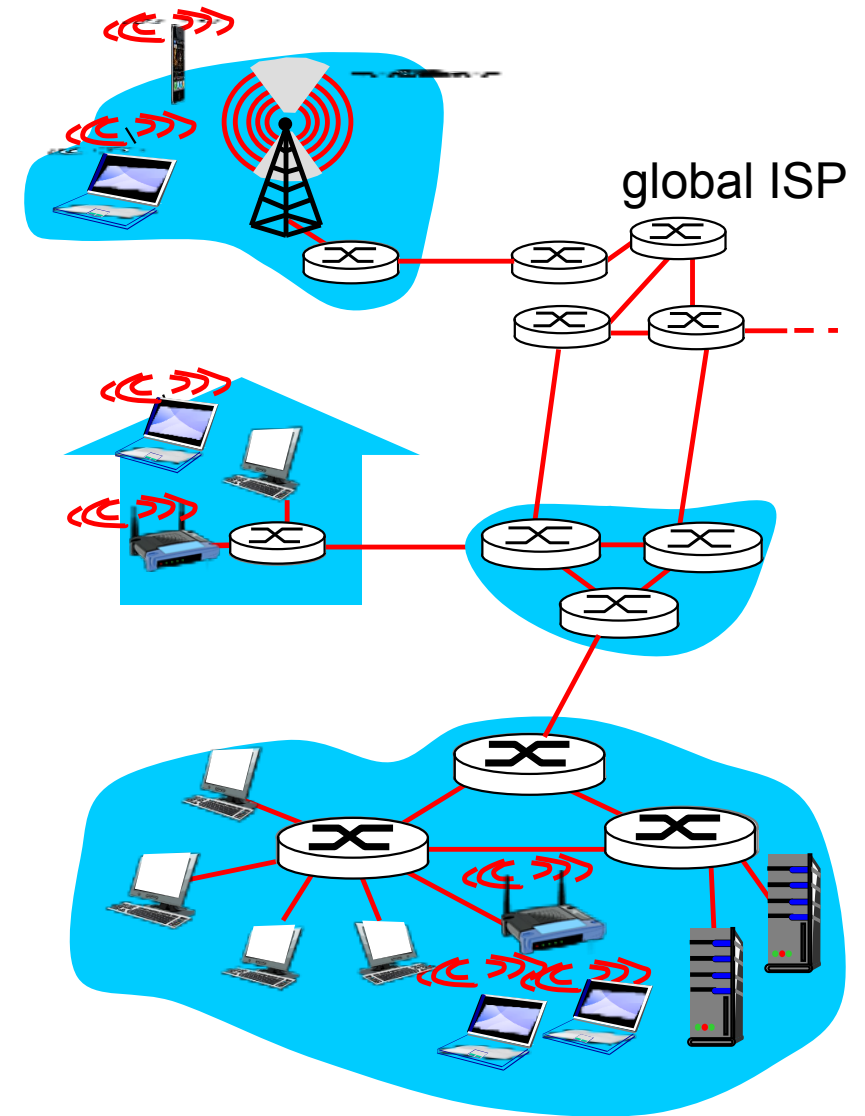
Introdução e Serviços

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.



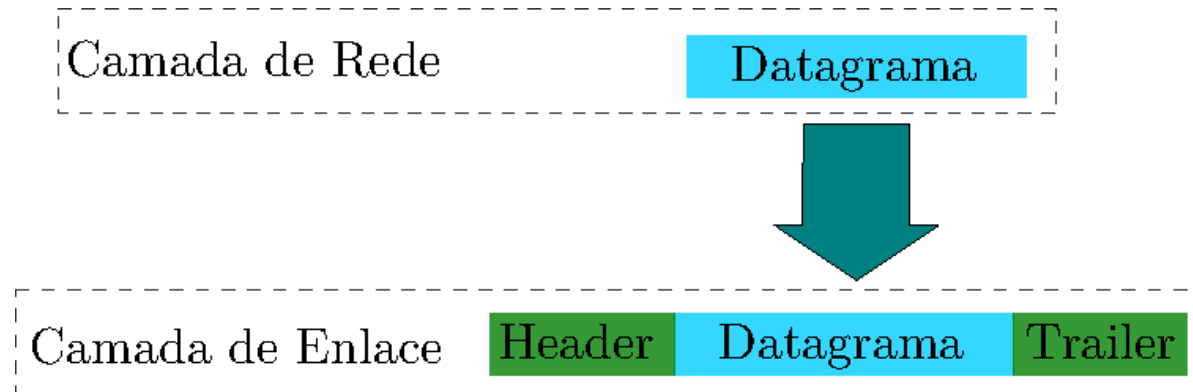
Contextualização

- 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)

