Análise de Códigos Detectores de Erros Utilizados na Camada de Transporte

Ricardo da Silva Barboza
Depto. Eletrônica e Sistemas
CTG - UFPE
Recife, PE, Brazil
rbarboza@ee.ufpe.br

Rafael Dueire Lins
Depto. Eletrônica e Sistemas
CTG - UFPE
Recife, PE, Brazil
rdl@ufpe.br

Ricardo Campello de Souza
Depto. Eletrônica e Sistemas
CTG - UFPE
Recife, PE, Brazil
ricardo@ufpe.br

***Resumo* – Este artigo analisa as taxas de falhas na detecção de erros em alguns dos códigos binários detectores de erros mais utilizados na camada de transporte. Foi desenvolvido, em software, um gerador de erros e testados conjuntos de dados uniformemente distribuídos e de dados enviesados. Os resultados mostraram altas taxas de falhas nesses códigos quando utilizados parâmetros que geram padrões de erros que alteram poucos bits nas palavras código. Em oposição ao que tem sido apresentado em artigos recentes, o *Internet Checksum* exibiu melhor desempenho quando os dados utilizados eram não uniformemente distribuídos.**

***Palavras-chave* – Detecção de erros, TCP-IP, Protocolos de Comunicação.**

***Abstract* – This paper analyses the failure rates in the detection of errors in some of the most frequently used binary error detecting codes in the transport layer of the TCP-IP protocol. A simulator that generates errors in uniformly and non-uniformly distributed data was developed. The results obtained show high failure rates in checksum codes whenever the errors modify few bits in the code words. Contrariwise to what has been reported in recent papers, the Internet Checksum exhibited better performance when non-uniformly distributed data was used.**

***Keywords* – Error detection, TCP-IP, Communication protocols.**

I. INTRODUÇÃO

Desde a padronização do *Internet Protocol* (IP) [12], dois protocolos de transporte estão disponíveis para aplicativos da camada de aplicação: o *Transmission Control Protocol* (TCP) [13] e o *User Datagram Protocol* (UDP) [11]. Devido a necessidades de novos serviços requeridos por aplicações recentes [10], atualmente estão sendo desenvolvidos outros protocolos para a camada de transporte pelo *Internet Engineering Task* Force (IETF), como o *Stream Control Transport Protocol* (SCTP) [10], o *Datagram Congestion Control Protocol* (DCCP) [5] e o *UDP Lite Protocol* [7].

Sendo um dos objetivos da camada de transporte o fornecimento de comunicação fim-a-fim de forma confiável, geralmente são empregados códigos de detecção de erros para verificar se a informação não foi alterada pelo canal de comunicação.

No caso do SCTP, o código detector de erro escolhido foi o Adler32 [10], e devido a um trabalho de J. Stone [17], foi verificado que para pacotes de dados de comprimento pequeno o Adler32 apresenta uma detecção de erros inferior ao esperado.

Segundo Stone, o problema encontra-se em um acumulador de 16 bits, chamado *S1*, que é a soma dos bytes do bloco de dados de entrada módulo 65521 (o maior primo menor que 2^{16}). Como *S1* acumula valores utilizando oito bits por vez, o bloco de entrada deve ter no mínimo 257 bytes para que todos os bits deste acumulador sejam afetados. Este quadro se agrava com o fato de que os dados que geralmente são transportados não possuem uma distribuição uniforme, resultando em uma probabilidade maior de um erro deixar um pacote danificado com um valor válido do que se todos os valores fossem uniformemente iguais. Stone [18] advoga que este problema atinge os códigos que geram seus bits de paridade através de alguma forma de soma dos dados de entrada. SCTP foi projetado para inicialmente transportar mensagens de sinalização (SS7) da rede pública de telefonia comutada (PSTN), que possuem menos que 128 bytes, o grupo que desenvolveu este protocolo alterou o método utilizado para detecção de erros.

