# Uma Análise da Eficácia dos Mecanismos de Detecção de Erros da Pilha de Protocolos da Internet

**Igor França Negrizoli**[1], **Luiz A. Rodrigues**[1] **e Marcio S. Oyamada**[1]

[1]Graduação e Pós-Graduação em Ciência da Computação
Universidade Estadual do Oeste do Paraná (UNIOESTE)
Cascavel – PR – Brasil

{firstname.lastname}@unioeste.br

***Abstract.*** *This paper presents an analysis of the transmission error detection mechanisms most used in TCP/IP networks, with special attention to their vulnerabilities. The objective is to evaluate the effectiveness of the mechanisms through the injection of errors in the application's data payload. The verification was done using raw sockets and a simulator, developed to generate scenarios according to statistical models. The results confirm that errors can pass unnoticed by the check procedures and reach the application layer.*

***Resumo.*** *Este artigo apresenta uma análise dos mecanismos de detecção de erros de transmissão mais utilizados em redes TCP/IP com atenção especial às suas vulnerabilidades. O objetivo é avaliar a eficácia dos mecanismos por meio de injeção de erros no campo de dados da aplicação. A verificação foi feita utilizando raw sockets e um simulador, desenvolvido para gerar cenários de acordo com modelos estatísticos. Os resultados confirmam que erros podem passar despercebidos pelos meios de verificação e alcançarem a aplicação.*

## 1. Introdução

A transferência de dados em redes de computadores está sujeita a diversos erros de transmissão, que podem ser causados por interferências eletromagnéticas ou falhas de componentes, tanto de hardware, quanto de software, gerando inversão ou inserção/supressão de bits. Além disso, a distribuição dos erros não é homogênea, podendo ocorrer em um único bit ou em rajadas, com vários bits consecutivos afetados [Maxino e Koopman 2009, Azahari et al. 2014, Koopman et al. 2015].

A pilha de protocolos da Internet, também conhecida como TCP/IP, dispõe de mecanismos de detecção de erros implementados em diferentes camadas. Nas camadas de Transporte e Rede, a soma de verificação (do inglês, *Checksum*) é o padrão dos protocolos TCP (*Transmission Control Protocol*), UDP (*User Datagram Protocol*), IP (*Internet Protocol*) e ICMP (*Internet Control Message Protocol*). Na camada de Enlace, o CRC (do inglês, *Cyclic Redundancy Check*) é utilizado pelos protocolos Ethernet (IEEE 802.3) e Wi-Fi (IEEE 802.11), entre outros [Kurose e Ross 2012]. Em relação ao escopo, o CRC, no nível do enlace, tem a responsabilidade de verificar as transmissões ponto-a-ponto, recalculando o código de detecção de erros a cada repasse do pacote. O mesmo ocorre com os pacotes IPv4, visto que os campos do cabeçalho podem ser modificados em cada salto e a soma deve ser refeita. Já o *Checksum*, é utilizado na comunicação fim-a-fim, sendo calculado e verificado somente pelos processos da aplicação (emissor e receptor).

Embora amplamente utilizados, o *Checksum* e o CRC possuem limitações em relação a eficácia da detecção de erros [Wolf e Blakeney 1988, Stone et al. 1998]. O Checksum, por exemplo, usa códigos de verificação de 16 bits e detecta desde modificações unitárias de bits a rajadas de até 16 bits. O CRC, por sua vez, implementa um mecanismo mais robusto, normalmente com códigos de $N = 32$ bits, e capacidade de detecção de erros de bit únicos até rajadas de erros de até $N$ bits, com alta probabilidade de detecção para rajadas maiores. No entanto, combinações de erros, especialmente em grandes rajadas, e falhas de verificação, podem passar despercebidos e alcançarem a aplicação [Pandurangan 2016].

O objetivo deste trabalho é apresentar uma análise dos mecanismos de detecção de erros implementados na pilha de protocolos da Internet por meio da utilização de estratégias de injeção de erros e simulação de modelos estatísticos que representam falhas de transmissão que podem ocorrer em diferentes camadas. Os resultados confirmam que erros podem passar despercebidos pelos diferentes mecanismos de verificação e alcançarem a camada de aplicação, podendo causar inconsistências e prejuízos aos usuários.

O restante deste trabalho está organizado nas seguintes seções. A Seção 2 descreve os mecanismos de detecção de erros da pilha TCP/IP. A Seção 3 apresenta os modelos de distribuição utilizados para a injeção de erros de comunicação. O simulador desenvolvido e os resultados de simulação são apresentados na Seção 4. A Seção 5 apresenta alguns trabalhos relacionados, incluindo soluções para mitigar as falhas de detecção. Por fim, a conclusão e os trabalhos futuros estão na Seção 6.

## 2. Detecção de Erros do TCP/IP

De acordo com Braden et al. (1988), mensagens com erros não detectados que alcançam as camadas superiores podem acabar sendo ignoradas, por não fazerem sentido no contexto do protocolo. No caso do TCP, por exemplo, erros de transmissão podem ocasionar a interrupção da conexão, o que, embora inconveniente, não atinge a integridade dos dados. Por outro lado, erros não detectados que atingem a camada de aplicação podem ser danosos, especialmente se a aplicação não possui um mecanismo próprio de verificação.

As duas principais técnicas de detecção de erros utilizadas na pilha de protocolos TCP/IP são o *Checksum* e o CRC, descritos a a seguir.

### 2.1. Checksum

O *Checksum* é o método de detecção utilizado para verificar se há erros de transmissão dos protocolos das camadas de transporte: TCP e UDP (cabeçalho e dados), e de rede: IPv4 (somente o cabeçalho) e ICMP. Estes protocolos possuem um campo de 16 bits que é enviado junto com os demais dados de identificação e controle.

O cômputo do *Checksum* está definido na RFC 1071 [Braden et al. 1988]:

1. Os bytes adjacentes são pareados para formar inteiros de 16 bits e a soma binária é realizada em cada bloco. Caso ocorra o bit de *carry* (vai um) na soma do bit mais significativo, o valor 1 (um) é adicionado ao resultado;
2. O campo em que está armazenado o *Checksum* não é considerado no cálculo e o complemento de um da soma (resultado binário da soma com os bits invertidos) é incluída no cabeçalho do protocolo;

3. A verificação do *Checksum* é feita pelo cálculo dos dados recebidos, incluindo o campo de *Checksum*, que deve resultar em todos os bits com valor 1 (um) para indicar uma transmissão possivelmente sem erros.

Como exemplo, considere que a sequência $A, B, C, D, ..., Y, Z$ representa uma cadeia de bytes com uma quantidade par de elementos. Neste caso, o *Checksum* é calculado como $[A, B] +' [C, D] +' ... +' [Y, Z]$, na qual $+'$ indica a soma por complemento de um dos inteiros de 16 bits formados pelos bytes adjacentes. Nos casos com quantidade ímpar de elementos, a última soma é realizada com o par $[Z, 0]$.

De acordo com Braden et al. (1988), o cálculo do *Checksum* possui duas propriedades matemáticas interessantes:

- É cumulativo e associativo: a soma pode ser dividida em blocos, desde que o tamanho par/ímpar da sequência seja respeitado. Por exemplo: $([A, B]+'[C, D]+' ... +' [J, 0]) +' ([0, K] +' ... +' [Y, Z])$;
- A ordem da soma é independente: a soma dos 16 bits dos pares pode ser feita em qualquer ordem. Por exemplo, $[B, A] +' [D, C] +' ... +' [Z, Y]$ tem o mesmo efeito que $[A, B] +' [C, D] +' ... +' [Y, Z]$.