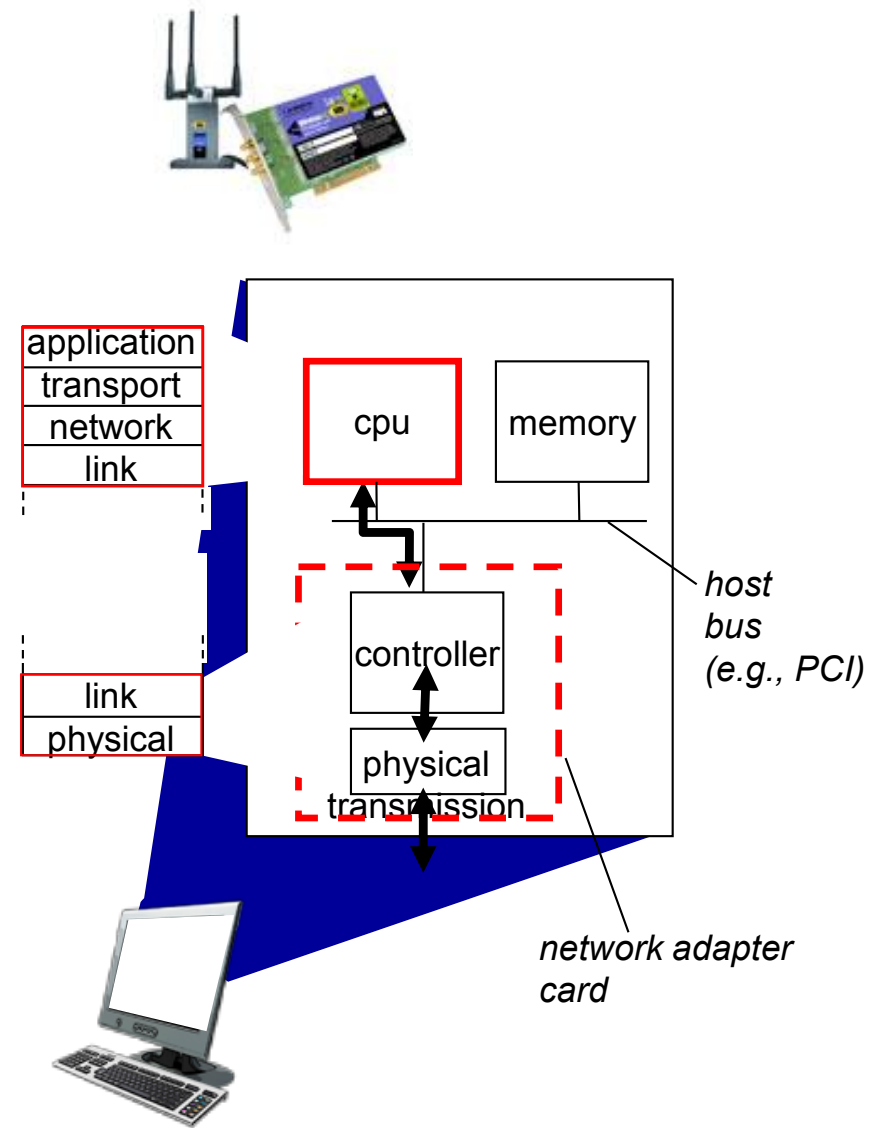
- Endereçamento.
 - **Endereço MAC.**
 - Identifica origem, destino no cabeçalho do quadro.
 - Em algumas redes, pode haver mais elementos endereçados.
 - **Diferente do endereçamento da camada de rede.**
 - *e.g.*, endereço IP.
- Entrega confiável de dados.
 - Já estudado em Redes I (camada de transporte, 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)

