UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA
INF01154 - Redes de Computadores N

PARTE 1 (CONTROLE DE FLUXO)

1 DEBATE EM AULA (QUESTÕES DEVEM SER RESPONDIDAS NO RELATÓRIO)

1. O que é controle de fluxo? Como é feito em TCP? E em UDP?
2. O que é janela deslizante? Qual a consequência de uma janela grande demais? E pequena demais?
3. Como relacionar tamanho da janela com banda? Como relacionar RTT com banda?
4. Com RTT (*Round Trip Time*) alto, é mais eficiente tamanho de janela grande ou pequena? Por quê?
5. O que é "tempo de timeout"? Qual o problema para determinar o tempo de timeout (*retransmit timer*) em redes reais? Qual a solução?
6. Qual a consequência das perdas num protocolo real, tipo TCP, em termos de controle de fluxo? Outra forma de perguntar: o que o protocolo assume quando tem perdas? Lembre que as perdas acontecem nas duas direções (na transmissão do pacote ou no ACK).

2 EXERCÍCIOS

1. **Fazer a lista de exercícios de debate em aula**
2. **Um canal tem uma taxa de 100 Mbit/s e um retardo de propagação de 20 ms e o programador escolheu utilizar quadros de 1.250 bytes. Para que tamanho de janela o sistema proporciona uma eficiência de pelo menos 50%? Desconsidere os tempos de inserção e desinserção dos pacotes na rede, ou seja, considere que o tempo de propagação é o mesmo para qq tamanho de janela.**
3. **Quadros de 1.250 bytes são enviados de A para B por um canal de 4 Mbit/s usando um satélite geoestacionário. Suponha que o tempo de propagação de A para B seja de 270 ms. Qual é a capacidade máxima do canal utilizando os seguintes mecanismos:**
   a. **Stop-and-wait?**
   b. **Janela deslizante de 10 pacotes?**
   c. **Janela deslizante de 250 pacotes?**

2.1 Experiência (deve aparecer no relatório)

1. Explore o demo Sliding Window em http://www2.rad.com/networks/2004/sliding_window/demo.html. Eventualmente será necessário entrar via web archive (http://web.archive.org). Explique apoiado com imagens da execução da tela do computador o funcionamento do demo sliding window. No mínimo, os seguintes itens devem ser explicados:
   a. Funcionamento de janela deslizante
   b. Comente três diferenças desse simulador em relação ao TCP utilizado na WEB.

2. Acesse o simulador em http://histrory.visualland.net/tcp_swnd.html ou http://histrory.visualland.net/tcp_video.php?video=tcp2%20sliding%20window&protocol=TCP&title=2v.Slidin

g%20Window. Verifique a simulação. Explique o mecanismo de crescimento da janela deslizante, definindo o slow-start e congestion avoidance.

PARTE 2 – Controle de erros

3 RUÍDO E ERROS

Nos dias de hoje, uma das certezas com a tecnologia disponível é a existência de ruído no canal de comunicação, que pode ocasionar eventualmente um ou mais erros na transmissão do sinal.

Ruído pode ser definido como sinais eletrônicos aleatórios que, adicionados ao sinal de informação, podem alterar seu conteúdo.

O canal ou meio físico produz distorções sobre os sinais transmitidos que podem ser classificados em duas grandes classes: *Distorções Lineares* e *Distorções Não Lineares*, ou também chamadas distorções aleatórias. As *Distorções Lineares* são minimizadas através de dispositivos chamados de equalizadores, de forma automática e adaptativa, de acordo com as variações do canal. Já as *Distorções Não Lineares* são de caráter totalmente imprevisível e, por isto, são capazes de provocar erros nas transmissões.

