Serviços da Camada de Enlace, Correção de Erros

Diego Passos

23 de Agosto de 2017

1 Introdução

Assim como no estudo das demais camadas da pilha de protocolos TCP/IP, a camada de enlace possui uma terminologia própria. Enquanto no estudo da camada de rede fazíamos a distinção entre roteadores e *hosts*, na camada de enlace não haverá essa diferenciação e nos referiremos a ambos genericamente com **nós**. Dois ou mais nós são ditos **adjacentes** se estes são conectados diretamente através de um **enlace** — ou ***link***, em inglês. Por fim, a unidade de transmissão (*i.e.*, o pacote) na camada de enlace é chamada de **quadro** — ou ***frame***, em inglês. Ao longo desta e das próximas aulas outros termos serão gradativamente introduzidos.

Como o próprio nome sugere, o enlace é um conceito central a esta camada. De fato, pode-se resumir a responsabilidade da camada de enlace na seguinte frase: **é a camada responsável pela transferência de pacotes entre nós conectados fisicamente por um enlace**. Enquanto a camada de rede se preocupa em determinar e encaminhar pacotes através de caminhos potencialmente compostos por múltiplos enlaces, a responsabilidade da camada de enlace começa e termina na transferência do pacote por um único enlace físico. Em outras palavras, a camada de rede determina o próximo enlace a ser utilizado em um caminho e a camada de enlace se encarrega de fazer com que o pacote chegue ao outro extremo do enlace — ou ao menos tentar.

Uma característica marcante da camada de enlace na pilha de protocolos TCP/IP é a grande variabilidade de tecnologias utilizadas. Enquanto na camada de rede da Internet o protocolo IP é — e precisa ser — ubíquo, na camada de enlace várias tecnologias são encontradas. Alguns exemplos tradicionais incluem o Ethernet, o Wi-Fi, o ADSL e o *Frame Relay*. Cada um desses protocolos tem suas características particulares — e são, de fato, muito diferentes uns dos outros. Eles variam, inclusive, em termos do meio físico utilizado para a transmissão de dados. Por exemplo, o Wi-Fi é uma tecnologia que utiliza enlaces sem fio, enquanto o Ethernet utiliza meios de transmissão cabeados. Esses protocolos também podem implementar conjuntos de serviços diferentes, de acordo com seus objetivos e com as características do meio físico utilizado.

2 Serviços da Camada de Enlace

Embora os conjuntos de serviços oferecidos por protocolos diferentes da camada de enlace variem, alguns serviços são típicos. Aqui, veremos uma breve lista (incompleta) desses serviços.

Um serviço bastante comum nessa camada é o de **encapsulamento e desencapsulamento**. Esse serviço, existente nas demais camadas da pilha TCP/IP estudadas até aqui, é análogo ao que ocorre nas camadas superiores. Isto é, ele consiste da adição (antes da transmissão) ou remoção (após a recepção) de informações de controle do protocolo do pacote em questão. Assim como nas camadas superiores, essas informações de controle são geralmente representadas na forma de um **cabeçalho**. No entanto, protocolos da camada de enlace muitas vezes empregam também um ***trailer***: bits de controle adicionados **ao final do pacote**. Em aulas posteriores, quando estudarmos o funcionamento do Ethernet, veremos um exemplo concreto e discutiremos a utilidade disso.

Outro serviço comumente encontrado na camada de enlace é o de **acesso ao meio**. Conforme estudaremos em aulas posteriores em maiores detalhes, enlaces de comunicação podem ser divididos em **enlaces dedicados** e **enlaces compartilhados**. Esse termo *compartilhado* significa que o mesmo meio físico é utilizado por múltiplos nós para a transmissão de seus pacotes. Esse compartilhamento, no entanto, não é trivial já que, normalmente, dois ou mais nós não podem transmitir **ao mesmo tempo pelo enlace**. Logo, é necessário algum nível de coordenação entre os vários nós que competem pela chance de utilizar o enlace. Essa coordenação é de extrema importância e tem influência direta na eficiência e desempenho do enlace de comunicação.