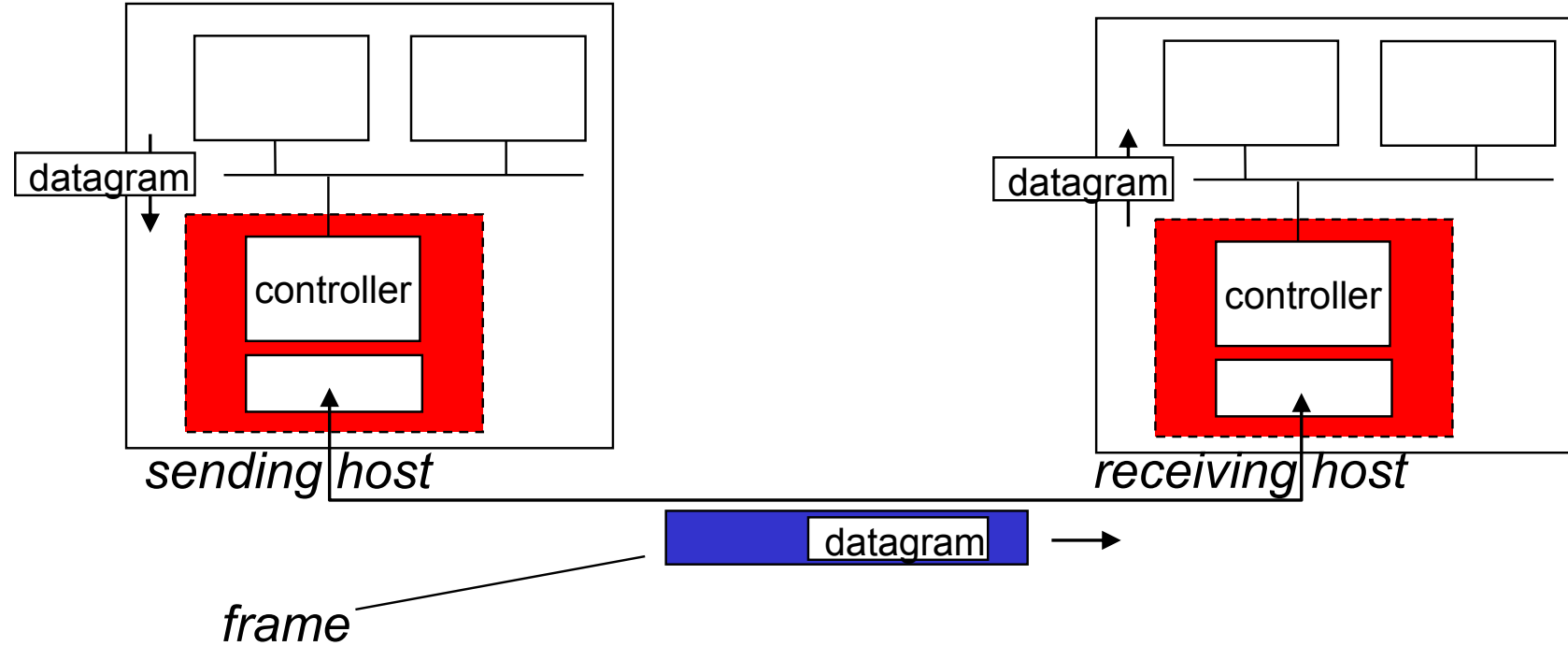
- Detecção de erros.
 - Erros causados por atenuação de sinal, ruído.
 - Receptor deve ser capaz de detectá-los.
 - Pede retransmissão ou descarta o quadro.
- Correção de erros.
 - Passo além do anterior.
 - Receptor identifica **e corrige** erros.
 - **Sem necessitar de retransmissões.**
- Controle 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, ...