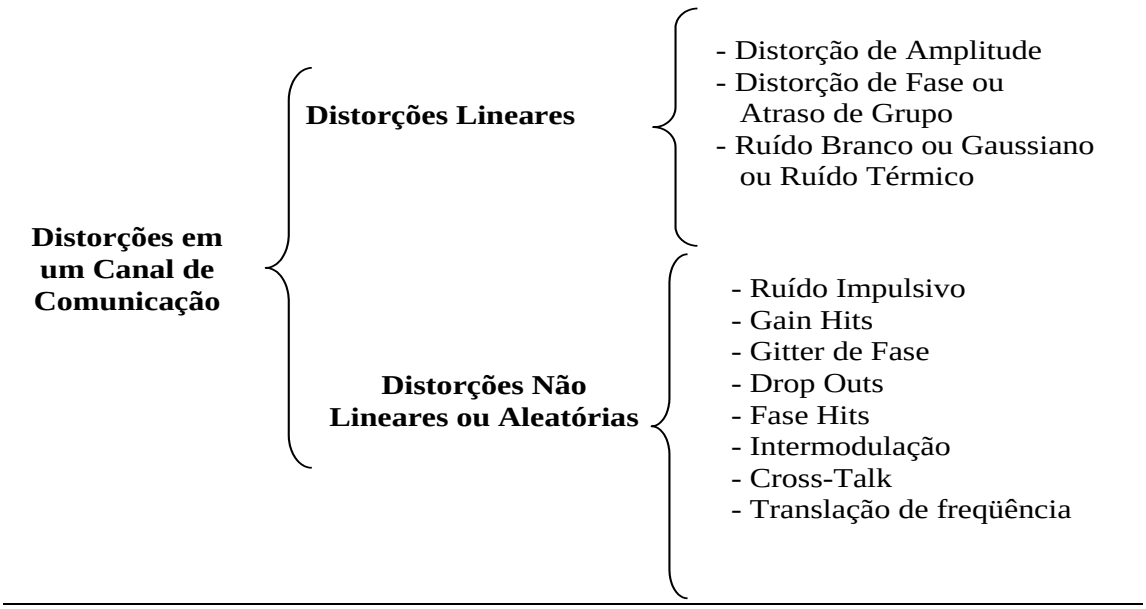


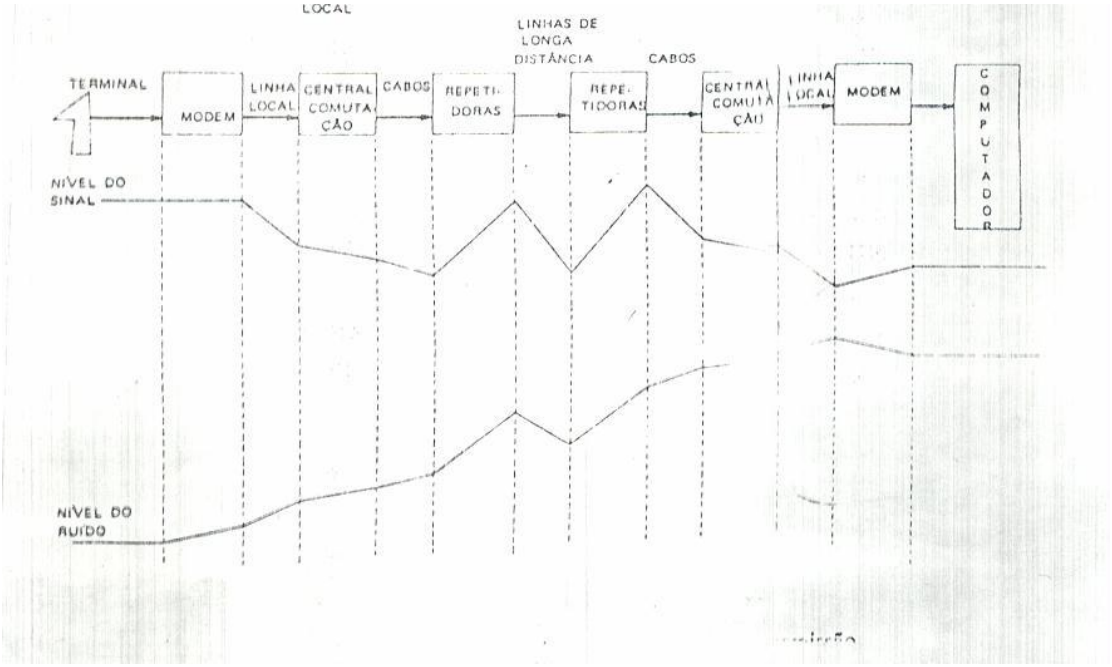
Fig. 1 - Classificação das principais distorções num meio, causadoras de erro

3.1 Ruído branco ou gaussiano

O ruído branco é um sinal cuja amplitude varia em torno de um certo nível, aleatoriamente no tempo, seguindo uma distribuição gaussiana. Em outras palavras, é um sinal que possui componentes em todo o espectro de freqüências de forma igualitária, somando-se ao sinal de dados.

Esse tipo de ruído acontece devido à agitação térmica das moléculas em um dado meio físico, sendo inevitável, pois as moléculas estão em constante movimento. Por este motivo é conhecido também como ruído térmico, sendo diretamente proporcional à temperatura do meio físico.

A figura a seguir ilustra a influência do ruído branco na comunicação de dados. Apesar de prejudicar a comunicação, é um ruído previsível, sendo portanto tratável. Atualmente o sinal é digital, e as repetidoras são regeneradoras, recuperando os níveis de sinal ruído originais.

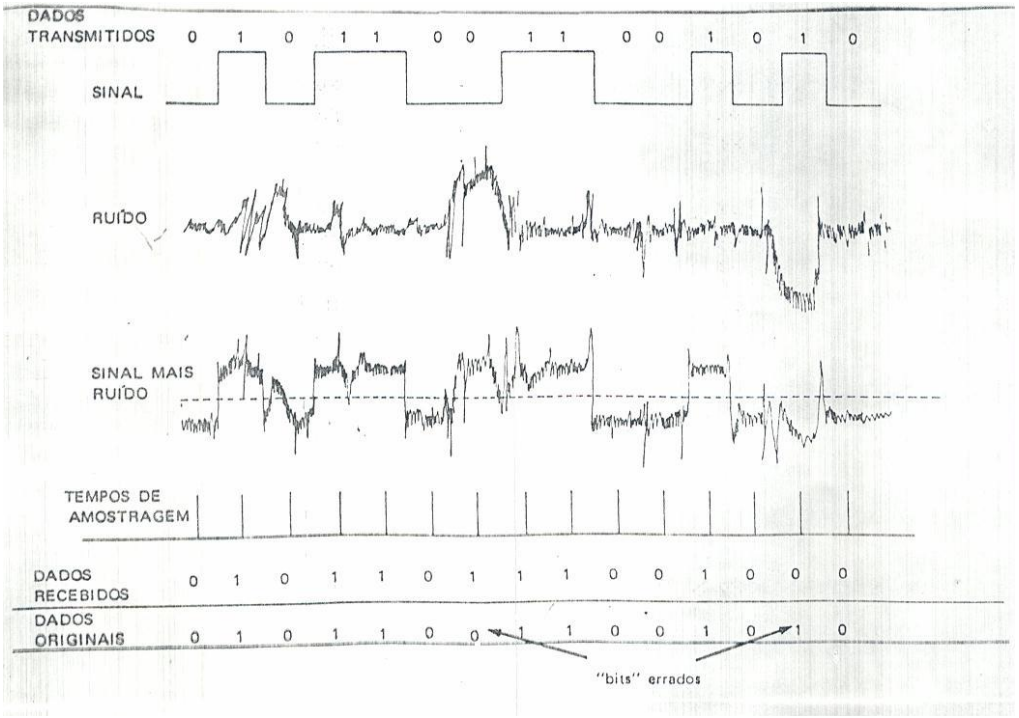


3.2 Ruído impulsivo

O ruído impulsivo ou transiente pode ser definido como qualquer surto de energia que exceda o ruído ambiente numa relação de 13dB em canais de voz. Este ruído pode acontecer devido a diversos fatores, como por exemplo: raios, acionamento de motores, acionamento de lâmpadas fluorescentes, e assim por diante.