A finalidade dos códigos detectores de erros é diminuir a taxa de erro das mensagens entregues a um receptor. Como nem todos os erros podem ser detectados, sempre existe uma taxa de erro residual [4]. Desta forma, este trabalho tem como objetivo, analisar vários códigos detectores de erros binários levando em conta suas características de detecção de erros em canais pouco ruidosos e que possuem erros em surtos, e encontrar suas taxas de falha. Os resultados desta análise podem ser de grande ajuda para a escolha de um entre os seguintes códigos detectores de erros empregados atualmente em protocolos da camada de transporte:

- Adler – Criado por Mark Adler para a biblioteca de compressão ZLIB [9] e utilizado na versão inicial do Protocolo SCTP;
- Fletcher – Proposto por John Fletcher [3] e utilizado na Camada de Transporte do Modelo OSI (*Open Systems Interconnection*) [15]. Neste trabalho serão analisadas quatro versões, duas dessas geram 16 bits de paridade e as outras duas 32 bits de paridade;
- *Internet Checksum* – Utilizado pelo conjunto de protocolos TCP/IP, em sua versão de 16 bits. Será analisada também uma versão pouco estudada que gera 32 bits de paridade;
- *Cyclic Redundancy Checks* (CRC) – Os códigos de redundância cíclica estão entre os mais utilizados em comunicação digital [20]. Analisaremos os que possuem polinômio gerador do CCITT (16 bits), o utilizado em redes locais Ethernet (CRC32B) e o encontrado por Castagnoli [2] (CRC32C). Este último substituiu o Adler32 no SCTP.

Na análise de desempenho, serão gerados pacotes de tamanhos definidos, a partir de dados reais e randômicos. Erros em surto e erros aleatórios serão introduzidos. O nosso interesse é calcular a Probabilidade P_{ue} de não detecção de erros (*Probability of undetected error*) dos códigos em diversas situações. A capacidade teórica de detecção de erros desses códigos é apresentada na Tabela 1.

Tabela 1 – Capacidade teórica de detecção de erros

Código	d_{min}	Surto**	P_{ue}
CRC16 CCITT	4 até 32.752 bits*	16	$1{,}53 \times 10^{-5}$
CRC32B	3 até 2.147.483.616 bits*	32	$2{,}33 \times 10^{-10}$
CRC32C	4 até 1.073.741.792 bits*	32	$2{,}33 \times 10^{-10}$
TCP16	2	15	$1{,}53 \times 10^{-5}$
TCP32	2	31	$2{,}33 \times 10^{-10}$
Fletcher16 255	3 até 2.040 bits*	7	$1{,}53 \times 10^{-5}$
Fletcher16 256	2	16	$1{,}53 \times 10^{-5}$
Fletch32 65535	3 até 524.280 bits*	15	$2{,}33 \times 10^{-10}$
Fletch32 65536	2	32	$2{,}33 \times 10^{-10}$
Adler32	3	23	$2{,}33 \times 10^{-10}$

* comprimento da mensagem a ser codificada.

**Menor comprimento de surto não 100% detectável.

Na Tabela 1, acima, a coluna d_{min} apresenta a distância mínima do código. Em alguns casos, a distância mínima pode variar com o tamanho da mensagem a ser codificada. Por exemplo, no caso do CRC16 CCITT, a distância mínima é d_{min}= 4 para mensagens de tamanho até 32.752 bits.

II. METODOLOGIA

Vários autores definem o erro em surto como sendo um padrão de erro que afeta *b* bits, onde o padrão de erro começa e termina com um valor diferente de zero [20][19][6]. Neste trabalho adotamos o modelo de surto de Wolf e Chun [21] (*b*:*p*), onde os erros ocorrem randomicamente dentro de um intervalo de *b* bits, com uma probabilidade *p* de erro de cada bit. Os códigos estudados em [21] foram os CRCs, e para eles o valor de $p = 0{,}5$, era considerado o pior caso. Com esta nova definição de surto, onde *p* pode ser diferente de 0,5, para um determinado *b* podem existir valores de *p*, diferentes de 0,5 que maximizam a P_{ue} do surto (*b*:*p*).

Desta forma, implementamos dois modelos de erros:

- Erros em surto afetando *b* bits consecutivos na palavra código, conforme definição de Wolf e Chun;
- Poucos erros independentes afetando alguns bits na palavra.

III. PARÂMETROS DE SIMULAÇÃO

Ao trabalharmos com o modelo de erros independentes, teremos apenas um parâmetro específico deste modelo, o número de bits afetados na palavra código. Este parâmetro irá variar de 2 a 9 bits que serão escolhidos randomicamente na palavra código. Estes valores refletem a baixa quantidade de bits que são afetados por este modelo [6][4].