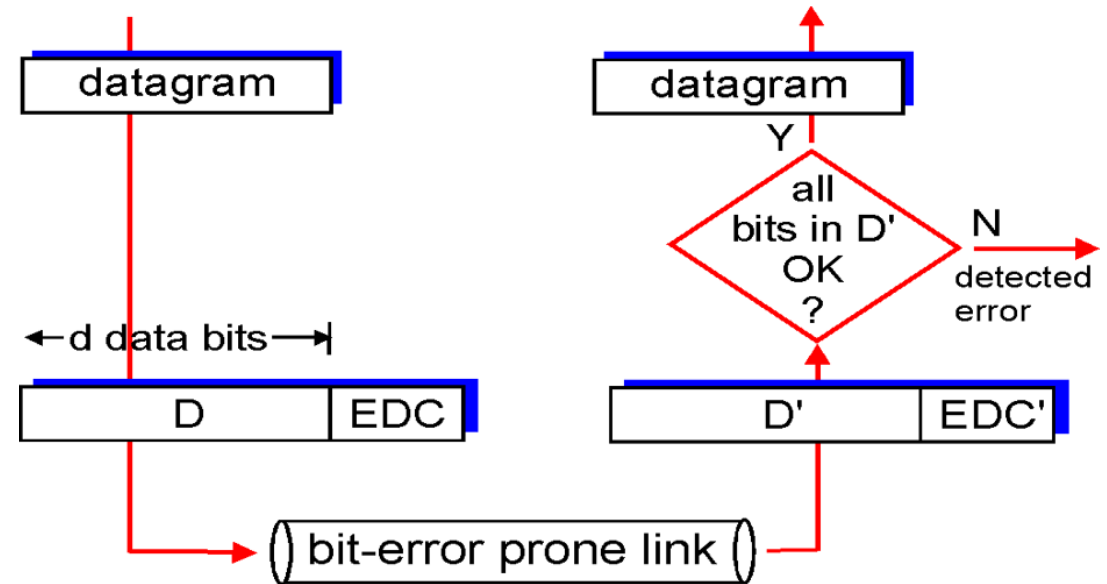
- Receptor:

- Verifica/recupera erros, transmissão confiável de dados, ...
- Extrai datagrama, repassa para camadas superiores.

Detecção e Correção de Erros

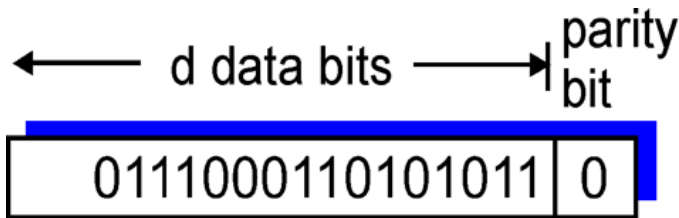
Detecção de Erros

- 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.



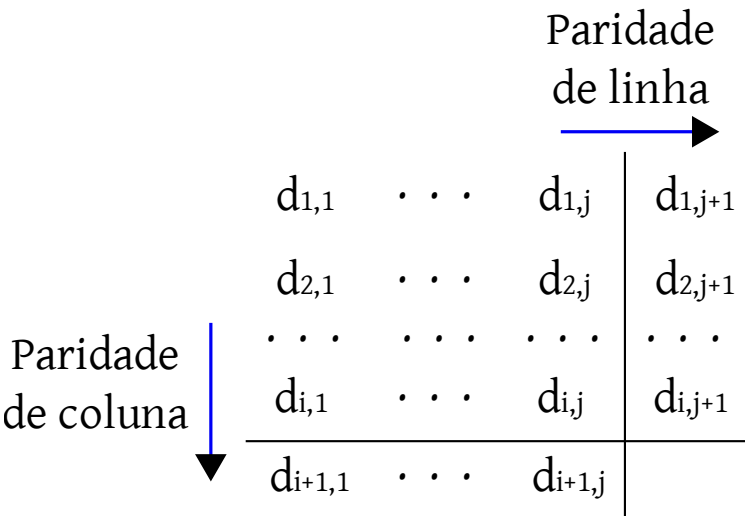
Paridade

- Um único bit de paridade:



- Paridade par vs. paridade ímpar.