A sua principal característica é que não é previsível, variando consideravelmente em amplitude, freqüência e periodicidade de ocorrência. Dessa forma, dificilmente pode-se tratar o ruído impulsivo, fazendo com que aconteçam os erros de transmissão de dados.

A figura a seguir ilustra uma comunicação de dados com a existência de ruído e também erros de transmissão.



Devido aos fatores citados acima (ruído e distorção), atualmente considera-se que os erros são inevitáveis numa transmissão de dados. Portanto, já que não se pode viver sem eles, deve-se encontrar uma forma de diminuir o prejuízo causado pela sua existência. Para isto existem os códigos de detecção e correção de erros, que serão tratados a seguir.

3.3 Características dos erros

- São inevitáveis em qualquer sistema de comunicação real;
- A distribuição dos erros não é homogênea. Algumas vezes acontecem rajadas de erros, com 8 ou mais bits sucessivos errados;
- Deve-se levar em conta o meio físico de transmissão de dados, para incluir maior ou menor redundância na transmissão, a fim de garantir que a informação recebida seja confiável;

Abordagens possíveis no tratamento de erros:

1. Ignorar o erro;
2. Detectar e solicitar a retransmissão em caso de erro;
3. Detectar e corrigir os erros na recepção de forma automática.

4 CÓDIGOS DE DETECÇÃO DE ERROS

São códigos que apenas detectam o erro, sem corrigir automaticamente o mesmo.

4.1 Paridade

O código de paridade é um dos mais simples existentes, consistindo basicamente no transmissor da mensagem adicionar um bit de redundância após um determinado número de bits (normalmente um byte). Este bit de redundância deve deixar a paridade do byte em um determinado tipo (par ou ímpar). Isto significa que, para uma **paridade par**, o número de bits "1" transmitidos deve ser **par** e, para uma **paridade ímpar**, o número de bits "1" transmitidos deve ser **ímpar**.

O exemplo a seguir mostra um único byte e seu controle de paridade. No caso foi utilizado paridade par, ou seja, o número de bits "1" transmitidos deve ser par.

b0	b1	b2	b3	b4	b5	b6	b7	P
1	0	0	0	1	1	0	0	1

Pode-se observar que a paridade não é um método satisfatório do ponto de vista da detecção de erros, pois caso dois bits (ou um número par de bits) cheguem errado ao receptor na seqüência em que é analisada a paridade, este código simplesmente não detecta o problema.

Para um bloco de bytes, a paridade pode ser feita de formas diferentes, sendo mais ou menos eficiente, dependendo do tipo de erro que acontece mais freqüentemente na linha (se erro simples, duplo, triplo, em rajada, etc). A seguir serão vistos dois tipos de paridade, a paridade horizontal e a paridade vertical.

HRC: Horizontal Redundancy Check

Neste tipo de código, o bit de paridade fica no final de cada linha transmitida, como mostra a figura a seguir.

b0	b1	b2	b3	b4	b5	b6	b7	P
0	1	1	0	1	1	0	1	1
1	0	0	0	1	0	0	0	0
1	1	0	0	1	1	1	1	0
0	1	0	1	1	0	0	1	0
1	1	0	1	1	1	0	0	1
1	0	1	1	0	1	0	1	1
0	1	0	1	1	0	1	1	1
1	1	1	0	1	1	0	0	1

VRC: Vertical Redundancy Check

Neste tipo de código, o bit de paridade fica no final de cada coluna transmitida, como mostra a figura a seguir. A vantagem deste método é que, caso aconteça um erro duplo ou em rajada, normalmente os bits errados estarão localizados na mesma linha, ou seja, em colunas diferentes, permitindo a detecção do erro.

	b0	b1	b2	b3	b4	b5	b6	b7
1	0	1	1	0	1	1	0	1
2	1	0	0	0	1	0	0	0
3	1	1	0	0	1	1	1	1
4	0	1	0	1	1	0	0	1
5	1	1	0	1	1	1	0	0
6	1	0	1	1	0	1	0	1
7	0	1	0	1	1	0	1	1
8	1	1	1	0	1	1	0	0
P	1	0	1	0	1	1	0	1

Exercício: Faça o VRC e o HRC para a string "Teste36". Use paridade "par".
OBS: se tiver VRC e HRC simultaneamente, posso detectar e corrigir o erro.

4.2 Checksum

O checksum pode ser definido como um byte que, somado à soma de todos os bytes transmitidos, torna o resultado da soma igual a zero.

Para o cálculo do checksum, o transmissor ao longo da transmissão faz a soma de todos os bytes de informação, armazenando este resultado em uma variável. O byte de checksum é calculado a partir desta variável, sendo um valor que, somado à ela, resulte em zero (complemento de 2). A figura a seguir mostra uma transmissão típica de um bloco, com o checksum sendo enviado no final deste bloco.