A partir destas propriedades, é possível, por exemplo, implementar soluções paralelas, incluindo a otimização para máquinas que operam com múltiplos de 16 bits e que permitem operações do tipo $[A, B, C, D] +' .. +' [W, X, Y, Z]$.

Em termos de eficácia, o *Checksum* detecta todos os erros de bit único, mas pode falhar para múltiplos erros em um mesmo bloco e não detectar erros de sequência, visto que $[A + B] = [B + A]$. Em geral, se o valor de um bloco é incrementado e o valor do bloco seguinte é decrementando na mesma proporção (Figura 1), os dois erros não podem ser detectados, já que o resultado da soma continua o mesmo [Azahari et al. 2014]. Além disso, de acordo com [Braden et al. 1988], mesmo que o cálculo do *Checksum* no receptor indique uma transmissão correta, a mensagem ainda pode conter erros com uma probabilidade de $2^{-d}$, onde $d$ é a quantidade de bits do *Checksum*.

| Cálculo no envio | | | | | | | | | Cálculo no recebimento | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | **0** | 1 | 0 | 1 | | 1 | 0 | 1 | 0 | **1** | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | **1** | 0 | 1 | 0 | → | 0 | 1 | 0 | 0 | **0** | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

**Figura 1. Exemplo de alinhamento de erros não detectado pelo *Checksum*.**

Na prática, para gerar o *Checksum*, o campo correspondente no protocolo é zerado e o processo é realizado incluindo os demais campos. Além disso, para os protocolos TCP, UDP e ICMP, um pseudocabeçalho é utilizado no cálculo com o objetivo de evitar pacotes entregues equivocadamente pela camada de rede [Postel 1980]. A Figura 2 apresenta os dados utilizados, que variam de tamanho de acordo com a versão do IP, mas incluem endereços de origem e destino, código do protocolo da camada superior (TCP=6, UDP=17 e ICMPv6=58) e tamanho do pacote da camada superior (cabeçalho + dados). No caso do UDP, o campo "Tamanho" presente no protocolo é utilizado. Para os protocolos que não possuem este campo, o valor deve ser calculado [Deering e Hinden 1998].

O *Checksum* do UDP é opcional com o IPv4. Neste caso, o valor zero (`0x0000`) indica que não houve cálculo. Note que, quando o resultado do cálculo é zero, o valor enviado é `0xFFFF` (inversão do valor), mas se o cálculo resultar em `0xFFFF`, o valor é man-

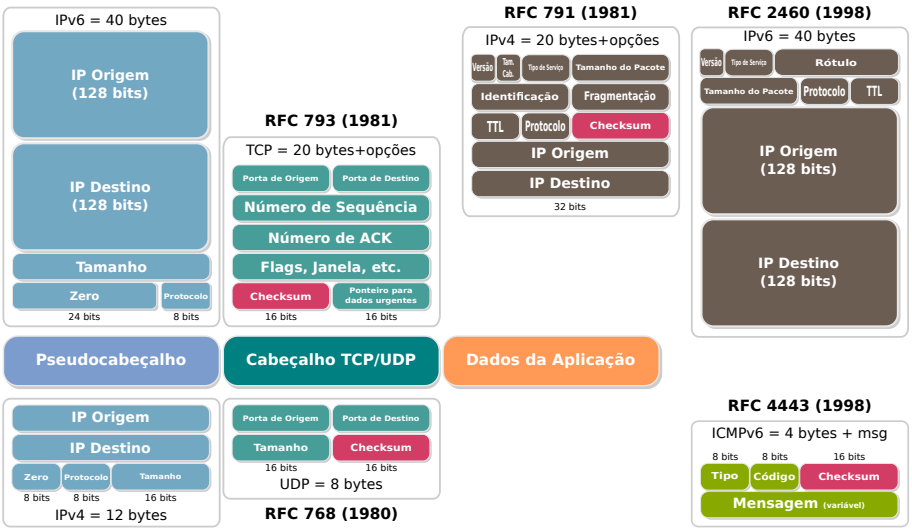


**Figura 2. Cabeçalhos utilizados no cálculo do *Checksum* do TCP/IP.**

tido sem inversão. Se o IPv6 é utilizado, o cálculo da soma de verificação do UDP é obrigatório e o receptor deve descartar pacotes com soma em zero [Deering e Hinden 1998].

No Protocolo IPv4, o *Checksum* é calculado apenas sobre o cabeçalho. Como o valor de TTL (*Time-to-Live*[1]) muda em cada salto, um novo cálculo é realizado em cada dispositivo no qual o cabeçalho IP é processado [Postel 1981]. Por esta razão, o campo foi retirado no IPv6 (Figura 2). Por outro lado, o protocolo de controle ICMPv6, adaptado para o IPv6, incluiu o campo para verificação em seu cabeçalho [Conta et al. 2006].

## 2.2. CRC

O CRC (*Cyclic Redundancy Check*) [Peterson e Brown 1961] utiliza códigos polinomiais para detectar erros de transmissão, especialmente os erros em bits consecutivos (rajadas). É o mecanismo utilizado pelos protocolos de enlace, tanto em redes cabeadas, como o IEEE 802.3 (Ethernet), quanto sem fio e móveis, como o IEEE 802.11 (Wi-Fi), Bluetooh, CDMA e LoRa. O processo realizado é o seguinte:

1. Emissor e receptor definem um polinômio gerador $G(x)$ (um polinômio com grau máximo $n$ possui $n + 1$ termos e, quanto maior for o seu grau, maior será a capacidade de detecção de erros);
2. Os blocos de dados são concatenados a um conjunto de zeros equivalente ao grau do polinômio e divididos por $G(x)$; a divisão é feita com operações binárias de ou-exclusivo (XOR);
3. O resto da divisão, em quantidade de bits equivalente ao grau do polinômio, é utilizado como CRC;
4. Para verificar a integridade dos dados recebidos, o receptor anexa ao dado original o próprio CRC recebido e efetua a divisão pelo polinômio acordado. Um erro é detectado se o resto da divisão for diferente de zero.

Considere, por exemplo, o polinômio de grau três $x^3 + x + 1$, que pode ser representado por $1x^3 + 0x^2 + 1x + 1$. Neste caso, os coeficientes são $1011$. Como o grau do

[1]O TTL (*Time-to-Live*) controla o número máximo de encaminhamentos do pacote antes de encontrar o receptor. O valor é decrementado em cada salto e, se chegar a zero, o pacote é descartado.

polinômio é três, três zeros são adicionados aos dados para a divisão e o CRC será os três bits do resto da divisão, como ilustrado na Figura 3.