O **endereçamento** é outro serviço bastante comum em protocolos da camada de enlace. Em geral, no cabeçalho de um quadro constarão, ao menos, os endereços do nó transmissor e do nó receptor

daquele quadro — em certas tecnologias, como o Wi-Fi, pode haver outros endereços pertinentes a um quadro. Isso é particularmente importante para enlaces compartilhados, já que múltiplos transmissores — e receptores — podem estar conectados ao mesmo enlace, justificando a necessidade dos endereços constarem no cabeçalho. É importante destacar que **os endereços utilizados na camada de enlace são diferentes dos endereços da camada de rede**. Isto é, ainda que a camada de rede atribua endereços IP para as interfaces de um nó, a camada de enlace tipicamente terá seus próprios endereços atribuídos para as interfaces. Esses endereços da camada de enlace são tipicamente chamados de **endereços MAC**[1].

Não tão comum, mas ainda presente em vários protocolos de camada de enlace, o serviço de **entrega confiável de dados** é similar ao serviço prestado pelo protocolo TCP na camada de transporte. A ideia é a utilização de técnicas e mecanismos para detectar a perda de quadros transmitidos pelo enlace e sua posterior recuperação através de retransmissões. Uma diferença bastante clara entre a entrega confiável de dados usada pelo TCP e aquela usada na camada de enlace está no nível de persistência dos protocolos: enquanto no TCP não há *a priori* um limite no número de vezes que um segmento é retransmitido, na camada de enlace tipicamente impõe-se uma limitação a esse número. Em outras palavras, mesmo que se use um mecanismo de entrega confiável de dados, o protocolo da camada de enlace pode simplesmente desistir da entrega de um certo quadro após um determinado número de tentativas de transmissão frustradas.

Mas qual seria a razão para que um protocolo desista de retransmitir um determinado quadro? Considere, por exemplo, um enlace compartilhado por múltiplos nós. Suponha que um transmissor esteja tentando transmitir um quadro para um nó que não está atualmente conectado ao enlace. Como o receptor não está conectado, a transmissão nunca será bem-sucedida. Se o transmissor continuar tentando realizar a transmissão indefinidamente, ele estará apenas desperdiçando recursos do enlace sem produzir um efeito prático positivo. Além disso, é comum assumirmos que, se o conteúdo daquele quadro for realmente importante, alguma camada superior se encarregará de detectar a perda e realizar a retransmissão fim-a-fim.

Esse último argumento, na verdade, indica que, a rigor, a implementação da entrega confiável de dados não é essencial na camada de enlace. Poderia-se, inclusive, dizer que ela é redundante. Por que motivo, então, certos protocolos dessa camada a implementam? A resposta para isso está nas características dos enlaces de comunicação. Alguns enlaces são extremamente susceptíveis a falhas nas transmissões. Nesses, perdas de quadros são ocorrências comuns. Embora deixar que uma camada superior se encarregue da detecção da perda e retransmissão fim-a-fim funcione, esse método passa a ser bastante ineficiente quando os níveis de perda de quadros no enlace aumentam. Isso porque os tempos envolvidos na detecção de uma perda de pacote nas camadas superiores são normalmente bem maiores que os necessários para que a própria camada de enlace detecte e se recupere de uma perda de quadros. Assim, para enlaces com altos índices de perda de quadros, a existência de um mecanismo de entrega confiável de dados que realize ao menos algumas tentativas de retransmissão pode levar a um ganho expressivo de desempenho, retirando parte do ônus das camadas superiores.

A **detecção de erros** também é um serviço bastante comum. Assim como em camadas superiores, a detecção de erro consiste na verificação da consistência do quadro pelo receptor. Analogamente à transmissão confiável de dados, esse serviço é particularmente importante para tecnologias que lidam com meios de transmissão muito susceptíveis a falhas. Nesses meios, o quadro pode ser recebido, porém com bits com valor corrompido. Através da adição de bits redundantes, métodos de detecção de erro permitem muitas vezes que o receptor detecte a ocorrência dessas corrupções e tome alguma atitude.

Essa atitude em face de corrupções no pacote pode ser o simples descarte do pacote ou o uso de algum mecanismo de **correção de erros**. O serviço de correção de erros é muito similar ao de detecção de erros: o transmissor insere bits redundantes no quadro de forma que, em caso de corrupção, o receptor *possa talvez* restaurar o conteúdo original do pacote. Note que estamos falando aqui de uma operação envolvendo apenas os bits do quadro recebido, sem qualquer tipo de retransmissão por parte do transmissor.

