

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
INSTITUTO DE INFORMÁTICA  
ENGENHARIA DE COMPUTAÇÃO

GERMANO VEIT MICHEL

**Estudo de Mecanismos FEC para  
Transmissão Confiável em UDP**

Prof. Dr. Sérgio Luis Cechin  
Orientador

Porto Alegre, Junho de 2010

## **AGRADECIMENTOS**

Em primeiro lugar agradeço aos meus pais, pelo apoio e pela incondicional torcida com a qual sempre contei em todas as etapas da minha vida.

Agradeço, de forma muito especial, ao professor Dr. Sérgio Luis Cechin por ter me proporcionado a oportunidade de desenvolver este trabalho, sob sua orientação e por oferecer, durante este período, todo o apoio necessário para seu desenvolvimento.

Dirijo, igualmente, meu reconhecimento e agradecimento à Universidade Federal do Rio Grande do Sul, especificamente ao Instituto de Informática, por oferecer um curso de qualidade inquestionável e todos os equipamentos, condições e materiais necessários, sempre.

Não poderia deixar de agradecer aos meus amigos e colegas que contribuíram para realização não só deste trabalho mas também para o meu curso de graduação.

# SUMÁRIO

<b>LISTA DE ABREVIATURAS E SIGLAS</b> . . . . .	5
<b>LISTA DE FIGURAS</b> . . . . .	6
<b>LISTA DE TABELAS</b> . . . . .	7
<b>RESUMO</b> . . . . .	8
<b>ABSTRACT</b> . . . . .	9
<b>1 INTRODUÇÃO</b> . . . . .	10
1.1 Contextualização . . . . .	10
1.2 Objetivos . . . . .	11
<b>2 PESQUISA BIBLIOGRÁFICA</b> . . . . .	12
2.1 Sistemas Digitais de Comunicação . . . . .	12
2.2 O Canal . . . . .	13
2.3 Modelo MR-OSI . . . . .	13
2.4 Protocolo UDP . . . . .	14
2.5 Protocolo TCP . . . . .	15
2.6 Automatic Repeat Request . . . . .	15
2.7 Forward Error Correction . . . . .	16
2.7.1 Redundância Tripla . . . . .	17
2.7.2 Código de Hamming . . . . .	17
2.7.3 Códigos Cíclicos . . . . .	19
2.7.4 Códigos Convolucionais . . . . .	19
2.7.5 Códigos Turbo . . . . .	20
2.7.6 Códigos LDPC . . . . .	20
<b>3 TRABALHOS RELACIONADOS</b> . . . . .	21
3.1 FEC em streaming de Vídeo . . . . .	21
3.2 FEC Adaptativo para UDP . . . . .	21
3.3 FEC vs. ACC para VoIP em redes congestionadas . . . . .	22
3.4 FEC na camada de Rede . . . . .	22
3.5 Comportamento do TCP com perda de quadros . . . . .	22
3.6 Considerações sobre os Trabalhos Relacionados . . . . .	22

<b>4</b>	<b>IMPLEMENTAÇÃO</b>	24
4.1	Visão Geral	24
4.2	Codificador e Decodificador FEC	24
4.3	Software de Gerenciamento dos Testes	25
4.4	Injetor de Falhas	26
<b>5</b>	<b>ESPECIFICAÇÃO DOS TESTES</b>	27
5.1	Descrição	27
5.2	Transmissão sem erros, BCH e Reed-Solomon	27
5.3	Transmissão com erros, BCH e Reed-Solomon	28
5.4	Transmissão com erros, TCP e UDP com Reed-Solomon	29
5.5	Faultlet para o Firmament	30
<b>6</b>	<b>EXECUÇÃO E RESULTADOS DOS TESTES</b>	32
6.1	Transmissão sem erros, BCH e Reed-Solomon	32
6.1.1	Detalhamento da Execução	32
6.1.2	Resultados	33
6.2	Transmissão com erros, BCH e Reed-Solomon	35
6.2.1	Detalhamento da Execução	35
6.2.2	Resultados	36
6.3	Transmissão com erros, TCP e UDP com Reed-Solomon	36
6.3.1	Detalhamento da Execução	36
6.3.2	Resultados	37
6.4	Resumo dos Resultados	37
6.5	Análise dos Resultados	40
6.5.1	FEC para Multicast	41
<b>7</b>	<b>CONCLUSÃO</b>	44
	<b>REFERÊNCIAS</b>	45