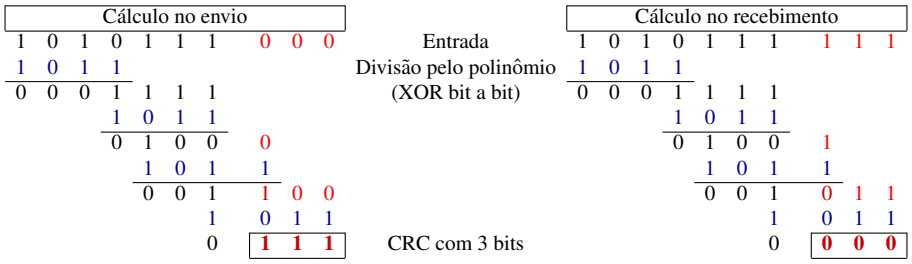


**Figura 3. Exemplo de Cálculo do CRC com polinômio 1011 (grau 3.)**

Em geral, os códigos CRC de $n$ bits detectam qualquer erro de um único bit, erros duplos, erros ímpares e rajadas de até $n$ bits [Azahari et al. 2014], podendo detectar rajadas maiores com uma probabilidade de $1 - 2^{-n}$ [Stone et al. 1998]. Como exemplo, o CRC-16 é um padrão para redes móveis (CDMA 2000) e LoRa (LongRange). Já o CRC-32 é usualmente utilizado nas transmissões da camada de enlace do padrão IEEE 802.3 (Ethernet) e 802.11 (Wi-Fi) [Rahmani et al. 2007]. No caso do padrão 802.3, o cálculo considera os dados de cabeçalho e dados. Cada protocolo pode utilizar um polinômio gerador específico, adequado de acordo com o meio de transmissão.

## 3. Modelos de Distribuição para Injeção de Erros de Comunicação

Dentro da teoria de probabilidades e estatística existem modelos de distribuição que podem ser aplicados na injeção de erros de transmissão, como os baseados em cadeias de Markov [Yu et al. 2008]. Estes modelos são comuns na modelagem de interferências em meios de comunicação [Tournoux et al. 2011], sendo utilizados neste trabalho para a injeção de erros nos pacotes em nível de *bit*. Os modelos utilizados são descritos a seguir.

### 3.1. Modelo de Bernoulli

O Modelo de Bernoulli [Jiang e Schulzrinne 2000] é baseado em um único parâmetro BER (*Bit Error Rate*) que varia no espaço contínuo $[0, 1]$, sendo $0 = 0\%$ e $1 = 100\%$. Uma taxa de $1\%$, por exemplo, é representada pelo valor $0,01$. Para uma sequência grande de $N$ bits, o número de erros se aproxima de $N * BER$. Uma característica deste modelo é que a probabilidade de um erro ocorrer independe do resultado anterior.

### 3.2. Modelo de Gilbert

Já o modelo de Gilbert [Gilbert 1960], também conhecido como *two-state* de Markov, permite a geração de modificações em rajada (*burst*). O modelo utiliza dois estados, conforme apresentado na Figura 4. O estado "1" *loss* representa a ocorrência do erro (inversão do bit), considerando que o bit anterior foi modificado. O estado "0" (*non-loss*) indica a ausência de modificações.

A probabilidade de transição do estado 0 para o estado 1 é dada por $p$ e $q$, e vice-versa. A probabilidade $q$ está relacionada à sequencialidade dos erros, isto é, ao tamanho da rajada. Uma vez no estado 1, a probabilidade de uma rajada de tamanho $k$ é $p_k = q(1 - q)^{k-1}$. A taxa de bits modificados $BER$ é dada por $p/(p + q)$. Erros em rajada podem ser modelados escolhendo-se apropriadamente valores para $p$ e $q$. Neste

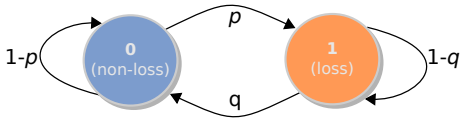


**Figura 4. Modelo de Gilbert (Fonte: [Jiang e Schulzrinne 2000]).**

trabalho foram utilizados para o cálculo de $p$ e $q$ a taxa de erros $BER$ e o tamanho médio das rajadas $k$, sendo $q = 1/k$ e $p = (q * BER)/(1 - BER)$.

É importante observar que quando $q$ é igual a $1 - p$, o modelo de Gilbert se resume ao modelo de Bernoulli. Além disso, para evitar algumas situações particulares, recomenda que os valores de $p$ e $q$ devem estar estritamente entre 0 e 1, com $0 \leq p, q \leq 1$. Quando esta condição não é satisfeita, podem ocorrer os seguintes cenários:

- Se $p = q = 1$: possui um comportamento conhecido como *flip-flop*. A cada passo o estado do autômato é alternado entre 0 e 1;
- Se $p = q = 0$: seja qual for o estado inicial, o mesmo será mantido em todos os passos subsequentes.
- Se $p = 0, 0 < q < 1$: se o processo inicia no estado 1, existe uma probabilidade $q$ de que, em algum passo, ocorrerá uma transição para o estado 0. Uma vez no estado 0, o mesmo será mantido em todos os demais passos. Quando isto ocorre, é dito que o processo foi absorvido no estado 0. O número de passos até que ocorra a absorção possui uma distribuição geométrica com média $1/q$. Obviamente, se o processo inicia no estado 0 ele é absorvido desde o início;
- Se $q = 0, 0 < p < 1$: o comportamento é o inverso do anterior, com a absorção ocorrendo no estado 1.

### 3.3. Modelo Periódico

O modelo de erros periódicos [Sousa e Ferreira 2007] utiliza uma abordagem determinística para gerar erros em intervalos constantes, podendo ser utilizado para erros únicos ou em forma de rajadas. O comprimento da rajada e do intervalo em que as erros ocorrem pode ser de tamanho fixo ou variar em um certo intervalo a cada período. O comprimento da rajada de erros é denotado $F$ ($F_{Min}$ e $F_{Max}$ para os casos onde o comprimento da rajada varia) e o intervalo com que esses erros ocorrem é denotado $T$ ($T_{Min}$ e $T_{Max}$ para os casos onde o comprimento do intervalo varia). Com isso, é possível simular cenários em que os erros ocorrem em rajadas de comprimento fixo ou variável e intervalos de ocorrência bem definidos, possivelmente alinhados verticalmente.

## 4. Resultados e Discussões

Para avaliar a eficácia dos mecanismos de detecção de erros utilizados na pilha de protocolos da Internet, foi desenvolvido um simulador[2] em C++ que permite criar pacotes com diferentes tamanhos e injetar erros (inversão de bits) de acordo com os modelos probabilísticos de Bernoulli, Gilbert e Periódico, descritos na Seção 3.

[2]Código-fonte disponível em `https://github.com/igorFNegrizoli/packetSenderSim`