O comprimento do surto *b* assumirá os valores de 8, 16, 24, 32, 40, 48, 56 e 64 bits, devido ao fato que na camada de transporte a checagem de erros é realizada somente dentro do computador de destino, até lá a camada de enlace de dados é responsável pela detecção de erros. Então, se um erro ocorrer há grande probabilidade de ter sido devido a falha de software ou em algum barramento de dados como existe entre a placa de rede e o processador ou entre a memória e o processador, como nesse caso os dados são transmitidos em paralelo por um múltiplo de 8, já que os computadores atuais trabalham com palavras múltiplas de 8 bits [16][3]. Já *p* irá variar de 0,01 (1%) a 1 (100%) em incrementos de 0,1 para que possamos encontrar a máxima P_{ue} dos códigos. A distribuição da quantidade de bits alterados no modelo de surto (*b*:*p*) de Wolf e Chun, segue a distribuição de Bernoulli (binomial), pois dentro da amplitude *b* do surto (número de provas), cada bit é alterado independentemente do outro por uma probabilidade *p*.

Para tornarmos o resultado da simulação o mais próximo de uma rede real, e para verificarmos se existe qualquer fraqueza dos códigos quando as mensagens possuem dados não uniformemente distribuídos, além dos dados randômicos, que são geralmente empregados em simulações como esta [4], utilizaremos mais 3 tipos de dados nos testes. Os tipos de dados empregados no teste consistem de:

- Dados uniformemente distribuídos – utilizou-se a função random() (C++ Builder 5 SP1), um gerador congruente multiplicativo de números randômicos com período igual a 2^{32} para retornar sucessivos números pseudo randômicos.
- Mensagens SS7 – sinalização entre centrais telefônicas, o arquivo em questão possui vários minutos de troca de informação real, como requisições de estabelecimento de chamadas. O SCTP foi desenvolvido inicialmente para transportar este tipo de dados.
- Português – texto (ASCII), de obras clássicas da língua portuguesa incluindo Machado de Assis, José de Alencar e Aluísio de Azevedo.
- Inglês – arquivo com todas as obras de Shakespeare codificadas em HTML.

Outro parâmetro utilizado na simulação foi o comprimento da mensagem codificada. Em redes Ethernet o limite de um pacote é de 12.144 bits (1518 bytes). O algoritmo de descoberta de MTU (*maximum transmission unit*) utilizado pelo TCP/IP [8] verifica o comprimento máximo dos pacotes que podem trafegar entre dois pontos antes de iniciar a transmissão de dados, limitando o comprimento máximo da maioria dos pacotes na Internet. Dessa maneira, geramos pacotes com 1.500 bytes, que simula o comprimento máximo de quadro que pode transitar em redes Ethernet e no TCP/IP simultaneamente. Segundo Stone [17] as mensagens de sinalização (SS7) são geralmente menores que 128 bytes, por isso para verificarmos palavras código de comprimento pequeno iremos gerar também pacotes de 100 bytes.

IV. VALIDAÇÃO DOS RESULTADOS

Apesar dos resultados anteriores com surtos (*b*:0,5) serem próximos aos calculados teoricamente, para validar o modelo de surto de Wolf e Chun, comparamos os resultados obtidos com os em [21]. Inicialmente, eles procuraram a probabilidade *p** que maximizava a P_{ue} dos códigos CRC16, CRC16 CCITT e CRC16 Q utilizando um surto de comprimento $b = 20$ (P_{ue}(20:*p**)). Através da variação do valor de *p* de 0,1 a 1,0. Em [21], a *p** do

CRC16 CCITT é igual a 0,2. Neste trabalho, utilizando os mesmos parâmetros encontramos o mesmo valor de p^*, ou seja, 0,2. O gráfico também se comportou de forma idêntica ao de [21], apresentando dois máximos locais, um para p menor que 0,5 e um para p acima de 0,5, como pode ser visto na Figura 1.

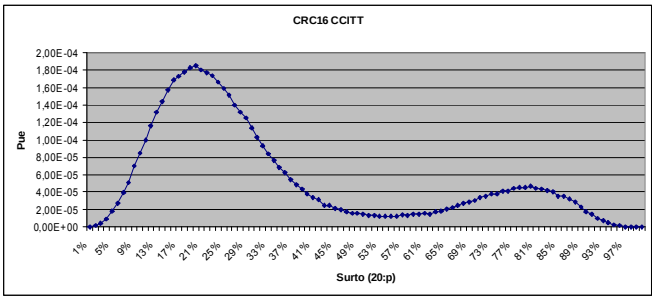

Figura 1 – Resultados da aplicação de surto (20:p) no código CRC16 CCITT. (Dados: Randômicos; Comprimento da mensagem: 100 bytes). Escala linear.