Pode haver ainda o serviço de **controle de fluxo**. Assim como o controle de fluxo do TCP, esse serviço objetiva evitar que o transmissor "afogue" o receptor, *i.e.*, que ele transmita mais rapidamente que o transmissor é capaz de receber. No entanto, por se tratar de um serviço no contexto da camada de enlace, estamos nos referindo às transmissões de quadros por um enlace, entre dois nós adjacentes. Geralmente, isso é necessário quando as interfaces de rede possuem *buffers* de recepção muito limitados.

Por fim, há o serviço de transmissão do quadro pelo enlace físico. Esse serviço obviamente precisa existir em qualquer protocolo de camada de enlace, mas há algumas pequenas variações de tecnologia

[1] Essa sigla MAC vem do inglês *Media Access Control*, um termo usado para denotar um subconjunto de funcionalidades da camada de enlace. Muitas vezes, o termo MAC é usado de maneira informal e imprecisa para denotar a própria camada de enlace, como em *a camada MAC*.

para tecnologia. Um enlace é dito ***full-duplex*** quando permite transmissões entre dois nós **nos dois sentidos simultaneamente**. Um enlace é dito ***half-duplex*** quando permite transmissões entre dois nós **nos dois sentidos, porém não ao mesmo tempo**. Um enlace pode ainda ser ***half-duplex***, o que significa que ele suporta apenas transmissões em um sentido.

3 Implementação da Camada de Enlace

No modelo TCP/IP, a camada de enlace é mandatória em todos os nós da rede, englobando, portanto, tanto *hosts*, quanto comutadores. Historicamente, ao contrário do que ocorre com a maioria dos componentes das camadas superiores, a camada de enlace tem sido majoritariamente implementada diretamente pela interface de rede — *i.e.*, a placa de rede ou NIC (do inglês *Network Interface Controller*). Vários serviços dessa camada são bastante sensíveis a tempo — em particular, o serviço de acesso ao meio comumente necessita da execução de determinadas ações em tempos bem precisos. Isso faz com que a implementação dessa camada em um *hardware* dedicado exclusivamente a isso seja uma abordagem popular. Nessa arquitetura, a interface de rede se comunicaria com o resto do sistema através de um barramento de entrada e saída, recebendo datagramas a serem enviados e repassando pacotes recém-recebidos e já completamente processados para a CPU/memória principal.

Nas últimas décadas, no entanto, tem havido uma migração de funcionalidades de *hardware* para *software* — não só na área de redes e comunicações de dados, mas em computação em geral. Por sua facilidade de desenvolvimento, manutenção e extensão, soluções baseadas em *software* se tornam vantajosas comercialmente. Do ponto de vista da camada de enlace, isso significou um processo de migração das funcionalidades possíveis de circuitos eletrônicos dedicados para *software*. Inicialmente, isso se deu quase exclusivamente através da adoção de *firmwares* — *i.e.*, um *software* executado por um processador na própria interface de rede —, mas hoje é comum uma arquitetura em que um *driver* executando no âmbito do próprio sistema operacional acumule diversas funções da camada de enlace.

4 Detecção e Correção de Erros

No restante dessa aula, discutiremos alguns dos principais mecanismos de detecção e correção de erros utilizados por protocolos da camada de enlace. Ambos os processos, de detecção e de correção de erros, têm a mesmo funcionamento básico:

- O transmissor insere no quadro um conjunto de **bits de redundância**. Esses bits são computados deterministicamente a partir dos bits do quadro original.
- O quadro, contendo agora os bits de redundância, é transmitido pelo enlace.
- Os bits do pacote são recebidos pelo receptor, mas alguns podem ter sido corrompidos no enlace. **Note que a corrupção pode, inclusive, atingir os bits de redundância**.
- O receptor efetua algum tipo de computação determinística sobre os bits do quadro e de redundância. O resultado dessa computação indica **probabilisticamente** se o quadro está íntegro (*i.e.*, sem corrupções).

Esse último ponto é muito importante: **nenhum método de detecção de erros é 100% confiável**. Ainda que o método determine que quadro não sofre corrupções, **é possível que tenha havido alguma corrupção não detectada**. Uma das maneiras de avaliar a qualidade de um método de detecção de erros é justamente a sua probabilidade de não detectar corrupções em pacotes.