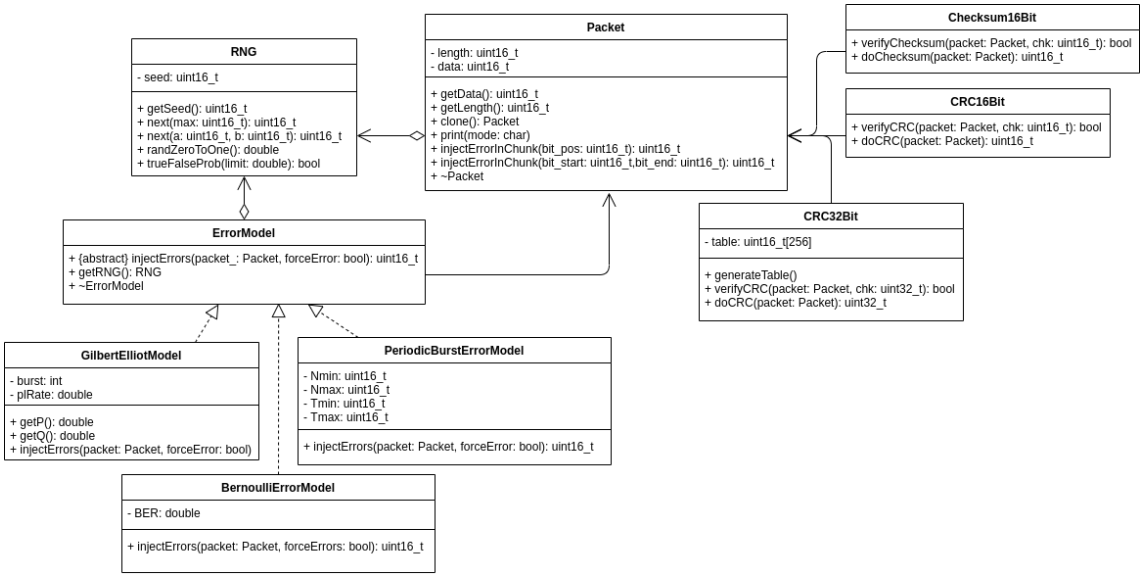


**Figura 5. Diagrama de classes do simulador desenvolvido.**

## 4.1. Ambiente de Simulação

O simulador foi implementado seguindo o paradigma de desenvolvimento de software orientado a Objetos. As classes elaboradas são apresentadas na Figura 5. A classe `RNG` é responsável por gerar números pseudo-aleatórios que serão atribuídos ao conteúdo (*payload*) dos pacotes, bem como nos modelos de injeção de erros. Para tanto, ela utiliza a biblioteca `random`[3] da linguagem C++. Uma semente inicial pode ser definida para permitir a reprodutibilidade dos experimentos.

Após a construção do pacote com os dados gerados, o cálculo de seu código de verificação é feito de acordo com os diferentes algoritmos implementados, como é feito pelo processo emissor. Em seguida, os erros de transmissão são injetados no pacote pelo modelo de erros instanciado (classes que implementam $ErrorModel$). Por fim, é feito novamente o cálculo dos códigos de verificação e comparado com o anterior, simulando a recepção do pacote no destinatário. Para que ao menos um erro de transmissão seja injetado em cada pacote (modelos com taxas de erro muito baixas podem não injetar erros em alguns pacotes), um parâmetro `forceError=true` pode ser utilizado para gerar ao menos um erro aleatório quando nenhum bit é modificado diretamente pelo modelo. Neste caso, como todos os pacotes terão ao menos um erro de bit, se o valor obtido pelo método de verificação é diferente, significa que o algoritmo obteve êxito em detectar o erro. Caso contrário, significa que o algoritmo de detecção de erros falhou em detectar o erro de transmissão injetado. Esse processo é repetido várias vezes e dados estatísticos referentes à eficácia dos algoritmos são gerados como pode ser observado nas tabelas expostas nesta seção.

## 4.2. Parâmetros de Simulação

A partir do simulador desenvolvido, foram elaborados diversos cenários de teste para avaliar a eficácia dos algoritmos de detecção de erros *Checksum* e CRC (16 e 32 bits).

[3] `https://www.cplusplus.com/reference/random/`

O tamanho dos pacotes $N$ variou em potência de 2 entre $2^3$ e $2^{10}$, mantendo-se menor que o tamanho de pacotes padrão da Internet de 1.500 bytes. Os pacotes foram submetidos a taxas de erros de $0,1\%$, $0,2\%$, $1\%$ e $2\%$ para os modelos de Bernoulli e Gilbert, e $6,25\%$ e $12,5\%$ para o modelo periódico.

Os códigos CRC utilizados nas simulações foram o CRC-16 (CCITT/X25), que é representado por `0x1021` ($x^{16} + x^{12} + x^5 + 1$), e para o CRC-32 foi usado o polinômio ITU-T (Autodin/Ethernet[4]), com valor `0x04C11DB7`.

Para cada tamanho de pacote, foram realizadas um milhão de execuções, cada uma delas simulando uma mensagem. Os pacotes foram preenchidos aleatoriamente utilizando a classe `RNG`, sendo a semente definida com o valor do tamanho do pacote $N$. Também foi definida uma semente fixa com o valor `0x2021` para a geração dos modelos de injeção de erros. Para cada cenário, foram calculados a média e o desvio padrão dos erros inseridos e foi registrado o número de execuções nas quais os mecanismos de detecção falharam. Para efeito de comparação, foi registrado ainda o número de ocorrências nas quais o *Checksum* e o CRC (16 ou 32 bits) falharam simultaneamente, isto é, nenhum dos dois métodos detectou o(s) erro(s) injetado(s).

### 4.3. Validação dos Modelos

Inicialmente, foram realizados alguns experimentos para validar a implementação dos modelos de Bernoulli, Gilbert e Periódico. As tabelas expostas a seguir apresentam as seguintes informações: o tamanho em bytes dos pacotes ($N(B)$), a porcentagem de pacotes que tiveram ao menos um bit invertido pelo modelo de erros (#pkts c/ erro), a média de bits que foram invertidos por pacote ($\bar{x}$), o desvio padrão desta média ($\sigma$), a porcentagem de bits invertidos no pacote, ou seja, $\bar{x}$ dividido pelo número de bits em cada pacote ($\%$), a quantidade de pacotes com erros injetados que não tiveram seus erros detectados pelos algoritmos de *Checksum*, CRC-16 e CRC-32 (CHK, CRC16 e CRC32 respectivamente), a quantidade de pacotes que não tiveram seus erros detectados tanto pelo algoritmo de *Checksum* quanto pelos algoritmos de CRC ($\cap_{16}$ para o *Checksum* em conjunto com CRC-16 e $\cap_{32}$ para o *Checksum* em conjunto com CRC-32). A Tabela 1 apresenta os resultados obtidos para o modelo de Bernoulli com $BER = 0,1\%$ e $BER = 1\%$. O parâmetro `forceError` com valor falso foi utilizado na Tabela 1 e na Tabela 2, isso faz com que alguns pacotes não apresentem nenhum erro de transmissão (isso acontece nos casos em que a probabilidade do erro ocorrer no modelo utilizado é muito baixa ou o tamanho do pacote é muito pequeno). Note que as taxas de erros são alcançadas de acordo com o esperado, quando o BER (Bit Error Rate) de 0,1% é passado como parâmetro para o simulador por exemplo, o modelo de erros produz um BER similar como é apresentado na última coluna da Tabela 1. Vale ressaltar que, para taxas baixas e pacotes com menor tamanho, nem todas as execuções têm erros injetados. Para pacotes de 8 bytes, por exemplo, apenas 6,2% das mensagens tiveram algum bit invertido para manter a taxa média de erros em 0,1%. Isto gera um valor médio de bits afetados $\bar{x}$ baixo e um desvio padrão $\sigma$ alto, visto que muitas execuções não tiveram bits modificados, mas foram incluídas na média. Resultado semelhante acontece com o modelo de Gilbert, apresentado na Tabela 2.