Após encontrar o p^* do CRC16 CCITT, para surto de comprimento b = 20, Wolf e Chun variaram o parâmetro b de comprimento do surto entre 16 e 32 bits. Utilizando estes mesmos parâmetros obtivemos valores praticamente idênticos aos de Wolf e Chun até 23 bits. Utilizamos dois algoritmos diferentes, um que gerava as palavras código e depois adicionava o ruído e outro que somente gerava o ruído para verificar se era uma palavra código do CRC16 CCITT válida, os mesmos resultados foram encontrados nos dois casos.

V. RESULTADOS OBTIDOS

Apresentaremos a seguir os resultados obtidos para cada um dos códigos analisados.

A. CÓDIGOS CÍCLICOS

Os CRCs obtiveram os melhores resultados de detecção de erros, seu comportamento, como teoricamente esperado, não é influenciado pelos tipos de dados utilizados na simulação nem pelo comprimento da palavra código, somente pelo modelo de erro empregado. Como teoricamente CRC32B e CRC32C devem apresentar uma falha em 2^{32} ($4{,}29 \times 10^9$) bits e utilizamos 5×10^9 iterações (5×10^7 iterações x 100 passos), geralmente só um erro passava despercebido por estes dois códigos em cada um dos testes realizados.

No surto (b:0,5) o CRC16 CCITT apresentou comportamento que se repetiu na maioria dos códigos. Quando teoricamente ele deveria detectar todos os erros (b = 8, 16), P_{ue} foi igual a zero, nos outros casos (b = 24, 32, 40, 48, 56, 64), P_{ue} manteve-se em 2^{-16}.

No modelo de erros independentes, o CRC16 CCITT se comportou de uma forma única entre os códigos estudados. Devido ao seu polinômio gerador ser da forma $p(x)(x + 1)$, todos os erros com peso ímpar foram detectados, mas com erros de peso par, P_{ue} foi o dobro do valor teórico, ou seja, $2x2^{-16}$(Figura 2). Na literatura, somente Fletcher [3] cita que apesar de alguns CRCs detectarem todos os erros de peso ímpar, eles teriam algum tipo de fraqueza em algum outro tipo de erro [1].

Como utilizamos mensagens de 100 e 1.500 bytes, e em ambos o CRC16 CCITT possui a mesma distância mínima, também não houve influência do comprimento das mensagens nos resultados.

Verificamos também que à medida que aumentávamos o comprimento do surto, havia uma diminuição dos dois máximos locais encontrados por Wolf e Chun [21].

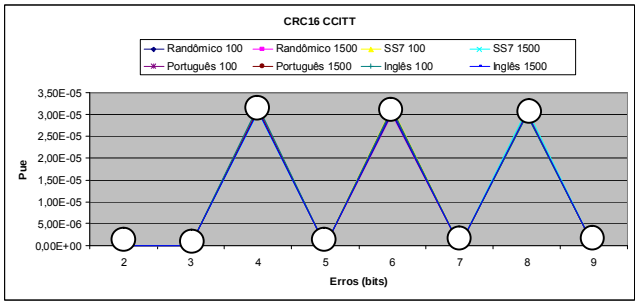

Figura 2 – CRC16 CCITT; Erros aleatórios, variando o tipo de dados e o comprimento da mensagem.

B. INTERNET CHECKSUM

O TCP16 e o TCP32 apresentaram comportamentos semelhantes, ambos possuem uma altíssima taxa de falhas quando os padrões de erros alteram poucos bits (atingindo 1 falha para 100 detecções corretas). Neste caso eles possuem as piores taxas de detecção encontradas.