No caso de um método de correção de erros, também chamado de FEC (do inglês *Forward Error Correction*), há uma etapa adicional que seria a correção do quadro em caso de corrupção detectada. Essa "correção" nada mais é que uma tentativa de fazer com que haja consistência entre os bits do quadro original e os bits de redundância, de forma que o novo conjunto de bits obtido passe no teste de detecção de erros. No entanto, assim como a simples detecção de erros não é 100% confiável, a correção de erros nem sempre funciona, podendo-se obter ao final do processamento um quadro ainda diferente do originalmente transmitido.

De modo geral, quanto maior o número de bits de redundância utilizados pelo método de detecção/correção de erros, maior a probabilidade de que os erros sejam detectados/corrigidos. Há, entretanto, métodos mais eficientes que outros na detecção de certos erros típicos introduzidos por enlaces de comunicação. Em outras palavras, dois métodos diferentes que utilizem a mesma quantidade de bits de redundância podem ter eficiências distintas na detecção/correção de erros comuns em certos tipos de enlaces.

4.1 Bits de Paridade

Como primeiro exemplo de método de detecção de erros, podemos citar a **paridade simples**. Nesse método, um único bit de redundância — também chamado de **bit de paridade** — é adicionado. Os bits do quadro são somados e, a depender do resultado ser par ou ímpar, utiliza-se o bit de redundância como 0 ou 1.

A paridade simples pode ser implementada usando-se duas convenções: **a paridade par** ou **a paridade ímpar**. Na paridade par, se a soma dos bits do quadro original for par, o bit de paridade deve ser 0; caso contrário, o bit de paridade deve ser 1. Na paridade ímpar, inverte-se a lógica: se a soma dos bits do quadro original for ímpar, o bit de paridade deve ser 0; caso contrário, o bit de paridade deve ser 1.

Uma outra forma de pensar nessas convenções — talvez mais intuitiva — é a seguinte. Na paridade par, deseja-se que a soma dos bits do quadro, **incluindo o bit de redundância**, seja par. Na paridade ímpar, deseja-se que a soma dos bits do quadro, **incluindo o bit de redundância**, seja ímpar. Como a soma dos bits do quadro original — *i.e.*, sem o bit de redundância — pode ser tanto par, quanto ímpar, o valor do bit de redundância é manipulado de forma a manter a paridade do quadro conforme a convenção.

A paridade simples sempre consegue detectar a existência de erros no quadro desde que um número ímpar de bits tenham sido corrompidos. Se o número de bits corrompidos for par, a paridade do quadro como um todo não muda e o método falsamente acredita que a mensagem continua íntegra. Repare, ainda, que a paridade simples não pode ser usada para inferir quais bits foram alterados, não sendo, portanto, útil para a correção do quadro.

Um método similar que permite também a correção do quadro é a chamada **paridade bidimensional**. Nesse método, o transmissor dispõe os bits do quadro original na forma de uma matriz $n \times m$ — n e m são parâmetros do método, conhecidos tanto pelo transmissor, quanto pelo receptor. Para cada linha da matriz, calcula-se um bit de paridade. Faz-se o mesmo para cada coluna. O transmissor, então, transmite os bits do quadro original, acrescido dos bits de paridade das linhas e das colunas. Ao receber essa sequência de bits, o receptor novamente dispõe os bits do quadro no formato matricial, recalcula as paridades das linhas e colunas e verifica se os bits de paridade recebidos batem com os calculados. Se sim, assume-se que o quadro está correto. Caso contrário, entende-se que há algum erro.

Ao adicionar mais bits de redundância — $n + m$ no total — a paridade bidimensional aumenta a probabilidade de detecção de erros. Mas o grande benefício desse método é sua capacidade de correção. Suponha que o receptor receba um quadro em que exatamente um bit foi corrompido. Digamos que esse bit corresponda à posição (i, j) da matriz. Como esse bit da linha i teve seu valor alterado — e apenas ele — então a paridade da linha i calculada pelo receptor será diferente do bit de paridade correspondente recebido junto ao quadro. Da mesma forma, a paridade calculada pelo receptor para a coluna j não corresponderá ao bit de paridade correspondente à coluna j recebido com o quadro. Assim, dado que as paridades da linha i e da coluna j falharam, o receptor pode inferir a posição (i, j) do bit corrompido.

Note que se mais de um bit for corrompido, esse método é incapaz de corrigir o quadro. De maneira análoga, é possível gerar corrupções de pacotes que façam com que o método nem mesmo seja capaz de detectar erros no quadro.