Para o modelo Periódico, as taxas de erros são determinadas pelos parâmetros de

[4]ITU-T T.810. Information technology – JPEG 2000 image coding system: Wireless. SERIES T: TERMINALS FOR TELEMATIC SERVICES. 2016. `https://www.itu.int/rec/T-REC-T.810/en`

**Tabela 1. Validação do modelo de Bernoulli implementado.**

| | BER = 0,1% | | | | BER = 1% | | | |
|---|---|---|---|---|---|---|---|---|
| N(B) | #pkts c/ erro (%) | $\bar{x}$ | $\sigma$ | BER | #pkts c/ erro (%) | $\bar{x}$ | $\sigma$ | BER |
| 8 | 6,22 | 0,06 | 0,25 | 0,1003 | 47,44 | 0,64 | 0,80 | 1,0009 |
| 16 | 11,97 | 0,13 | 0,36 | 0,0997 | 72,31 | 1,28 | 1,13 | 0,9997 |
| 32 | 22,57 | 0,26 | 0,50 | 0,0998 | 92,35 | 2,56 | 1,59 | 0,9994 |
| 64 | 40,02 | 0,51 | 0,71 | 0,0997 | 99,41 | 5,12 | 2,25 | 0,9996 |
| 128 | 64,08 | 1,02 | 1,01 | 0,1000 | 100,0 | 10,24 | 3,18 | 0,9997 |
| 256 | 87,09 | 2,05 | 1,43 | 0,0999 | 100,0 | 20,48 | 4,50 | 0,9999 |
| 512 | 98,33 | 4,10 | 2,02 | 0,1000 | 100,0 | 40,97 | 6,37 | 1,0002 |
| 1024 | 99,97 | 8,19 | 2,86 | 0,1000 | 100,0 | 81,91 | 9,00 | 0,9999 |

**Tabela 2. Validação do modelo de Gilbert para rajadas de tamanho 2 (k = 2).**

| | BER = 0,1% | | | | BER = 1% | | | |
|---|---|---|---|---|---|---|---|---|
| N(B) | #pkts c/ erro (%) | $\bar{x}$ | $\sigma$ | BER | #pkts c/ erro (%) | $\bar{x}$ | $\sigma$ | BER |
| 8 | 3,14 | 0,06 | 0,43 | 0,0972 | 27,66 | 0,63 | 1,34 | 0,9836 |
| 16 | 6,21 | 0,13 | 0,62 | 0,1000 | 47,73 | 1,27 | 1,92 | 0,9944 |
| 32 | 12,03 | 0.26 | 0,87 | 0,0996 | 72,64 | 2,55 | 2,72 | 0,9946 |
| 64 | 22,60 | 0,51 | 1,24 | 0,1000 | 92,49 | 5,11 | 3,87 | 0,9978 |
| 128 | 40,07 | 1,02 | 1,75 | 0,0998 | 99,43 | 10,23 | 5,47 | 0,9988 |
| 256 | 64,08 | 2,04 | 2,47 | 0,0997 | 100,0 | 20,47 | 7,75 | 0,9993 |
| 512 | 87,06 | 4,09 | 3,50 | 0,0998 | 100,0 | 40,94 | 10,95 | 0,9996 |
| 1024 | 98,35 | 8,20 | 4,96 | 0,1001 | 100,0 | 81,91 | 15,50 | 0,9999 |

rajada $F$ e intervalo $T$. A Tabela 3 apresenta o resultado para intervalos com $F = 1$ erro de bit a cada intervalo $T = 16$ bits e com erros rajadas $F = [1, 3]$ (entre um e três) bits a cada 16 bits. Estão incluídos também a quantidade de execuções nas quais os métodos de detecção falharam. Visto que o intervalo escolhido para T em ambos os cenários garante que pelo menos um erro estará presente em todos os pacotes, não se faz necessário um mecanismo para forçar que os erros ocorram como o `forceError`. As colunas $\cap_n$ indicam a quantidade de testes nos quais o CRC-$n$ falhou nos mesmos cenários que o *Checksum*.

**Tabela 3. Validação do modelo Periódico.**

| | F=1, T=16 | | | | | | | | F=[1,3], T=16 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N(B) | $\bar{x}$ | $\sigma$ | % | CHK | CRC16 | $\cap_{16}$ | CRC32 | $\cap_{32}$ | $\bar{x}$ | $\sigma$ | % | CHK | CRC16 | $\cap_{16}$ | CRC32 | $\cap_{32}$ |
| 8 | 4,0 | 0,0 | 6,25 | 375437 | 0 | 0 | 0 | 0 | 8,00 | 1,63 | 12.5028 | 47707 | 0 | 0 | 0 | 0 |
| 16 | 8,0 | 0,0 | 6,25 | 273451 | 0 | 0 | 0 | 0 | 16,00 | 2,31 | 12.5013 | 31879 | 657 | 190 | 0 | 0 |
| 32 | 16,0 | 0,0 | 6,25 | 196383 | 0 | 0 | 0 | 0 | 32.00 | 3.27 | 12.5016 | 22469 | 263 | 10 | 0 | 0 |
| 64 | 32,0 | 0,0 | 6,25 | 139866 | 0 | 0 | 0 | 0 | 64,00 | 4,62 | 12.5001 | 15838 | 234 | 3 | 0 | 0 |
| 128 | 64,0 | 0,0 | 6,25 | 98904 | 0 | 0 | 0 | 0 | 128,00 | 6,53 | 12.5003 | 11376 | 236 | 3 | 0 | 0 |
| 256 | 128,0 | 0,0 | 6,25 | 70579 | 0 | 0 | 0 | 0 | 256.00 | 9,24 | 12.5000 | 8046 | 209 | 2 | 0 | 0 |
| 512 | 256,0 | 0,0 | 6,25 | 49800 | 0 | 0 | 0 | 0 | 511.97 | 13.05 | 12.4994 | 5837 | 256 | 1 | 0 | 0 |
| 1024 | 512,0 | 0,0 | 6,25 | 35377 | 0 | 0 | 0 | 0 | 1023.99 | 18.46 | 12.4999 | 4027 | 215 | 0 | 0 | 0 |

Em relação ao tamanho da rajada (múltiplos erros que ocorrem um após ao outro), a Tabela 4 apresenta o comportamento dos modelos para pacotes com tamanho de 8 bytes apresentados nas tabelas anteriores. O número entre colchetes em cada coluna representa o comprimento (em bits) da rajada de erros detectada nos pacotes, o número à direita representa a quantidade de ocorrências para rajadas com aquele comprimento. Apenas os pacotes com erros injetados foram considerados. Para Bernoulli, a distribuição dos bits com erro é mais uniforme, o que gera poucas rajadas e desvio padrão menor (0,25)

(Tabela 1). Para Gilbert, os erros tendem a ocorrer de forma mais concentrada, com rajadas variando de 2 a 14 bits, o que justifica o desvio padrão maior (0,43) (Tabela 2). Nota-se também que um menor número de pacotes foi afetado com o modelo de Gilbert. No modelo periódico, considerando o intervalo de tamanho de rajada configurados, há uma distribuição equilibrada do comprimento da rajada de bits modificados conforme é esperado. Isso valida a consistência dos modelos de injeção de erros e fidelidade da distribuição dos erros injetados aos parâmetros de entrada.

**Tabela 4. Número de ocorrências de cada tamanho de rajada de erros para cada modelo.**

| Bernoulli | Gilbert | | Periódico |
|---|---|---|---|
| BER = 0.001 | BER = 0.001 k = 2 | | F = [1,3] T = 16 |
| [1] = 1000859 | [1] = 984287 | [5] = 893 | [1] = 1332865 |
| [2] = 64 | [2] = 7779 | [6] = 427 | [2] = 1332497 |
| | [3] = 3786 | ... | [3] = 1334638 |
| | [4] = 1848 | | |

### 4.4. Eficácia de Detecção do Checksum

Ao executar testes utilizando o modelo Periódico, registrados na Tabela 3, observa-se o efeito que os erros alinhados verticalmente têm sobre o *Checksum*. Os cenários foram configurados para erros com bit único (quadro da esquerda) e rajadas com [1,3] bits (quadro da direita). A coluna CHK indica o número de testes nos quais o método da soma falhou em detectar os erros em razão da propriedade comutativa da soma binária. Nos testes seguintes, diferentemente dos testes apresentados anteriormente sobre o modelo de Gilbert e de Bernoulli, o parâmetro `forceError` tem valor verdadeiro, ao menos um erro ocorre por pacote. Já que o objetivo destes testes é avaliar a eficiência da detecção de erros dos algoritmos em vez de validar os modelos de injeção de erros, não faz sentido permitir que pacotes sem erros injetados sejam avaliados. Caso ao final da execução do modelo de injeção de erros nenhum erro for injetado no pacote, um erro é injetado em um bit aleatório do pacote.