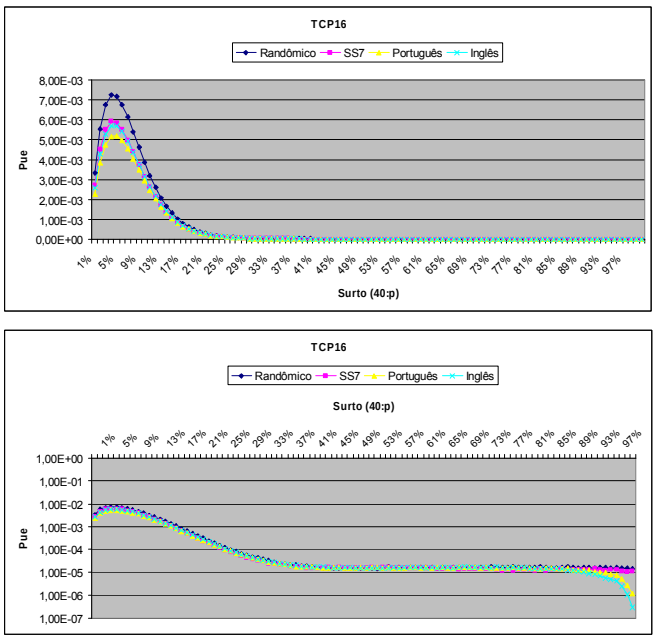

Figura 3 – TCP16; Surto (40:p), variando o tipo de dados e a probabilidade do surto (a) Escala linear; (b) Escala logarítmica.

Não encontramos, porém, resultados que dão apoio à idéia que dados não uniformemente distribuídos afetem de forma negativa

e significativa a capacidade de detecção de códigos como este, pois dos tipos de dados incluídos na simulação, o que apresentou o pior resultado foi o que possuía dados uniformemente distribuídos (dados randômicos). Ao contrário do CRC16 CCITT, o *Internet Checksum* é afetado pela distribuição dos dados. Foi verificada uma diferença de 60% entre os valores obtidos entre os dados randômicos e o texto em português (Figura 3). Possivelmente, esse fato deve-se à maneira de decodificar e não ao código (linear).

A única situação em que os dados não uniformemente distribuídos apresentaram uma taxa de falhas maior em relação aos dados uniformemente distribuídos foi quando havia uma condição inicial para ocorrência da falha na detecção do erro. No caso do TCP16 isto ocorre quando um surto de 16 bits substitui a palavra 0000_H pela $FFFF_H$ e vice versa. Então, se os dados forem uniformemente distribuídos e a probabilidade de ocorrência destas palavras no fluxo de dados é de $2x2^{-16}$, e com um surto no qual $p = 0,5$, a probabilidade de falha é de $2x2^{-16} \times 2^{-16} = 2x2^{-32} = 4,66x10^{-10}$. Dentre os dados analisados, as mensagens SS7 apresentam o maior número dessas palavras, esta probabilidade foi maior, em torno de $5x10^{-8}$. Quando p é igual a 1, utilizando dados uniformemente distribuídos, a probabilidade teórica de falha é de $2x2^{-16} = 3,05x10^{-5}$; encontramos em nossa simulação o valor de $3,12x10^{-5}$.

C. FLETCHER

Em geral os códigos Fletcher possuem altas taxas de falhas em padrões de erros que alteram poucos bits, mas não chegam aos níveis do TCP16 e TCP32. Um dos fatores determinantes das características do *checksum* de Fletcher foi a da aritmética de complemento a um ou complemento a dois. Desta forma, podemos separar nossa análise em dois grupos. No primeiro, do complemento a um, implementamos os códigos Fletcher16 255 e Fletcher32 65535. Do outro grupo foram implementados os códigos Fletcher16 256 e Fletcher32 65536.

O grupo formado pelo complemento a um possui um desempenho 10 vezes melhor, em relação ao complemento a dois, na máxima P_{ue} com o modelo de surto, mas não há garantia de detecção de surtos relativamente pequenos. Por exemplo, o Fletcher16 255 só detecta todos os surtos até o comprimento de 7 bits, o pior resultado desta característica de detecção entre os códigos estudados. Como observado no *Internet Checksum*, este grupo de códigos é influenciado pelo tipo de dados sendo a diferença de 50% entre o pior e melhor caso.

Outro comportamento que também encontramos, nesse grupo de códigos, foi aquele em que certas falhas em detectar erros necessitavam de uma condição inicial. O resultado é que os tipos de dados que apresentavam um maior número dessas condições, como as palavras FF_H e 00_H, para o Fletcher16 255 e $FFFF_H$ e 0000_H para o Fletcher32 65535, também possuíam as maiores taxas de falhas. Por exemplo, as taxas de falhas para o surto (8:0,5) e surto (16:0,5), são maiores quando utilizamos os dados que possuíam mensagens SS7. O surto (8:1,0) aplicado no Fletcher16 255 apresentou alta taxa de falhas, neste ponto calculamos a P_{ue} teórica. Para que esse erro ocorra, é necessário que no fluxo de dados esteja presente a seqüência FF_H ou 00_H alinhadas em byte e que após o erro ocorrer, as palavras sejam alteradas da mesma forma que o *Internet Checksum*. Então a P_{ue} do surto (8:1,0) = $(2*(1/2^8))/8 = 9,77x10^{-4}$; encontramos em nossa simulação o valor de $9,74x10^{-4}$, uma diferença de 0,3%.