## LISTA DE ABREVIATURAS E SIGLAS

3G	Família de Padrões para Telecomunicação Móvel
ACC	Frame Accumulator
ARPA	Advanced Research Projects Agency
ARQ	Automatic Repeat Request
BCH	Bose e Ray-Chaudhuri (Códigos)
CD	Compact Disc
CRC	Cyclic Redundancy Check
DSL	Digital Subscriber Line
DVB	Digital Video Broadcasting
DVB-S2	Digital Video Broadcasting Second Generation
DVD	Digital Video Disc
FEC	Forward Error Correction
H.264	Padrão de compressão de vídeo do ITU-T
IP	Internet Protocol
ITU	International Telecommunication Union
LDPC	Low-Density Parity Check
MR-OSI	Open System Interconnection Reference Model
RAID	Redundant Array of Independent Discs
RoHC	Robust Header Compression
RSC	Recursive Systematic Convolutional Code
RTT	Round Trip Time
SEC	Single Error Correction
SMS	Short Message Service
SNR	Signal to Noise Ratio
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator

## LISTA DE FIGURAS

Figura 2.1:	Sistema Digital de Comunicação Típico . . . . .	12
Figura 2.2:	Codificação para Controle de Erros no Emissor . . . . .	13
Figura 2.3:	Diagrama do Modelo MR-OSI . . . . .	14
Figura 2.4:	O Formato do Datagrama UDP . . . . .	15
Figura 2.5:	O Formato do Pacote TCP . . . . .	16
Figura 2.6:	Sistema Digital de Comunicação com FEC . . . . .	17
Figura 2.7:	Redundância Tripla . . . . .	18
Figura 2.8:	Código de Hamming: Inserção dos Bits de Paridade . . . . .	18
Figura 4.1:	Transmissão com FEC . . . . .	24
Figura 4.2:	Quadro do Protocolo UDP . . . . .	25
Figura 4.3:	Quadro do Protocolo UDP, codificado com FEC na área de mensagem	25
Figura 4.4:	Esquema das Funcionalidades do Software de Gerenciamento de Testes	25
Figura 4.5:	Esquema de Avaliação da Implementação . . . . .	26
Figura 6.1:	Detalhes da Execução do Teste - Sem Erros na Rede . . . . .	33
Figura 6.2:	Tempo de Codificação na Ausência de Erro . . . . .	33
Figura 6.3:	Tempo de Decodificação na Ausência de Erro . . . . .	34
Figura 6.4:	Tempo Total na Ausência de Erro . . . . .	34
Figura 6.5:	Detalhes da Execução do Teste - Com Erros na Rede . . . . .	35
Figura 6.6:	Tempo de Decodificação na Presença de Erro . . . . .	36
Figura 6.7:	Detalhes da Execução do Teste com TCP e UDP+FEC . . . . .	37
Figura 6.8:	TCP vs FEC em Redes com Corrupção de Pacotes . . . . .	40
Figura 6.9:	Comportamento do TCP com Erros no Canal . . . . .	41
Figura 6.10:	Visão de um Sistema Multicast . . . . .	41
Figura 6.11:	Previsão de Custo Computacional para Multicast na Ausência de Erros	43

## LISTA DE TABELAS

Tabela 6.1:	BCH e Reed-Solomon - Tempo de Codificação . . . . .	38
Tabela 6.2:	BCH e Reed-Solomon - Tempo de Decodificação sem Erros . . . . .	38
Tabela 6.3:	BCH e Reed-Solomon - Tempo Total de Transmissão . . . . .	38
Tabela 6.4:	BCH e Reed-Solomon - Tempo de Decodificação com Erros . . . . .	39
Tabela 6.5:	Transmissão com Erros, TCP vs. UDP com Reed-Solomon . . . . .	39

## RESUMO

Este trabalho apresenta um estudo de mecanismos FEC para transmissão de dados sobre o protocolo UDP. Uma visão geral dos algoritmos de codificação FEC existentes é dada, expondo suas vantagens e limitações. Duas destas codificações foram comparadas através de testes reais, no quesito desempenho. Também é feita uma comparação do UDP/FEC com o protocolo confiável TCP. Então apresenta-se uma breve previsão do comportamento de codificações FEC para uso em Multicast. Por fim, chega-se a novas ideias sobre implementar FEC para uso geral em larga escala, com sugestões para solucionar os problemas descobertos e motivação para trabalhos futuros.