Quanto maior o tamanho dos pacotes, menores são as porcentagens de falhas de detecção apresentadas. Isso se deve a menor probabilidade de todos os pares de erros estarem alinhados verticalmente nos blocos de soma a medida que o comprimento dos pacotes aumenta.

O número de falhas de detecção do algoritmo de *Checksum* é alto em consequência dos erros perfeitamente alinhados byte a byte pelo modelo Periódico. Visto que no início de cada bloco de 16 bits pode ocorrer uma rajada de erros de um a três bits de comprimento, a probabilidade dos erros se alinharem verticalmente com erros de estado diferente é alta. Na Tabela 5 observa-se um desempenho melhor do algoritmo de *Checksum* quando os erros seguem a distribuição de Gilbert. Neste teste, a maioria das rajadas tem comprimento entre um e três e a probabilidade dessas rajadas ocorrerem varia de 0,1% a 2%. Diferente do modelo Periódico, essas rajadas não são perfeitamente alinhadas verticalmente. Também nota-se que o *Checksum* tem suas falhas complementadas pelos algoritmos de CRC, conforme é apresentado nas colunas $\cap_{16}$ e $\cap_{32}$. Isso indica que, quando usados em conjunto, eles podem detectar grande parte dos erros de transmissão em rajadas.

**Tabela 5. Eficácia dos algoritmos com erros injetados pelo modelo de Gilbert.**

| | k=3, BER=0,001 (0,1%) (p=0,0003, q=0,3333) | | | | | | | | k=3, BER=0,002 (0,2%) (p=0,0007, q=0,3333) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N(B) | $\bar{x}$ | $\sigma$ | % | CHK | CRC16 | $\cap_{16}$ | CRC32 | $\cap_{32}$ | $\bar{x}$ | $\sigma$ | % | CHK | CRC16 | $\cap_{16}$ | CRC32 | $\cap_{32}$ |
| 8 | 1,04 | 0,45 | 1,6264 | 0 | 0 | 0 | 8 | 0 | 1,08 | 0,63 | 1,6908 | 8 | 0 | 0 | 25 | 0 |
| 16 | 1,08 | 0,64 | 0,8466 | 4 | 0 | 0 | 29 | 0 | 1,17 | 0,92 | 0,9146 | 27 | 0 | 0 | 69 | 0 |
| 32 | 1,17 | 0,94 | 0,4583 | 30 | 0 | 0 | 77 | 0 | 1,35 | 1,34 | 0,5274 | 114 | 1 | 0 | 186 | 0 |
| 64 | 1,36 | 1,34 | 0,2647 | 107 | 0 | 0 | 178 | 0 | 1,73 | 1,95 | 0,3376 | 361 | 8 | 0 | 341 | 0 |
| 128 | 1,73 | 1,95 | 0,1692 | 372 | 8 | 0 | 373 | 0 | 2,55 | 2,89 | 0,2490 | 1104 | 26 | 0 | 689 | 2 |
| 256 | 2,55 | 2,89 | 0,1245 | 1064 | 33 | 0 | 720 | 0 | 4,34 | 4,30 | 0,2121 | 2320 | 75 | 1 | 1140 | 5 |
| 512 | 4,35 | 4,31 | 0,1061 | 2245 | 99 | 1 | 1246 | 3 | 8,25 | 6,32 | 0,2014 | 2536 | 196 | 0 | 1658 | 3 |
| 1024 | 8,25 | 6,30 | 0,1007 | 2563 | 205 | 1 | 1653 | 4 | 16,39 | 9,03 | 0,2001 | 883 | 249 | 0 | 1977 | 5 |
| | k=3, BER=0,01 (1%) (p=0,0034, q=0,3333) | | | | | | | | k=3, BER=0,02 (2%) (p=0,0068, q=0,3333) | | | | | | | |
| N(B) | $\bar{x}$ | $\sigma$ | % | CHK | CRC16 | $\cap_{16}$ | CRC32 | $\cap_{32}$ | $\bar{x}$ | $\sigma$ | % | CHK | CRC16 | $\cap_{16}$ | CRC32 | $\cap_{32}$ |
| 8 | 1,42 | 1,44 | 2,2263 | 127 | 2 | 0 | 137 | 0 | 1,89 | 2,08 | 2,9475 | 417 | 11 | 0 | 206 | 0 |
| 16 | 1,91 | 2,15 | 1,4900 | 543 | 3 | 0 | 329 | 0 | 2,94 | 3,15 | 2,2937 | 1392 | 13 | 0 | 551 | 0 |
| 32 | 2,96 | 3,23 | 1,1573 | 1418 | 29 | 0 | 689 | 1 | 5,25 | 4,73 | 2,0514 | 2576 | 83 | 0 | 1023 | 2 |
| 64 | 5,28 | 4,81 | 1,0315 | 2646 | 73 | 0 | 1167 | 3 | 10,23 | 6,92 | 1,9971 | 2150 | 166 | 0 | 1478 | 3 |
| 128 | 10,25 | 7,03 | 1,0012 | 2157 | 206 | 0 | 1659 | 2 | 20,45 | 9,89 | 1,9972 | 380 | 260 | 0 | 1513 | 2 |
| 256 | 20,45 | 9,99 | 0,9986 | 434 | 263 | 0 | 1798 | 0 | 40,93 | 13,99 | 1,9985 | 36 | 260 | 0 | 1682 | 0 |
| 512 | 40,93 | 14,14 | 0,9993 | 27 | 208 | 0 | 1851 | 0 | 81,89 | 19,78 | 1,9993 | 11 | 241 | 0 | 1566 | 0 |
| 1024 | 81,94 | 20,03 | 1,0002 | 9 | 278 | 0 | 1743 | 0 | 163,80 | 27,99 | 1,9995 | 12 | 252 | 0 | 1651 | 0 |

### 4.5. Eficácia de Detecção do CRC

Para os testes com o algoritmo de CRC-16, foi usado o polinômio `0x1021` (CCITT/X25). Os resultados evidenciam que o CRC, embora seja computacionalmente mais custoso, na maioria dos casos, é mais eficiente que o *Checksum* para a detecção de erros de transmissão, conforme já registrado nas Tabelas 3 e 5.

A Tabela 5 apresenta os resultados para o modelo de Gilbert com rajadas médias de três bits de comprimento. Nota-se novamente a maior eficiência do algoritmo de CRC para a detecção de erros em rajada. Mesmo que seja usado um polinômio de apenas 16 bits, a eficiência de detecção de erros de transmissão do algoritmo é superior ao algoritmo de *Checksum*, obtendo 0.02% de falhas de detecção de erros em seu pior teste.