No Fletcher16 255 também notamos a alteração da distância pela alteração do tamanho das mensagens. Como a distância mínima deste código é 3 para mensagens de até 2.040 bits, quando codificamos mensagens de 100 bytes todos os erros de 2 bits foram detectados, mas nas mensagens de 1.500 bytes houve uma alta taxa de falhas pela alteração de 2 bits ($2,01x10^{-4}$). No Fletcher32 65535 não verificamos esta situação, pois a alteração da distância só ocorre quando são utilizadas palavras código maiores que 524.280 bits, um valor superior aos geralmente encontrados em redes de computadores.

No Fletcher32 65535 o tamanho das mensagens influenciou na capacidade de detecção. Houve uma taxa 10 vezes maior de falhas de detecção quando utilizamos as mensagens de 100 bytes em relação às de 1.500 bytes.

Nos códigos que utilizam complemento a dois, a alteração do comprimento da palavra código foi em torno de 5% da taxa de falhas, já a aplicação dos diferentes tipos de dados não produziu alteração significativa nos resultados.

D. ADLER

Os resultados do Adler32 foram parecidos com os do Fletcher32 65535, ambos são susceptíveis ao comprimento das mensagens e ao tipo de dados empregados (40% de diferença entre o melhor e o pior caso). Em relação ao comprimento da mensagem o emprego de mensagens de 1.500 bytes no modelo de erros aleatórios resultou em taxas de falhas de 10 a 20 vezes melhores que o de mensagens de 100 bytes. No modelo de surto este valor foi de 20% a 80% melhor.

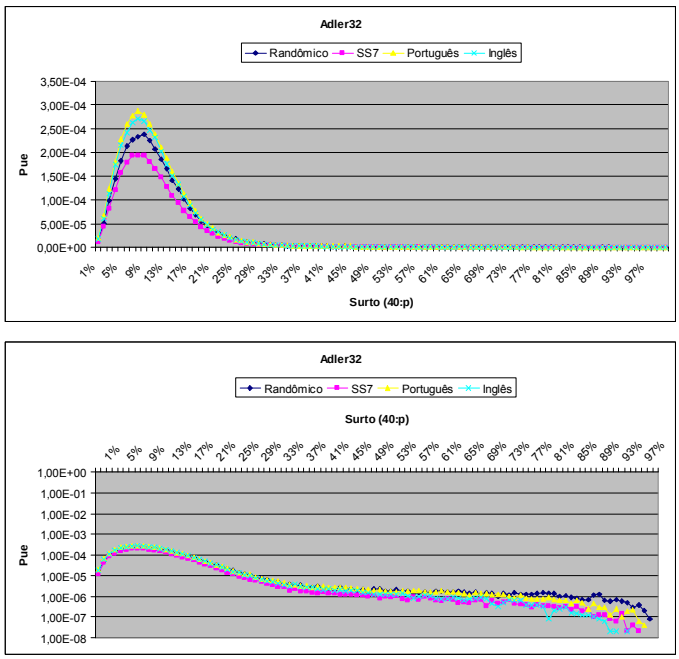

Figura 4 – Adler32; Surto (40:p), variando o tipo de dados e a probabilidade do surto (Comprimento da mensagem: 100 bytes). (a) Escala linear; (b) Escala logarítmica.

O padrão de erro que afeta 24 bits, citado por Sheinwald [14] resultou nas maiores taxas de falhas do Adler32, mesmo assim não chegou aos piores níveis do TCP32.

A P_{ue} do Adler32 ficou em torno de 10^{-5} a 10^{-6}, como se estivéssemos utilizando um código que usa de 14 a 19 bits redundantes. Desta forma encontramos as taxas de falhas de detecção deste código em unidades de transporte típicas (Figura 4).

VI. CONCLUSÕES

Neste trabalho foi testada a eficiência de códigos detectores de erro em situações próximas às encontradas em uma rede de computadores. É importante ressaltar que os protocolos de transporte são utilizados em conjunto com o protocolo de rede IP. Um pacote de dados trafegando pela Internet, pode passar por enlaces sem fio, fibras óticas e cabos de par trançado, além de roteadores e *middle-boxes* (NATs e *Proxies*) [17]. Dessa maneira, a modelagem de tal canal e de um padrão de erro para esta situação se torna difícil. Então o código que minimize a taxa de erro residual nesse caso poderia ser o que apresentasse bons resultados na maioria dos testes, como opção. As Figuras 5 e 6 apresentam as capacidades de detecção dos códigos estudados, pelo emprego de surtos (Dados: Randômicos; Comprimento da mensagem: 100 bytes).