- Paridade bi-dimensional:



1	0	1	0	1	1
1	1	1	1	0	0
0	1	1	1	0	1
0	0	1	0	1	

Sem Erros

1	0	1	0	1	1
1	0	1	1	0	0
0	1	1	1	0	1
0	0	1	0	1	

Erro de
paridade

Erro único
corrigível

Erro de
paridade

Paridade Bidimensional: Mais Exemplos

Configurações

Linhas	<input type="text" value="3"/>
Colunas	<input type="text" value="3"/>
Convenção	<div>Par ▼</div>

Transmissor

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	

Receptor

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	

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*):

Mensagem	<input type="text" value="RedesII"/>
Checksum	<input type="text" value="a219"/>
<input type="button" value="Calcular"/>	

- 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 \Rightarrow 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
 - ...

Mensagem	<input type="text" value="RedesII"/>
Checksum	<input type="text" value="a219"/>
<input type="button" value="Calcular"/>	

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:
 $M = 1 \ 1 \ 0 \ 1 \ 0$
 $D = x^4 + x^3 \quad + x^1$
 - Escolhe um *polinômio gerador* **G** (grau **r**).
 - Encontra um polinômio **R** de grau menor que r , tal que:
 $G | (D \cdot x^r + R)$
- **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:**

$$\begin{array}{r} x^8 + x^6 + x^5 + x^4 \\ -x^8 \qquad -x^5 \\ \hline x^6 \qquad + x^4 \\ -x^6 \qquad -x^3 \\ \hline \qquad + x^4 + x^3 \\ \qquad -x^4 \qquad -x \\ \hline \qquad \qquad x^3 + x \\ \qquad -x^3 \qquad -1 \\ \hline \qquad \qquad \qquad x + 1 \end{array}$$
$$\left| \begin{array}{r} x^3 + 1 \\ \hline x^5 + x^3 + x + 1 \end{array} \right.$$

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

CRC: Exemplo (II)

● Alternativa: pensando diretamente nos bits

● Transmissor

	Msg						CRC		
	1	0	1	1	1	0	0	0	0
\oplus	1	0	0	1					
	0	0	1	0	1	0	0	0	0
\oplus			1	0	0	1			
	0	0	0	0	1	1	0	0	0
\oplus					1	0	0	1	
	0	0	0	0	0	1	0	1	0
\oplus						1	0	0	1
	0	0	0	0	0	0	0	1	1