4.2 *Checksum*

O *checksum* é um método relativamente simples de verificação de erros que já foi estudado no contexto de camadas superiores da pilha de protocolos TCP/IP — especificamente, nas camadas de rede e transporte. Como o nome sugere, o *checksum* é basicamente uma soma. Embora existam várias pequenas variações do método, nessa seção focaremos no chamado *Internet Checksum*, a variante usada tipicamente em protocolos da Internet (como o IP e o UDP, por exemplo).

No *Internet Checksum*, o pacote[2] é visto como uma sequência de valores de 16 bits. Se o tamanho do pacote não for múltiplo de 16 bits, realiza-se um *padding* — *i.e.*, introduz-se um último byte com zeros ao final do pacote para efeito do cálculo do *checksum*[3]. Esses valores são, então, somados em **complemento a um com 16 bits**. Na prática, isso significa que, para uma dada parcela da soma, se houver *overflow*, deve-se somar 1 ao resultado. Ao final da soma, inverte-se os bits do resultado (*i.e.*, bits iguais a 1 viram 0 e vice-versa). Esse último valor é o *checksum* do pacote.

[2]Como o método de *checksum* é usado por protocolos de outras camadas da pilha de protocolos TCP/IP, nesta seção utilizaremos o termo genérico *pacote*, ao invés de *quadro*.

[3]Repare que o pacote não é efetivamente mudado. Ele ainda é transmitido com o número exato de bits da sua versão original. Essa adição de bits é feita apenas para o cálculo do *checksum*.

O transmissor calcula o *checksum* do pacote e transmite ambas as informações: o pacote original e o *checksum* calculado. Do outro lado, o receptor recebe o pacote, calcula ele próprio o *checksum* e compara ao valor de *chechsum* recebido. Se os valores forem iguais, assume-se que o pacote está íntegro. Caso contrário, assume-se que há alguma corrupção.

Um aspecto interessante do *checksum* (ao menos da versão comumente utilizada na Internet) é a explicação de por que os bits são invertidos ao final da soma. A princípio, essa operação adicional pode parecer inútil. E, de fato, o método funcionaria igualmente bem — do ponto de vista da probabilidade de detecção de erros — sem essa inversão final.

A motivação para essa inversão está na simplificação da implementação. Lembre-se que a soma utilizada no *checksum* da Internet é efetuada em complemento a um. Nessa representação de números, a operação de inversão de um valor bit-a-bit corresponde a inverter o sinal do número. Assim, **o valor final do *checksum* da Internet é o negativo da soma dos valores de 16 bits que compõem o pacote**.

A vantagem dessa abordagem fica clara quando consideramos a implementação típica da verificação de *checksum* pelo receptor. Ao invés de separar o valor do *checksum* enviado pelo transmissor do restante do pacote para a verificação da integridade, o receptor simplesmente soma cada grupo de 16 bits do pacote, incluindo o *checksum*. Se o pacote for íntegro, o valor do campo *checksum* será exatamente o negativo da soma em complemento a um do restante do pacote. Logo, ao somar todo o pacote, incluindo o *checksum*, **um pacote íntegro deverá resultar em zero**. Qualquer valor diferente de zero sugere algum tipo de corrupção nos bits do pacote. Note que, nessa abordagem, a verificação do *checksum* pelo receptor pode usar exatamente a mesma rotina de cálculo de *checksum* usada na transmissão — acrescida do teste ao final de se o valor do *checksum* calculado é zero.

Note que o *checksum* é apenas um método para verificação de erros. Em caso de detecção de alguma corrupção, ele não permite a inferência de qual ou quais bits causaram o erro. Tipicamente, um pacote que não passa na verificação é simplesmente descartado.

Além disso, o *checksum*, embora normalmente melhor que uma paridade simples, não é particularmente bom na detecção de erros típicos introduzidos por enlaces de comunicação. Mais especificamente, o *checksum* costuma gerar **colisões de valores** para pacotes parecidos. Por exemplo, o *checksum* usado na Internet resulta no mesmo valor — $0xB3B6$ — quando calculado sobre as *strings* "testar", "tfssar", "settar" e "reutar". Isso aumenta a possibilidade dos chamados **falsos positivos**: o pacote é corrompido pelo enlace mas a versão corrompida coincidentemente tem exatamente o mesmo valor de *checksum*, fazendo com que o receptor não detecte os erros.