Bloco de Informações	Checksum

Supondo que se queira transmitir a seguinte informação (bytes de 4 bits): "1001", "0010", "1110", "0110", deve-se enviar através da linha a informação mais o byte de checksum, que é calculado da seguinte forma:

```
   B1 1 0 0 1 +
   B2 0 0 1 0 =
Soma1 1 0 1 1 +
   B3 1 1 1 0 =
Soma2 1 0 0 1 +
   B4 0 1 1 0 =
Soma3 1 1 1 1 (soma de todos bytes) +
      0 0 0 1 (checksum: complemento de dois)

      0 0 0 0 (resultado da soma de todos bytes
               mais checksum deve dar zero -
               conferido pelo receptor)
```

Do ponto de vista do receptor, ele deve receber todos os bytes, inclusive o de checksum, e somá-los. O resultado da soma deve dar zero. Se não der zero, significa que aconteceu algum erro na transmissão de dados e aquele bloco deve ser retransmitido.

Exercício: Faça o checksum para a string “Teste36”.

4.3 CRC (Cyclic Redundancy Code)

O CRC também é conhecido como **código polinomial**, pois as strings de bits são tratadas como representações de polinômios com coeficientes 0 e 1 somente. Assim, por exemplo, a string de bits "10011" seria encarada do ponto de vista de polinômios como $\mathbf{1*X^4+0*X^3+0*X^2+1*X^1+1*X^0}$, ou simplesmente $\mathbf{X^4+X+1}$.

Existe um **polinômio gerador G (x)** do CRC, que pode variar dependendo do protocolo utilizado. O mais comum é o CRC-16, que é o seguinte: $X^{16}+X^{15}+X^2+1$, entretanto, existem diversos outros, como o CRC-32, utilizado nas redes locais Ethernet, Token Ring e Token Bus. Tanto o transmissor como o receptor devem utilizar o mesmo polinômio gerador, para que cheguem ao mesmo valor de CRC.

Em termos de detecção de erros este sistema (CRC-16) é bastante eficiente. Para um bloco de 32Kbytes, o CRC-16 Permite detectar todos erros simples, duplos, triplos e de número ímpar no bloco. Além disto, pode detectar todos erros em rajada de comprimento 16 ou menos, 99,997 % dos erros em rajada de 17 bits e 99,998 % dos erros em rajada de 18 bits ou mais.

O cálculo que o transmissor deve fazer para obter o CRC é o seguinte: considerando que M (x) é a string de bits que devem ser transmitidos, e G (x) o polinômio gerador, e r o grau do polinômio gerador, **então o CRC é o resto da divisão de X^r * M (x) / G (x).** Este valor deverá ser transmitido no final da mensagem, tal como o checksum.

Exemplo: Usando aritmética polinomial módulo-2, encontre o CRC a ser adicionado na mensagem M=11100110, sendo que o polinômio gerador $G(x) = X^4+X^3+1$.

G(x) é um polinômio de grau 4, portanto, r = 4

Escrevendo a mensagem na forma polinomial. $M(x) = X^7+X^6+ X^5+X^2+X$

Efetua-se a fórmula: $X^r*M(x)/G(x) = (X^{11}+X^{10}+X^9+X^6+X^5) / (X^4+X^3+1)$. OBS: em aritmética polinomial módulo-2, $X^8+ X^8=0$, e $X^8 – X^8=0$

O resultado é: $R(x) = X^2+X$, que dá um CRC = 0110

O pacote a ser transmitido é: T(x) = 111001100110

A fim de o receptor testar erros, ele deve fazer T(x) / G(x), e o resultado tem que ser zero para indicar transmissão sem erros. $T(x)= X^{11}+X^{10}+ X^9+X^6 +X^5+X^2+X$

Para esse exemplo, todos erros até 4 bits podem ser detectados, resultando num R(x) diferente de zero no receptor. Teste em casa...

OBS: para mais bytes em M(x), basta **pegar sempre um número de bits igual à ordem de G(x)**, somar o resto parcial obtido com o novo byte da mensagem (equivalente a um “ou exclusivo”) e fazer o processo novamente.

Por exemplo, para M(x)=11100110 01100101 00001001 e $G(x)= X^8+X^7 +1$, uma forma de fazer é calcular o resto de:

$(X^{31}+X^{30}+ X^{29}+X^{26} +X^{25}+X^{22}+ X^{21}+X^{18}+ X^{16}+X^{11} +X^8) / (X^8+X^7+1)$

que é igual a: $X^7+ X^5+X^4 +X^3+X^2$, resultando um CRC = 10111100 **(1)**

O mesmo resultado pode ser obtido pegando o resultado do primeiro byte (11100110) R1(x) = resto de: $(X^{15}+X^{14}+ X^{13}+X^{10} +X^9) / (X^8+X^7+1) = X^5+X^4+ X^3+X+1$ = (00111011). Somando (ou fazendo “ou exclusivo” com o segundo byte tem-se: (00111011) XOR (01100101) = (01011110) = $X^6+X^4+X^3+X^2+X$.

Aplicando-se novamente o polinômio gerador tem-se R2(x) = resto de: $(X^{14}+X^{12}+ X^{11}+X^{10}+X^{9}) / (X^{8}+X^{7}+1)$ = $X^{7}+X^{6}+X^{5}+X^{3}+X+1$ = (11101011). Somando (ou fazendo "ou exclusivo" com o terceiro byte tem-se: (11101011) XOR (00001001) = (11100010) = $X^{7}+X^{6}+ X^{5}+X$.

Aplicando-se novamente o polinômio gerador tem-se R3(x) = resto de: $(X^{15}+X^{14}+ X^{13}+X^{9}) / (X^{8}+X^{7}+1)$ = $X^{7}+X^{5}+X^{4}+X^{3}+X^{2}$ = (10111100), que é o CRC do sistema, conforme obtido no cálculo completo visto em (1).

Exercício: fazer CRC pelos dois métodos tendo M(x)=1100 1010 0011 e G(x)=$X^{4}+X^{3}+1$.