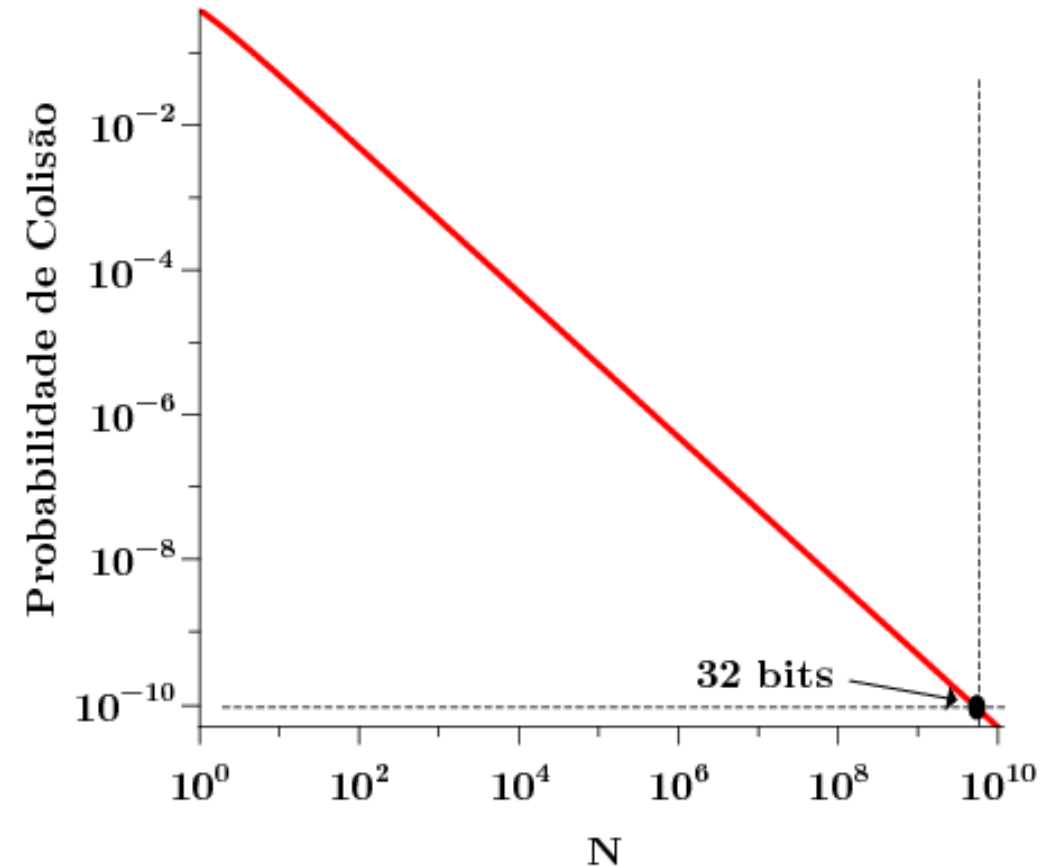
● Receptor

	Msg						CRC		
	1	0	1	1	1	0	0	1	1
\oplus	1	0	0	1					
	0	0	1	0	1	0	0	1	1
\oplus			1	0	0	1			
	0	0	0	0	1	1	0	1	1
\oplus					1	0	0	1	
	0	0	0	0	0	1	0	0	1
\oplus						1	0	0	1
	0	0	0	0	0	0	0	0	0

- Computacionalmente simples.
- Efetivo em detectar erros comuns.
- Parametrizável.
 - Número de bits (grau do polinômio gerador).
 - Polinômio gerador em si.
- **Ampla adoção** 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.

Mensagem	RedesII
CRC32	6959b4c4
<input type="button" value="Calcular"/>	

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?

Resumindo...

- Camada de enlace:
 - Responsável por transferir datagramas entre nós **diretamente conectados**.
 - Encapsula datagramas em unidade chamada de **quadro**.
 - Também gerencia o **acesso ao meio de transmissão**.
 - **Provê endereçamento diferente** do provido pela camada de rede.
 - Pode prover **entrega confiável de dados**.
 - Provê **Deteccção de erros** (e possível correção).
 - Pode prover **controle de fluxo**.
 - Enlace pode ser **half-duplex, full-duplex**.
- Deteccção/correção de erros:
 - Importante, principalmente em **enlaces propensos a perdas**.
 - Deteccção é probabilística, i.e., **pode falhar**.
 - Utiliza bits **redundantes**: quanto mais bits, menor probabilidade de falha, maior o *overhead*.
 - Bit de paridade único: **detecta** um número ímpar de erros.
 - Paridade bi-dimensional: **corrige** único erro.
 - Método simples de **FEC**.
 - Checksum: **detecta** erros.
 - CRC: **detecta** erros melhor que o checksum.

Leitura e Exercícios Sugeridos

- Serviços providos pela camada de enlace:
 - Páginas 318 a 323 do Kurose (Seção 5.1).
 - Exercícios de fixação 2 e 3 do capítulo 5 (Kurose).
- Detecção e correção de erros:
 - Páginas 323 a 328 do Kurose (Seção 5.2).
 - Problemas 1, 2 e 3 do capítulo 5 (Kurose).

Próxima Aula...

- Continuação do estudo da camada de enlace.
- Próximo tópico: protocolos de acesso múltiplo.
 - Quais problemas podem decorrer de múltiplas transmissões simultâneas?
 - Como coordenar o acesso de múltiplos nós a um enlace compartilhado?
 - O quão eficientes são estes protocolos?

Exercício

- Considere um esquema de paridade bidimensional que use 3 linhas e quatro colunas, com paridade par.
- Calcule os bits de paridade para um pacote formado pelos seguintes bits:
 - 11010011 10110010 01001000
- Adicionalmente, insira um erro em um dos bits do pacote e mostre como o método é capaz de detectá-lo.