**Palavras-chave:** Mecanismos de Transmissão em Rede, FEC, ARQ, UDP, TCP, Redundância, Falhas, Recuperação.



## **A Study of FEC Mechanisms for Reliable Transmission over UDP**

### **ABSTRACT**

This work presents a study of FEC mechanisms for transmitting data over the UDP protocol. An overview of existing FEC encoding algorithms is given, as well as their strong points and limitations. Two of these encodings were compared by testing their actual performance. Also, a comparison of UDP/FEC to the TCP protocol is made. Then, a brief FEC coding behavior for use in multicast prediction is presented. Finally, new ideas for wide deployment of FEC for network communications are presented, with suggestions on how to solve the problems discovered in this work. Ideas for future work are also given.

**Keywords:**

# 1 INTRODUÇÃO

## 1.1 Contextualização

Uma rede de computadores pode ser definida como um grupo interconectado de computadores. Estas redes estão presentes em grande parte das atividades desenvolvidas pelo ser humano. Desde os estudos iniciais da ARPA (Advanced Research Projects Agency) sobre a ARPANET (Advanced Research Projects Agency Network), no início da década de 1960, elas vem ganhando popularidade, novos recursos e, junto com estes últimos, trazem novos desafios para serem resolvidos.

Seja no escritório, para compartilhar uma planilha de dados com o colega de trabalho, em casa para ver um e-mail ou mesmo assistir TV, enviar uma mensagem SMS (Short Message Service) para um amigo. Até na indústria para coordenar os elementos de produção, as redes de computadores não são apenas essenciais para o desenvolvimento das mais diversas atividades, mas ganham importância a cada dia num mundo que tende cada vez estar mais interconectado.

É impossível falar de redes de computadores sem mencionar protocolos e meios de comunicação. Meios de comunicação são os meios usados para transmitir informações de um transmissor para um receptor. Já protocolos de comunicação são o conjunto de regras para a representação de dados, sinalização, autenticação e detecção de erros necessários para que se troque informações num canal de comunicação. O MR-OSI (Open System Interconnection Reference Model) é uma descrição abstrata para o projeto de protocolos de rede que atuam sobre um canal de comunicação.

É neste MR-OSI que as redes de computadores são baseadas. Ele divide a arquitetura de uma rede em sete camadas de modo que, a princípio, todas camadas são independentes e podem ser modificadas para atenderem novos requisitos de funcionalidade ou parâmetros de desempenho.

As funcionalidades requeridas e as demandas sobre as redes de computadores crescem exponencialmente. As cargas das redes de telecomunicações estão sempre altas, mesmo com os constantes investimentos para fornecer melhores serviços e atender a demanda. A tendência é que cada vez se troque mais informações nessas redes. Com a inclusão digital, mais pessoas terão acesso a Internet, por exemplo. Portanto, neste contexto de rápidas mudanças, é conveniente buscar soluções alternativas para os problemas existentes. Seja por meio de novas ideias ou reavaliação de ideias que não faziam muito sentido no passado.

FEC (Forward Error Correction) é um sistema de controle de erros para a transmissão de dados. Para funcionar, o transmissor adiciona dados redundantes nas mensagens. Sua principal vantagem é que se pode recuperar dados corrompidos ou perdidos sem a necessidade de retransmissão. Em casos onde a retransmissão é muito custosa, este paradigma

é bastante interessante.

O UDP (User Datagram Protocol) é um protocolo que dá a camada de aplicação o serviço de entrega não garantido de datagramas. Seu uso é adequado para sistemas que não suportam o sobrecusto de usar TCP (Transmission Control Protocol) ou que admitem algum tipo de problema na entrega das mensagens. Podemos ver a utilização de FEC sobre UDP como um jeito de melhorar a transmissão no sentido de suportar a corrupção de alguns bytes.