A figura 3 apresenta uma implementação em *hardware* para obtenção do FCS na transmissão. Na recepção, será usado um circuito idêntico para a verificação do resto da divisão de *T(x)* por *G(x)*.

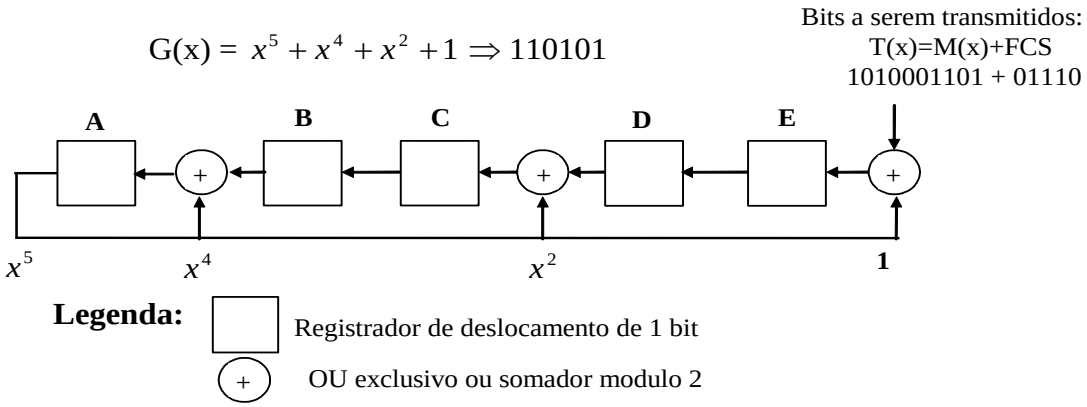


Fig. 3 - Implementação de um circuito com registradores de deslocamento para fazer a divisão pelo polinômio $x^5 + x^4 + x^2 + 1$ e obter o FCS

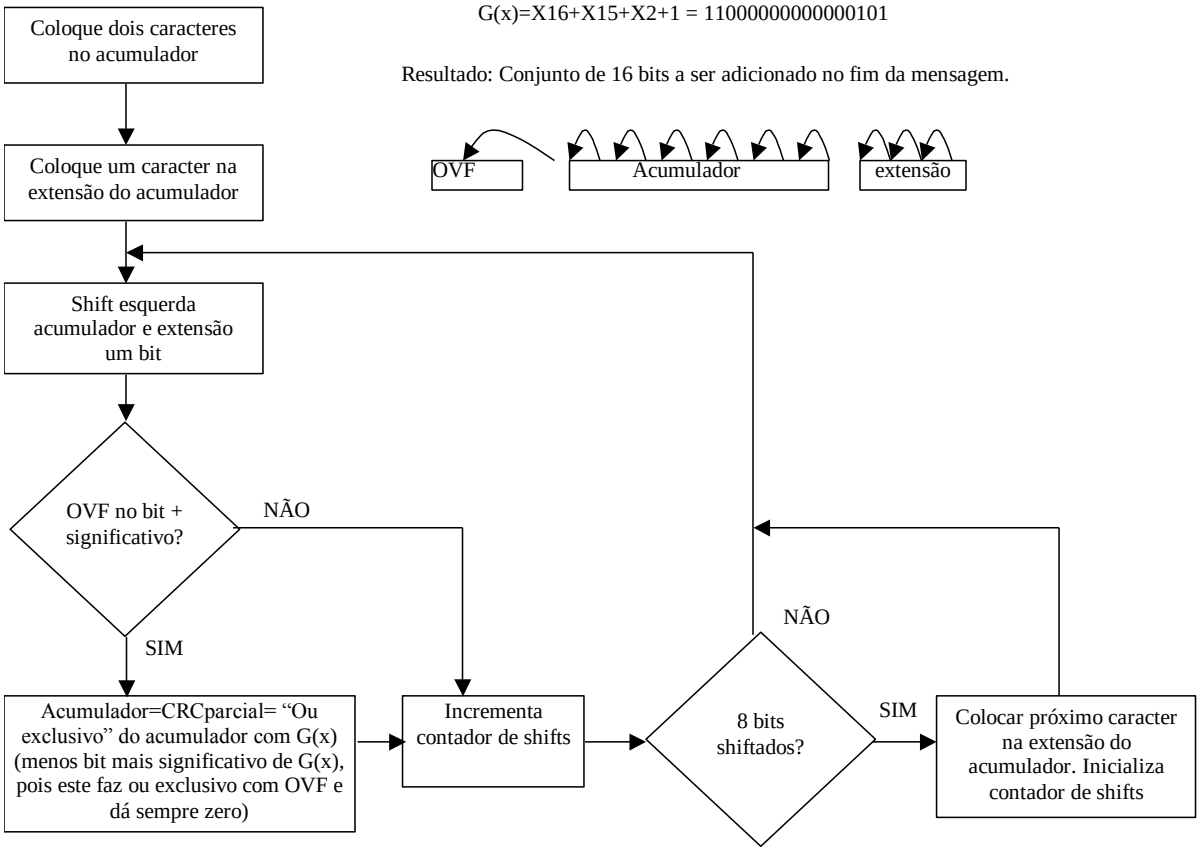
Tab. 2 - Estado dos registradores por pulso de relógio

Registrador	A	B	C	D	E	Bits de Entrada	
Condição Inicial	0	0	0	0	0		
Etapa 1	0	0	0	0	1	**1**	
Etapa 2	0	0	0	1	0	**0**	
Etapa 3	0	0	1	0	1	**1**	**Mensagem**
Etapa 4	0	1	0	1	0	**0**	**M(x) a**
Etapa 5	1	0	1	0	0	**0**	**ser**
Etapa 6	1	1	1	0	1	**0**	**enviada**
Etapa 7	0	1	1	1	0	**1**	**(10 bits)**
Etapa 8	1	1	1	0	1	**1**	
Etapa 9	0	1	1	1	1	**0**	
Etapa 10	1	1	1	1	1	**1**	
Etapa 11	0	1	0	1	1	0	
Etapa 12	1	0	1	1	0	0	**5 bits zero**
Etapa 13	1	1	0	0	1	0	**acrescentados**
Etapa 14	0	0	1	1	1	0	
Etapa 15	**0**	**1**	**1**	**1**	**0**	0	

⇑ **Resto da divisão ou FCS**

Transmissor envia *T(x)* = *M(x)* + FCS = 1010001101 01110
Se no receptor a divisão por *G(x)* resultar em resto igual a zero, significa que não ocorreu erro na transmissão.