Outro problema que pode ocorrer no *checksum* — em qualquer dos mecanismos de verificação de erros, na verdade — são os **falsos negativos**. Suponha, por exemplo, que durante a transmissão do pacote, os bits do pacote original permaneçam íntegros, mas os bits do *checksum* sofram corrupção. A verificação de integridade no receptor certamente falhará, sugerindo ao receptor que os dados foram corrompidos. Como resultado, o receptor descartará o pacote quando, na verdade, apenas os bits de redundância — não importantes para o receptor do pacote — estão corrompidos. O receptor, no entanto, não tem como distinguir as duas situações.

4.3 CRC

O último mecanismo de detecção de erros que estudaremos nessa disciplina é o CRC (do inglês *Cyclic Redundancy Check*). A ideia do CRC é mais complexa — ou, ao menos, mais abstrata — que a dos demais métodos estudados até aqui. No entanto, o CRC é provavelmente o método de verificação de erros mais amplamente usado pelos protocolos da camada da enlace, justificando seu estudo nessa disciplina.

O método de CRC se baseia em uma abstração matemática conhecida como *corpos finitos* ou *corpos de Galois* — em homenagem ao matemático francês Évariste Galois. Não é objetivo dessa disciplina, no entanto, explicar em detalhes os aspectos matemáticos do CRC. Aqui, nos restringiremos a um estudo mais prático do método e de suas características principais.

De certa forma, o cálculo do CRC se assemelha ao do *checksum*, no sentido de que ambos são baseados em uma "conta" efetuada sobre os bits da mensagem original. A operação realizada pelo CRC, no entanto, dá mais peso à posição de cada bit, reduzindo a probabilidade de colisão dos valores de CRC de pacotes com conteúdos parecidos.

O algoritmo de computação do CRC trata um pacote como um **grande polinômio** com coeficientes iguais a 0 ou 1. Por exemplo, suponha um pacote constituído pela seguinte sequência de bits: 11001. Esse pacote é entendido pelo CRC como o polinômio $D = x^4 + x^3 + 1$. Em outras palavras, lendo-se da direita para a esquerda, o valor do bit da posição i se torna o coeficiente do termo em x^i.

Além do próprio pacote sobre o qual se deseja calcular o CRC, esse método utiliza um segundo polinômio chamado de **polinômio gerador**, que será denotado daqui em diante simplesmente por G.

Esse polinômio G é um parâmetro do método e deve ser previamente conhecido tanto pelo transmissor (que calculará o CRC) quando pelo receptor (que fará a verificação). O grau do polinômio G, que chamaremos simplesmente de r, é equivalente ao número de bits de redundância gerados pelo método.

De posse da representação polinomial do pacote (um polinômio D) e do polinômio gerador, o CRC do pacote é simplesmente o resto da divisão de $D \cdot x^r$ por G. Alguns aspectos práticos devem ser observados nessa divisão. Em primeiro lugar, todas as operações feitas entre os coeficientes dos polinômios envolvidos devem ser realizadas em módulo 2. Isso significa que:

- $0 - 0 = 0$.
- $1 - 0 = 1$.
- $1 - 1 = 0$.
- $0 - 1 = 1$.

Além disso, o resto da divisão de dois polinômios é, também, um polinômio. Logo, ao final da computação, os bits de redundância adicionados ao pacote correspondem aos coeficientes do polinômio do resto (do maior para o menor termo). É importante destacar ainda que o CRC calculado deve sempre ser representado com r bits, mesmo que o polinômio do resto contenha coeficientes zerados à esquerda.

Mas como é feita a verificação por parte do receptor? Basicamente, o receptor recebe dois conjuntos de bits: os bits do pacote original possivelmente corrompidos (que chamaremos de D') e os bits de CRC também potencialmente corrompidos (que chamaremos de R'). Basta ao receptor calcular a divisão de $D' \cdot r + R'$ por G. Se o resto for zero, assume-se que a mensagem está íntegra. Caso contrário, há alguma inconsistência.