Neste trabalho os CRCs obtiveram os melhores resultados de taxa de erro residual, portanto são ótimos candidatos a serem empregados na camada de transporte. Um dos problemas enfrentados pelos CRCs no entanto, é que em software sua codificação ainda é lenta comparada aos *checksums*.

O tipo de dado empregado em nossos testes somente influenciou os resultados dos códigos não lineares, como era de ser esperado. A maior taxa de falhas no *Internet Checksum* foi obtida com dados uniformemente distribuídos. Vale ressaltar que várias fontes citam que este tipo de dados levaria à taxas de falhas bem menores que as obtidas para dados não uniformemente distribuídos.

As simulações efetuadas confirmam o problema existente no Adler32 para pacotes em torno de 100 bytes e foi verificado que ele também possui uma taxa de erros elevada mesmo quando são utilizados pacotes de 1.500 bytes. Esses resultados mostram que este código não é aconselhável para a verificação da existência de erros em pacotes de redes de computadores.

AGRADECIMENTOS

Agradecemos ao Prof. Dr. Valdemar C. da Rocha Jr. pelas suas sugestões no desenvolvimento deste trabalho.

REFERÊNCIAS

[1] T. Baicheva, S. Dodunekov, P. Kazakov, *Undetected error probability performance of cyclic redundancy-check codes of 16-bit redundancy*, IEEE Proceedings on Communications, 147:253-256, October 2000.

[2] G. Castagnoli, S. Braeuer, M. Herrman, *Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits*, IEEE Trans.Comm., Vol. 41(6):883-892, June 1993.

[3] J. Fletcher, *An Arith. Checksum for Serial Transmissions*, IEEE Trans. Comm, Vol. 30(1):247-252, January 1982.

[4] G. Holzmann, *Design and Validation of Computers Protocols*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[5] E. Kohler, *et al.*, *Datagram Congestion Control Protocol (DCCP)*, Internet draft, draft-ietf-dccp-spec-11.txt, March 2005.

[6] S. Lin, D. Costello, *Error Control Coding: Fundamentals and Applications*, 2nd edition, Prentice Hall, 2004.

[7] L. Larzon, *et al.*, *The Lightweight User Datagram Protocol (UDP-Lite)*, Internet RFC 3828, July 2004.

[8] J. Mogul, *et al.*, *Path MTU Discovery*, Internet RFC 1191, November 1990.

[9] P. Deutsch, et al., ZLIB Compressed Data Format Specification version 3.3, Internet RFC 1950, May 1996.

[10] R. Stewart, *et al.*, *Stream Control Transmission Protocol*, Internet RFC 2960, October 2000.

[11] J. Postel, *et al.*, *User Datagram Protocol*, Internet RFC 768, August 1980.

[12] J. Postel, *Internet Protocol*, Internet RFC 791, Sept 1981.

[13] J. Postel, *Transmission Control Protocol*, Internet RFC 793, September 1981.

[14] D. Sheinwald.*On Fletcher and Adler codes, and classic CRCs*, private communication, January 2001.

[15] K. Sklower, *Improving the Efficiency of the OSI Checksum Calculation*, ACM Computer Communication Review, Vol. 19, No. 5, pp. 32-43, October 1989.

[16] W. Stallings, *Computer Organization and Architecture*, 5ª Ed. Prentice Hall, New Jersey, 1999.

[17] J. Stone, *et al.*, *Stream Control Transmission Protocol (SCTP) Checksum Change*, Internet RFC 3309 Sept. 2002.

[18] J. Stone, et al., *Performance of checksums and CRCs over real data*, IEEE/ACM Transactions on Networking, Vol. 6, N° 5 , pp. 529-543. October, 1998.

[19] A. Tanenbaum, *Computer Networks*, 4ª ed., Addison Wesley, 2003.

[20] S. Wicker, Error Control Systems for Digital Communication and Storage, Prentice Hall, 1995.

[21] J. Wolf, D. Chun, *The Single Burst Error Detection Performance of Binary Cyclic Codes*, IEEE Transactions on Communications, Vol. 42, n° 1, pp. 11-13, January 1994.