Outro mecanismo similar de geração do CRC via computador pode ser visto pelo fluxograma a seguir.



Eficiência do Método CRC

A eficiência de um método de detecção de erros fornece a percentagem de erros que são detectados pelo método. Nenhum método detecta 100% dos erros ocorridos em uma transmissão. Os erros que passam despercebidos constituem o que é chamado de taxa residual de erro do sistema de transmissão.

O método CRC é hoje o mais largamente utilizado em sistemas de transmissão, por dois motivos: sua simplicidade de implementação, normalmente através de um *hardware* muito simples (vide figura 3), e sua alta eficiência.

Supondo um polinômio gerador dado por *G(x)*, a eficiência do CRC pode ser avaliada pelas seguintes constatações.

1. São detectados todos os erros de um bit.
2. São detectados todos os erros duplos desde que *G(x)* tenha pelo menos três termos.
3. São detectados todos os erros de bit, em número impar, desde que *G(x)* tenha um fator + 1 (por isso todos os polinômios padronizados apresentam este termo).
4. São detectados todos os erros de rajada desde que o comprimento da rajada seja menor que o comprimento do polinômio *G(x)*, isto é, seja menor ou igual ao comprimento do campo FCS.
5. A maioria dos erros de rajada maior que *G(x)*.

5 CÓDIGOS DE CORREÇÃO AUTOMÁTICA DO ERRO

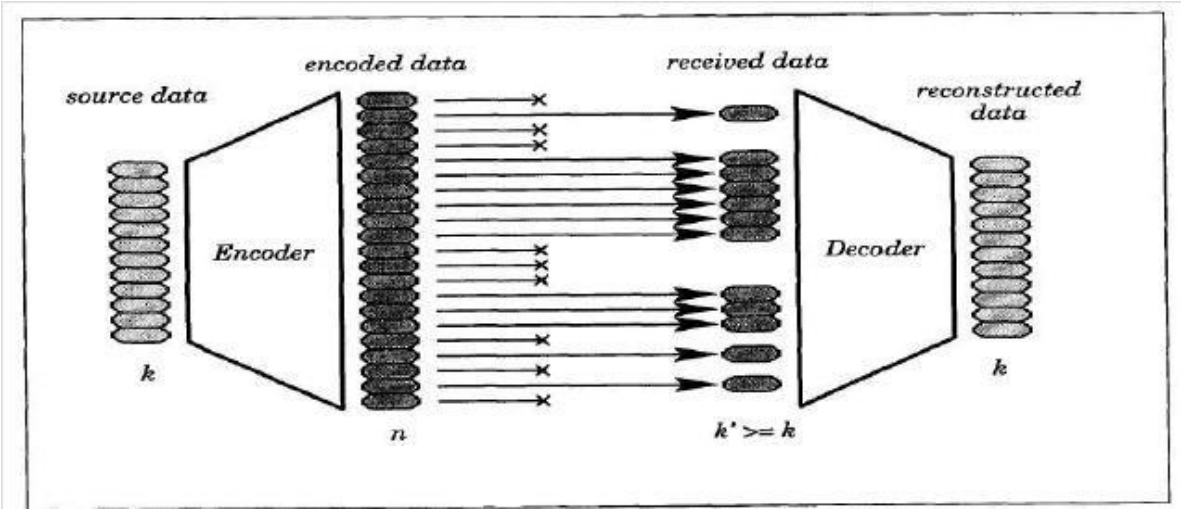
Neste tipo de código, são enviados caracteres redundantes suficientes para descobrir se deu erro na transmissão e a posição onde ocorreu o erro, permitindo assim uma correção automática, sem a necessidade de retransmissão de blocos.

Tipos:

Convolucionais – Operam sobre bits ou streams de símbolos de tamanho arbitrário. Ex: Viterbi

Códigos de Bloco – Operam sobre blocos de tamanho fixo pré-determinado. Ex: Reed-Solomon, BCH, Golay, Hamming

A figura a seguir ilustra um exemplo ideal de FEC, onde existem k dados origem que são codificados em n dados. Se o receptor conseguir recuperar até k dados dos n transmitidos, é possível reconstruir os dados originais.



5.1 BITS DE HAMMING

Supondo o seguinte byte de informações que deve ser transmitido para o receptor: "01010100".

Um exemplo de código de Hamming prevê a criação de bits redundantes, inserindo-os em determinadas posições do byte, que são 2^0, 2^1, 2^2, 2^3, 2^4, e assim por diante, resultando nas posições 1, 2, 4, 8, 16, 32, etc. Desta forma, os bits a serem transmitidos ficam conforme mostra a figura a seguir.