Na prática, implementações do CRC não utilizam explicitamente a abstração de polinômios. Ao invés disso, o processo de divisão e cálculo do resto são geralmente executados diretamente sobre os bits do pacote e (da representação binária) do polinômio gerador utilizando-se algoritmos relativamente simples. Um desses algoritmos funciona da seguinte maneira:

- Os bits do pacote são dispostos em uma linha e os bits do polinômio gerador em outra. No caso da geração do CRC, os bits do pacote devem ser completados com r zeros ao final. Para verificação, os r bits mais à direita devem ser iguais ao valor de CRC recebido.
- Enquanto houver bits iguais a 1 na linha do pacote (desconsiderando-se os r bits mais à direita), deve-se:
  - Alinhar o bit mais à esquerda do polinômio gerador ao bit igual a 1 mais à esquerda do pacote.
  - Para cada bit do polinômio gerador, alterar o valor do bit correspondente do pacote para o resultado do *ou-exclusivo* entre os dois bits.

Quando esse algoritmo é usado para calcular o CRC pelo transmissor, ao final do processamento o valor do CRC será o valor dos r bits mais à direita. Já para a verificação do CRC, se ao final desse processo os r bits mais à direita forem todos iguais a zero, assume-se que a mensagem é íntegra (há alguma inconsistência, caso contrário).

Como citado no início dessa seção, o CRC é um método de verificação de erros amplamente adotado na prática. Como qualidades, pode-se citar sua simplicidade de implementação, sua capacidade de detectar erros comumente introduzidos por enlaces de comunicação, sua capacidade de parametrização — tanto o grau do polinômio, quanto o polinômio gerador em si podem ser escolhidos.

A escolha do polinômio gerador, inclusive, tem direta relação com os tipos de erros detectados pelo CRC. Isso significa que alguns polinômios são "melhores" que outros para certas situações. Para facilitar a escolha de "bons" polinômios, existem vários padronizados (*e.g.*, CRC32, CRC16-IBM, CRC40-GSM).

Independentemente dos parâmetros escolhidos, no entanto, o CRC pode falhar. Ambos os casos de falsos positivos e de falsos negativos, discutidos no contexto do *checksum*, se aplicam, também, ao CRC. De fato, isso é verdade para qualquer método desse tipo. Há sempre, por exemplo, a possibilidade do pacote original e da sua versão corrompida coincidentemente resultarem nos mesmos bits de redundância. Isso ocorre pelo simples fato de estarmos tentando "resumir" a informação de um pacote potencialmente grande em alguns poucos bits de redundância. Em outras palavras, as funções que mapeiam os pacotes aos valores dos bits de redundância nesses métodos necessariamente não são injetivas.

Mas o quão provável são essas colisões dos valores de CRC? Assumindo algumas hipóteses razoáveis na prática, pode-se modelar essas colisões através de um problema clássico da área de probabilidade chamado de *paradoxo do aniversário*. Através dessa modelagem, conclui-se que a probabilidade de dois pacotes quaisquer possuírem o mesmo valor de CRC é dada por:

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

onde e denota a base do logaritmo neperiano (aproximadamente $2,71$) e N denota a quantidade de valores diferentes de CRC possíveis. Note que com um CRC de r bits, há 2^r possíveis valores de CRC. Logo, N cresce exponencialmente com o aumento do número de bits usados no CRC. Para $r = 32$ bits (um valor bastante comum em aplicações em redes de computadores), essa probabilidade se aproxima de 10^{-10}.

Nota-se, portanto, que na prática, embora exista a probabilidade de falha, ela pode ser considerada desprezível para várias aplicações.

4.4 CRC *vs. checksum*

Ao longo da seção anterior, repetidamente citou-se que o CRC é superior ao *checksum* em termos de capacidade de detecção de erros típicos e que, de fato, o CRC é uma escolha comum em protocolos da camada de enlace. No entanto, em camadas superiores, protocolos como o IP e o UDP utilizam o *checksum*, e não o CRC. Qual seria o motivo para isso?

Uma justificativa comumente citada é a complexidade computacional. Embora o CRC seja relativamente simples, o *checksum* ainda tem uma complexidade computacional mais baixa, especialmente para implementações em *software*. Como historicamente a camada de enlace foi implementada majoritariamente em *hardware*, é razoável assumir a possibilidade de uso de um circuito especializado para o cálculo rápido do CRC. Já para as camadas superiores, muitas vezes implementadas em *software*, pode-se argumentar que o uso de um mecanismo de verificação de erros mais pesado computacionalmente pode afetar o desempenho da comunicação.