Observa-se na Tabela 6 que o algoritmo de CRC continua sendo eficaz na detecção de erros não só nos cenários nos quais os erros são injetados em rajadas, mas também nas situações em que ocorrem erros esparsos, como os que são gerados pelo modelo de Bernoulli. O CRC, mesmo utilizando um polinômio de 16 bits, teve apenas 0.04% como sua maior taxa de falha de detecção. Além disso, observa-se que o algoritmo de CRC obtém sucesso na detecção dos erros nas situações em que o *Checksum* falha, mostrando novamente que utilizá-los em conjunto torna a detecção de erros mais eficiente.

Como complemento e visando avaliar o impacto da escolha do polinômio na eficácia de detecção, o polinômio para CRC-32 indicado em Koopman (2015), com valor $P_b =$`0xC9D204F5`, foi comparado com o CRC-32 ITU-T utilizado nos testes anteriores (representado por $p_a$). Segundo o autor, $P_b$ é o melhor polinômio de 32 bits com Distância de Hamming igual a 4, especialmente para os casos nos quais a taxa de erros é menor que 0,001% e os erros são constantes e independentes.

Os cenários testados, apresentados na Tabela 7, incluíram o modelo de Bernoulli com $BER = 0,001$ (Bernoulli 0,1%) e $BER = 0,01$ (Bernoulli 1%), o modelo de Gilbert com $BER = 0,01$ e $k = 3$ (Gilbert 1%) e $BER = 0,001$ e $k = 3$ (Gilbert 0,1%) e o modelo de rajadas periódicas com $F = [1, 3]$ e $T = 16$ (Periódico [1,3],16). Nota-se

**Tabela 6. Eficácia dos algoritmos com erros injetados pelo modelo de Bernoulli.**

| | BER=0,001 (0,1%) | | | | | | | | BER=0,002 (0,2%) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N(B) | $\bar{x}$ | $\sigma$ | % | CHK | CRC16 | $\cap_{16}$ | CRC32 | $\cap_{32}$ | $\bar{x}$ | $\sigma$ | % | CHK | CRC16 | $\cap_{16}$ | CRC32 | $\cap_{32}$ |
| 8 | 1,00 | 0,05 | 1,5656 | 41 | 0 | 0 | 0 | 0 | 1,01 | 0,09 | 1,5743 | 169 | 1 | 0 | 0 | 0 |
| 16 | 1,01 | 0,09 | 0,7872 | 194 | 0 | 0 | 0 | 0 | 1,03 | 0,18 | 0,8045 | 728 | 5 | 0 | 1 | 0 |
| 32 | 1,03 | 0,19 | 0,4024 | 758 | 2 | 0 | 12 | 0 | 1,11 | 0,37 | 0,4342 | 2393 | 3 | 0 | 61 | 0 |
| 64 | 1,11 | 0,37 | 0,2169 | 2442 | 3 | 0 | 155 | 0 | 1,38 | 0,72 | 0,2701 | 5857 | 21 | 0 | 445 | 0 |
| 128 | 1,38 | 0,72 | 0,1351 | 5987 | 104 | 0 | 527 | 0 | 2,18 | 1,27 | 0,2124 | 9215 | 203 | 0 | 1108 | 2 |
| 256 | 2,18 | 1,28 | 0,1062 | 9208 | 300 | 0 | 1177 | 1 | 4,11 | 1,99 | 0,2008 | 5784 | 337 | 0 | 1687 | 1 |
| 512 | 4,11 | 1,99 | 0,1004 | 5714 | 343 | 0 | 1682 | 0 | 8,19 | 2,86 | 0,2000 | 772 | 246 | 0 | 1883 | 2 |
| 1024 | 8,19 | 2,86 | 0,1000 | 792 | 244 | 0 | 1858 | 1 | 16,38 | 4,04 | 0,1999 | 42 | 226 | 0 | 1908 | 1 |
| | BER= 0.01 (1%) | | | | | | | | BER= 0.02 (2%) | | | | | | | |
| N(B) | $\bar{x}$ | $\sigma$ | % | CHK | CRC16 | $\cap_{16}$ | CRC32 | $\cap_{32}$ | $\bar{x}$ | $\sigma$ | % | CHK | CRC16 | $\cap_{16}$ | CRC32 | $\cap_{32}$ |
| 8 | 1,16 | 0,46 | 1,8195 | 2564 | 69 | 0 | 4 | 0 | 1,55 | 0,86 | 2,4255 | 5695 | 299 | 0 | 49 | 1 |
| 16 | 1,56 | 0,87 | 1,2157 | 6557 | 105 | 0 | 137 | 1 | 2,63 | 1,48 | 2,0585 | 7992 | 326 | 0 | 413 | 0 |
| 32 | 2,63 | 1,49 | 1,0291 | 8321 | 230 | 0 | 764 | 3 | 5,13 | 2,23 | 2,0031 | 3383 | 289 | 0 | 1088 | 1 |
| 64 | 5,12 | 2,24 | 1,0005 | 3609 | 245 | 0 | 1402 | 0 | 10,24 | 3,17 | 1,9998 | 254 | 237 | 0 | 1218 | 1 |
| 128 | 10,24 | 3,18 | 0,9996 | 289 | 219 | 0 | 1508 | 0 | 20,47 | 4,48 | 1,9993 | 31 | 251 | 0 | 1093 | 0 |
| 256 | 20,48 | 4,51 | 1,0001 | 19 | 270 | 0 | 1516 | 0 | 40,96 | 6,34 | 2,0000 | 19 | 250 | 0 | 1095 | 0 |
| 512 | 40,97 | 6,37 | 1,0002 | 16 | 239 | 0 | 1530 | 0 | 81,93 | 8,96 | 2,0001 | 13 | 229 | 0 | 1161 | 0 |
| 1024 | 81,94 | 9,01 | 1,0002 | 12 | 234 | 0 | 1490 | 0 | 163,83 | 12,67 | 1,9999 | 16 | 254 | 0 | 1201 | 0 |

uma eficácia expressiva do polinômio de Koopman para o conjunto de dados avaliado, exceto para o modelo Periódico, que possui taxas mais elevadas de erros.

**Tabela 7. Comparação da eficácia do CRC32 nos cenários anteriores com os polinômios $P_a$ (`0x04C11DB7`) e $P_b$ (`0xC9D204F5`).**

| | Bernoulli 0,1% | | Bernoulli 1% | | Gilbert 0,1% | | Gilbert 1% | | Periódico [1,3],16 | |
|---|---|---|---|---|---|---|---|---|---|---|
| N(B) | $P_a$ | $P_b$ | $P_a$ | $P_b$ | $P_a$ | $P_b$ | $P_a$ | $P_b$ | $P_a$ | $P_b$ |
| 8 | 0 | 0 | 4 | 0 | 8 | 0 | 137 | 0 | 0 | 0 |
| 16 | 0 | 0 | 137 | 2 | 29 | 0 | 329 | 0 | 0 | 190 |
| 32 | 12 | 0 | 764 | 7 | 77 | 0 | 689 | 0 | 0 | 35 |
| 64 | 155 | 0 | 1402 | 14 | 178 | 0 | 1167 | 6 | 0 | 34 |
| 128 | 527 | 2 | 1508 | 16 | 373 | 0 | 1659 | 16 | 0 | 36 |
| 256 | 1177 | 14 | 1516 | 22 | 720 | 2 | 1798 | 23 | 0 | 30 |
| 512 | 1682 | 18 | 1530 | 11 | 1246 | 10 | 1851 | 12 | 0 | 28 |
| 1024 | 1858 | 13 | 1490 | 14 | 1653 | 11 | 1743 | 12 | 0 | 37 |

Inicialmente, o fato do polinômio da ITU-T não ter um bom desempenho causa estranhamento pelo motivo deste polinômio ser utilizado em aplicações tradicionais como Ethernet, MPED-2, PNG e Gzip. Porém, este acontecido é totalmente compreensível, visto que os cenários podem não refletir fielmente as situações reais da rede.