...	12	11	10	9	8	7	6	5	4	3	2	1
X	0	1	0	1	H4	0	1	0	H3	0	H2	H1

A segunda etapa é descobrir os valores de "H", ou seja, se são bits "0" ou bits "1". Para isto, deve-se fazer um **ou-exclusivo** entre todas as posições que contenham bits "1" no byte acima, resultando nos bits de Hamming, como mostra o seguinte desenvolvimento:

```
 6: 0  1  1  0 XOR
 9: 1  0  0  1
R1  1  1  1  1 XOR
11: 1  0  1  1 =
R2  0  1  0  0 - Bits de hamming
    H4 H3 H2 H1
```

Com os bits de Hamming calculados, basta inseri-los nas posições reservadas, obtendo-se o pacote que deverá ser transmitido para o receptor, mostrado a seguir.

...	12	11	10	9	8	7	6	5	4	3	2	1

X	0	1	0	1	0	0	1	0	1	0	0	0

O receptor, por sua vez, deve fazer o mesmo processo, ou seja, calcular o ou-exclusivo para todas as posições que possuam bits "1" do pacote recebido. O resultado deve dar zero. Se o resultado não der zero, vai informar a posição onde ocorreu o erro.

Supondo que deu problema na transmissão e o bit referente à posição 11 foi recebido errado (era para ser "1" e foi recebido "0"), como mostra a figura a seguir.

...	12	11	10	9	8	7	6	5	4	3	2	1
X	0	0	0	1	0	0	1	0	1	0	0	0

O receptor ainda não sabe da existência do erro, portanto, vai fazer o ou-exclusivo de todas as posições com bits "1" menos a posição 11, pois ele recebeu como "0" e não sabe que deveria ser "1". O cálculo é mostrado a seguir.

```
 4: 0  1  0  0 XOR
 6: 0  1  1  0
R1  0  0  1  0 XOR
 9: 1  0  0  1 =
R2  1  0  1  1 - Indica erro na posição 11
```

O resultado não foi zero, logo, o receptor sabe que deu erro. A posição do bit errado é exatamente o resultado alcançado, ou seja, 11. Desta forma, o receptor automaticamente vai na posição 11 do pacote recebido e inverte o bit existente nesta posição, corrigindo a informação recebida.

Este tipo de código é mais utilizado para transmissão de dados quando o canal é simplex, pois não é possível solicitar retransmissões. De outra forma, os códigos de detecção e retransmissão são mais utilizados, pelos seguintes fatores:

- O número de bits redundantes nos códigos de autocorreção é maior, provocando uma necessidade de transmitir mais bits para a mesma informação;
- Os códigos de autocorreção possuem problemas quando acontecem erros em rajadas, ou mesmo erros duplos, não sendo tão confiáveis quanto os de detecção.

Exercício: Calcular os bits de Hamming para a seqüência “100011010001”. Testar a recepção caso tenha virado o bit 12. O que o resultado indica?

Hamming estabeleceu em 1950 as principais bases teóricas da teoria do erro, ou seja, a sua detecção e correção. Vamos supor palavras de código formadas por n bits, sendo que destes m são bits de dados e r são bits redundantes ou de código. Portanto, $n = m + r$ forma uma palavra de código de n bits que será utilizada na troca de informação. Duas palavras de código de n bits diferem em várias posições de bit, que podem ser obtidas através de uma soma módulo 2 (“ou” exclusivo).

Ex: 10001001 e 10110001
Soma módulo 2 será:

```
  10001001
+ 10110001
  00111000
```

As duas palavras diferem em três posições de bit (onde o resultado foi 1).

O número de posições de bit em que duas palavras de código diferem é chamado de *Distância de Hamming* (DH). Duas palavras de código com DH igual à *d* requerem *d* erros de bit simples para converter uma em outra. As 2^m combinações possíveis dos *m* bits de dados são legais, mas, dependendo de como são calculados os bits redundantes, nem todas as combinações, $2^n = 2^m.2^r = 2^{m+r}$, são possíveis.

Dado o algoritmo para calcular os bits redundantes, é possível construir a lista completa das palavras de código possíveis e, desta lista, achar as duas palavras código cuja distância de Hamming (DH) é mínima. Esta distância é a distância de Hamming do código completo.

As propriedades de *detecção de erros* e *correção de erros* dependem desta distância (DH), e podem ser enunciadas da seguinte forma:

1. Para detectar *d* erros em uma palavra de código é necessário um código com uma DH de *d+1* bits.
   Prova: A ocorrência de *d* erros simples não consegue mudar uma palavra de código válida em uma outra palavra de código válida.

2. Para corrigir *e* erros é necessário um código com uma DH de *2e+1* bits.
   Prova: As palavras de código válidas estarão tão longe, que mesmo com *e* erros simples, a palavra de código original ainda está mais próxima que qualquer outra palavra de código e, desta forma, pode ser determinada de forma unívoca.

Ex: Consideremos um código com apenas 4 palavras de código válidas:
0000000000, 0000011111, 1111100000 e 1111111111.

Este código, como se pode observar, tem uma DH igual a 5 (pois é o menor número de bits errados que pode transformar uma palavra na outra), o que significa que:

a) detecta até 4 erros, pois DH = d + 1 = 5, ou seja, d = 4;
b) corrige até 2 erros, pois DH = 2e + 1 = 5, ou seja, e = 2.

Supondo que queremos projetar um código com *m* bits de palavra código e *r* bits de redundância e que seja capaz de corrigir todos os erros simples. Cada uma das 2^m palavras de código legais possui *n* palavras código ilegais a uma distância 1 dela. Estas palavras de código são formadas simplesmente invertendo cada um dos *n* bits das *n* palavras código formadas a partir dela. Desta forma, cada uma das 2^m palavras de código legais requer *n +1* padrões de bit associados a ele. Como o número total de padrões de bit é igual a 2^n, nós devemos ter *(n + 1)* $2^m \leq 2^n$. Como assumimos que *n = m + r*, esta condição se torna *(m + r + 1)* $\leq 2^r$. Portanto, dado *m* temos como definir um limite inferior para o número de bits redundantes para corrigir um erro.