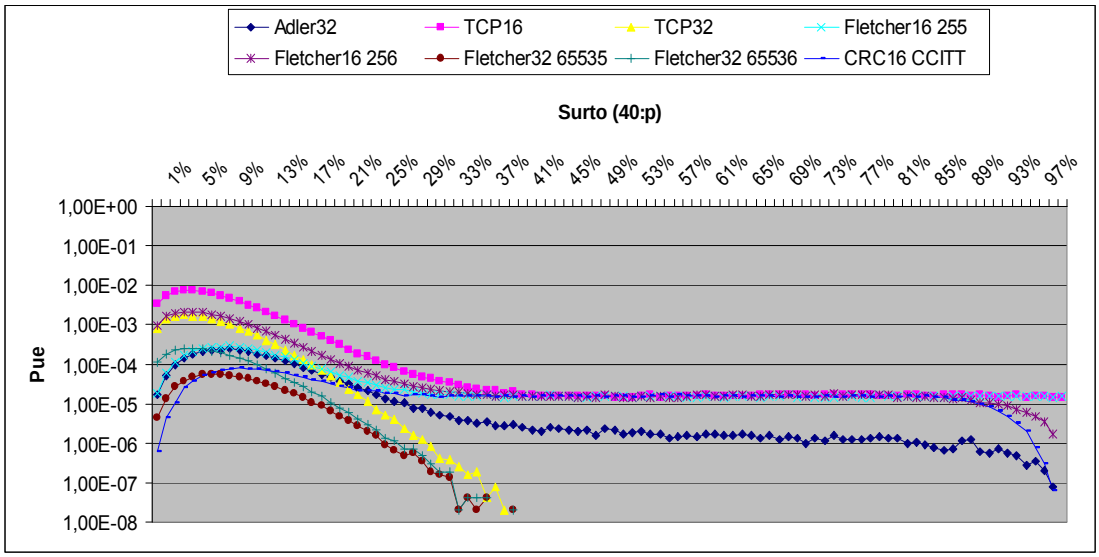

Figura 5 – Comparação das capacidades de detecção dos códigos estudados, pelo emprego de surto(40:*p*).
(Dados: Randômicos; Comprimento da mensagem: 100 bytes)

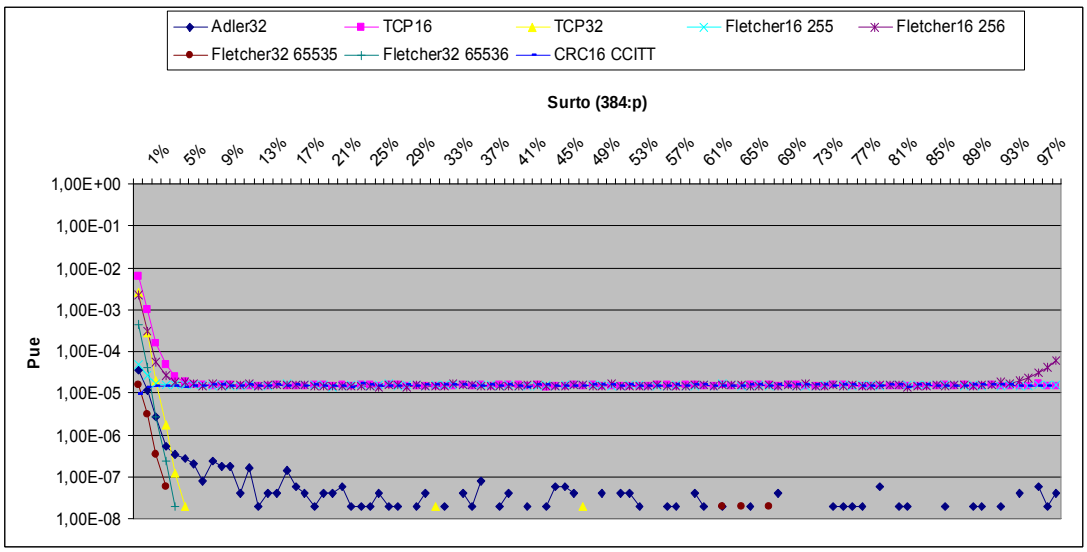

Figura 6 – Verificação do comportamento dos códigos estudados em surtos de grande comprimento; surto (384:*p*).
(Dados: Randômicos; Comprimento da mensagem: 100 bytes)