## 5. Trabalhos Relacionados

O trabalho de Stone et al. (1998) avaliou o comportamento do *Checksum* e do CRC sobre dados reais de sistemas de arquivos UNIX. O resultado indica que estes mecanismos são mais eficientes para dados uniformes, o que não é o padrão para o cenário avaliado. Os autores sugerem como soluções para aprimorar a eficácia dos mecanismos de verificação de erros de transmissão o uso de compactação, bem como de outros algoritmos, como o de Fletcher. Em outro trabalho posterior, Stone e Partridge (2000), avaliaram 500 mil pacotes com falhas nos *Checksum* TCP, UDP e IP. A conclusão foi que o *Checksum* falha em detectar um pacote em cada 16 milhões a 10 bilhões de pacotes, dependendo do padrão dos dados, e sugerem o emprego de mecanismos de verificação na camada de aplicação.

Maxino e Koopman (2009) avaliaram a eficácia de diversas soluções de detecção de erros na transmissão de dados em dispositivos de redes embarcados. O estudo considerou erros aleatórios e de rajadas. De acordo com os autores, o *Checksum* baseado em complemento de um deve ser utilizado quando o objetivo é reduzir o custo computacional. O CRC, por outro lado, embora mais custoso, possui maior eficácia da detecção. O trabalho também compara outros mecanismos, como Fletcher, Adler e complemento de dois, sendo os dois últimos os mais indicados para o tráfego em redes de computadores.

O TCP MD5 [Heffernan 1998] é uma solução de autenticação de segmentos TCP para a transmissão de informações em conexões persistentes, como no BGP (*Border Gateway Protocol*). A proposta utiliza o método de assinatura com *hash* MD5, enviada no campo de opções do TCP, e inclui o pseudocabeçalho TCP IPv4, o cabeçalho TCP e os dados. A solução foi modificada e tornada obsoleta pela RFC 5925 [Touch et al. 2010] em razão das vulnerabilidades do MD5 e para incluir suporte ao IPv6. O novo padrão, denominado TCP-AO (*TCP Authentication Option*) utiliza protocolos MAC (*Message Authentication Codes*) para proteger as conexões longas de ataques de reprodução.

## 6. Conclusão e Trabalhos Futuros

A partir dos resultados obtidos em diferentes cenários simulados com variação do tamanho da sequência de dados e dos parâmetros dos modelos, foi observado que em diversos cenários os erros injetados são ignorados tanto pelo CRC quanto pelo *Checksum*. Embora com uma baixa probabilidade, está evidenciado que erros de transmissão podem passar despercebidos pelos mecanismos de verificação de erros implementados nas camadas do TCP/IP. Portanto, aplicações que confiam nos protocolos da rede podem receber dados incorretos, que podem causar desde falhas de execução a inconsistências em arquivos e bancos de dados e, consequentemente, perda de informações.

Como trabalhos futuros, pretende-se incluir ao simulador outros modelos de injeção de erros em pacotes e avaliá-los comparando os algoritmos de detecção já implementados com outras soluções de *Checksum* da literatura, como Fletcher e Addler, além de outros polinômios para o CRC. Com isso, será possível avaliar as melhores soluções para cenários específicos de comunicação de rede.

## Agradecimentos

Ao MEC-SESU, pelo financiamento do projeto de iniciação científica a partir do Programa de Educação Tutorial (PET).

## Referências

Azahari, A., Alsaqour, R., Uddin, M. e Al-Hubaishi, M. (2014). Review of error detection of data link layer in computer network. *ARPN J. Eng. and Applied Sciences*, 9(1):1–4.

Braden, R., Borman, D., Partridge, C. e Plummer, W. W. (1988). Computing the internet checksum. RFC.

Conta, A., Deering, S. e Gupta, M. (2006). Internet control message protocol (icmpv6) for the internet protocol version 6 (ipv6) specification. RFC.

Deering, S. E. e Hinden, R. M. (1998). Internet protocol, version 6 (ipv6) specification. RFC.

Gilbert, E. N. (1960). Capacity of a burst-noise channel. *The Bell System Technical Journal*, 39(5):1253–1265.

Heffernan, A. (1998). Protection of bgp sessions via the tcp md5 signature option. RFC.

Jiang, W. e Schulzrinne, H. (2000). Modeling of packet loss and delay and their effect on real-time multimedia service quality. In: *PROCEEDINGS OF NOSSDAV '2000*.

Koopman, P., Driscoll, K. R. e Hall, B. (2015). Selection of cyclic redundancy code and checksum algorithms to ensure critical data integrity. Technical report, U.S. Department of Transportation.

Kurose, J. F. e Ross, K. W. (2012). *Redes de Computadores e a Interent: uma abordagem top-Down*. Pearson, 6 ed.

Maxino, T. C. e Koopman, P. J. (2009). The effectiveness of checksums for embedded control networks. *IEEE Trans. on Dependable and Secure Computing*, 6(1):59–72.

Pandurangan, V. (2016). Linux kernel bug delivers corrupt TCP/IP data to Mesos, Kubernetes, Docker containers. https://tech.vijayp.ca/linux-kernel-bug-delivers-corrupt-tcp-ip-data-to-mesos-kubernetes-docker-containers-4986f88f7a19. Acesso em: 27/02/2019.

Peterson, W. W. e Brown, D. T. (1961). Cyclic codes for error detection. *Proceedings of the IRE*, 49(1):228–235.

Postel, J. (1980). User datagram protocol. RFC.

Postel, J. (1981). Internet protocol. RFC.

Rahmani, M., Hintermaier, W., Muller-Rathgeber, B. e Steinbach, E. (2007). Error detection capabilities of automotive network technologies and ethernet - a comparative study. In: *2007 IEEE Intelligent Vehicles Symposium*, pp. 674–679.

Sousa, P. B. e Ferreira, L. L. (2007). Bit error models. Technical report, Polytechnic Institute of Porto.

Stone, J., Greenwald, M., Partridge, C. e Hughes, J. (1998). Performance of checksums and CRC's over real data. *IEEE/ACM Trans. Netw.*, 6(5):529–543.

Stone, J. e Partridge, C. (2000). When the crc and tcp checksum disagree. In: *Proc. Conf. on Appl., Tech., Arch.s, and Protocols for Comp. Comm.*, SIGCOMM '00, pp. 309–319, New York, NY, USA. ACM.

Touch, J., Mankin, A. e Bonica, R. (2010). The tcp authentication option. RFC.

Tournoux, P. U., Lochin, E., Lacan, J., Bouabdallah, A. e Roca, V. (2011). On-the-fly erasure coding for real-time video applications. *IEEE Transactions on Multimedia*, 13(4):797–812.

Wolf, J. K. e Blakeney, R. D. (1988). An exact evaluation of the probability of undetected error for certain shortened binary CRC codes. In: *MILCOM 88, 21st Military Communications Conference*, pp. 287–292 vol.1.

Yu, X., Modestino, J. W. e Tian, X. (2008). The accuracy of markov chain models in predicting packet-loss statistics for a single multiplexer. *IEEE Transactions on Information Theory*, 54(1):489–501.