5.2 Codificação Reed Solomon

Utilizada em CDs, DVDs, discos rígidos, HDTV, cable modems e transmissões espaciais. Ideal para aplicações em que ocorrem erros em rajadas.

Seu princípio de funcionamento é baseado no teorema de álgebra linear em que dados k pontos, é possível determinar um polinômio único de grau k-1.

A codificação consiste em determinar o polinômio de grau k-1 que passa pelos k pontos. O polinômio então é avaliado em vários pontos (mais do que k), que são os dados que serão transmitidos. O decodificador consegue

recuperar a mensagem desde que receba pontos suficientes para deduzir qual era o polinômio original. Suas propriedades são:

Identificado por (n, k)

k = número de símbolos de entrada

n = número de símbolos após a codificação

É capaz de corrigir (n-k)/2 símbolos errados. Múltiplos bits errados em um símbolo contam como um erro.

Padrão (255, 223) é normalmente utilizado

Referências:

Error-correcting code

http://en.wikipedia.org/wiki/Error-correcting_code

http://www.aero.org/publications/crosslink/winter2002/04.html

http://www.aero.org/publications/crosslink/winter2002/04_sidebar1.html

Reed Solomon

http://en.wikipedia.org/wiki/Reed-Solomon_error_correction

http://sidewords.files.wordpress.com/2007/12/thesis.pdf (página 48)

Robert H. Morelos-Zaragoza: "The Art of error Correcting Coding", Capítulo 4 – (análise matemática e algoritmo Reed-Solomon)

6 ATIVIDADES

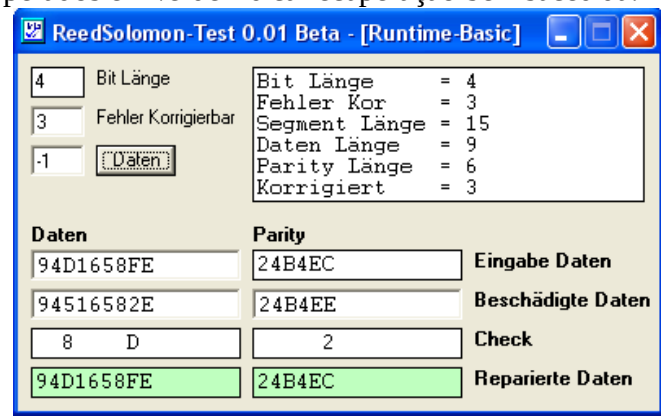
6.1 Exercícios

1. **Utilizando o polinômio CRC-CCITT (x16 + x12 + x5 + 1), gere um código CRC de 16 bits para uma mensagem formada por um bit 1 seguido de quinze bits 0.**
2. **Considere o código Reed Solomon utilizado em TV Digital, com o identificador (204,188), com 16 bytes de redundância. Para cada bloco, é possível corrigir 8 bytes errados. Descreva com detalhes uma forma de aumentar o número de bytes corrigidos para 80 bytes errados num pacote, sem aumentar a redundância de cada quadro individual.**

6.2 Experiências

1. **O simulador ReedSolomon-Test permite verificar o funcionamento da correção de erros com o Reed-Solomon. Está em alemão, mas o significado é:**
   - Bit Länge – Número de bits por símbolo (fixo)
   - Fehler Korrigierbar – Número de símbolos máximo que o código corrige
   - Botão Daten – Gera dados aleatórios se campo ao lado possuir valores negativos.
   - Eingabe Daten – Dados de entrada

- Beschädigte Daten – Dados corrompidos
- Check – Diferença entre dados de entrada e dados corrompidos
- Reparierte Daten – Dados Recuperados
- Campos que não estão circulados podem ser editados
- Ao editar um campo pressione a tecla Enter para atualizar os dados
- Campo de dados recuperados em verde indica recuperação bem sucedida.



Baixe o simulador em:

http://runtimebasic.net/_media/Projekt:ReedSolomon.zip?id=Projekt%3ADownload&cache=nocache e verifique seu funcionamento editando o campo de dados corrompidos, sua paridade e o valor do campo *Fehler Korrigierbar*. Questões:

a) O que determina o número no campo *Fehler Korrigierbar* ?
b) Qual é o número máximo de **bits** (eficiência do algoritmo) que podem ser corrigidos com *Fehler Korrigierbar = 3?* a) no melhor caso de erros na linha; b) no pior caso de erros. A ideia da questão é reforçar que o Reed Solomon trabalha com símbolos, e não com bits.
c) No caso dos CDs, erros em rajadas maiores que a capacidade de correção de cada bloco (*frame*) são comuns. Que técnica poderíamos utilizar para evitar a perda de dados com rajadas grandes sem ter que aumentar o número de símbolos de redundância da codificação? Explique.

2. Baixe o simulador da página da disciplina (só funciona em 32 bits), leia o help e efetue os testes indicados, utilizando os arquivos como base (eles estão localizados no diretório de instalação do programa). Execute passo a passo os comandos mostrados no help. Edite o arquivo manualmente e veja se ele consegue corrigir o número de erros prometido. O relatório deve conter o resultado obtido passo a passo, aumentando o número de símbolos errados até o sistema não recuperar mais os erros. Justifique. Os passos a serem efetuados são:

a) RS Input: configurar os parâmetros do Reed Solomon (default – RS (255,249) símbolos de 8 bits).
b) Encode: Entrar o arquivo de dados para codificação
c) Insert Errors: Inserir erros no arquivo, de preferência de forma manual (editando o arquivo codificado e gerando um ".err" com algum editor de texto).
d) Decode: Selecionar o arquivo codificado (já com os erros – ".err") para decodificação.
e) Compare: Comparar o arquivo original e o decodificado, vendo se os erros foram corrigidos.