## 1.2 Objetivos

O objetivo deste projeto é estudar e testar os mecanismos FEC, como *payload* do protocolo de rede UDP, numa arquitetura cliente-servidor. Traçar os perfis dos algoritmos de codificação do ponto de visto de desempenho e estabelecer uma comparação, em termos de taxas de transmissão, com protocolos mais confiáveis, como TCP. Ou seja, investigar se FEC é uma alternativa viável para o uso na prática.

## 2 PESQUISA BIBLIOGRÁFICA

### 2.1 Sistemas Digitais de Comunicação

Sistemas digitais de comunicação transmitem informações de um lado para o outro. Antes de mais nada, é importante definir o que é informação. Segundo o dicionário Oxford, informação é o conhecimento comunicado sobre algum fato, sujeito ou evento particular, para alguém. É conveniente poder medir a quantidade de informação: para uma informação sem redundâncias, sua quantidade de informação  $H$ , em bits, está vagamente ligada ao inverso da probabilidade  $P$  desta informação ser escolhida dentro de um conjunto de informações (SWEENEY, 2002, pág 3). Isto é mostrado na equação abaixo.

$$H = \log_2(1/p)$$

Um sistema digital de comunicação típico é mostrado na figura 2.1. O objetivo de um sistema de comunicação é fazer com que a informação saia da origem, através do canal e chegue no destino sem ser comprometida. Uma transmissão com sucesso depende de quão precisamente o receptor pode determinar o sinal transmitido pelo emissor. Quando o sinal determinado pelo receptor é exatamente igual ao transmitido pelo emissor, podemos dizer que o canal usado é livre de ruído (sinais elétricos não desejados que acompanham a mensagem). Infelizmente, este não é o caso. Sistemas de comunicação reais precisam conviver com a presença do ruído. Shannon mostrou em 1949 (SHANNON, 1949) que os canais têm como característica uma capacidade máxima. Esta capacidade máxima limita a quantidade de informação que se pode transmitir sem que haja o comprometimento da integridade desta informação, pelo ruído.

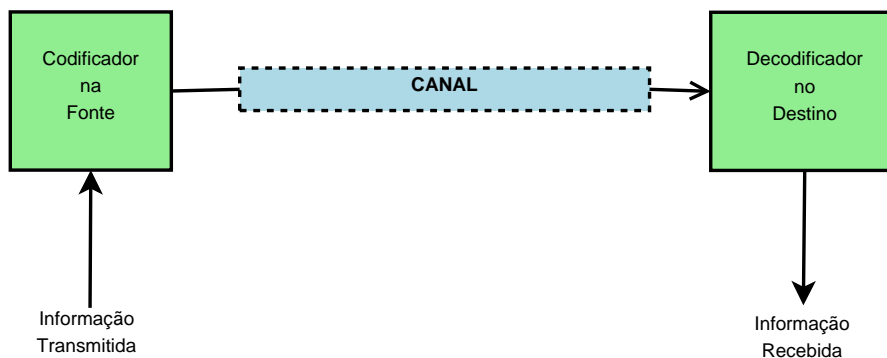


Figura 2.1: Sistema Digital de Comunicação Típico

## 2.2 O Canal

O meio de transmissão introduz incertezas sobre a integridade da informação transportada. Podemos descrever os tipos de canais de acordo com as probabilidades de erros que eles introduzem (SWEENEY, 2002, pág 7).

- Canal sem Memória: a probabilidade de erro é independente de um símbolo para o próximo.
- Canal Simétrico: a probabilidade de o símbolo  $i$  ser recebido como  $j$  é a mesma de o  $j$  ser recebido como  $i$ .
- Canal com Ruído Gaussiano Aditivo Branco: um canal sem memória que o sinal sofre a adição do ruído banda-larga cuja amplitude é dada pela distribuição normal de Gauss.
- Canal com Erros em Rajada: os erros acontecem em rajadas, ou seja, existem períodos de poucos erros e períodos com taxa de erros muito elevada.

Na prática, os canais reais são combinações destes tipos de canais apresentados. Como consequência de existirem diferentes tipos de canais, é razoável se esperar que também existam diferentes maneiras de se lidar com esses problemas, algumas bastante adequadas para um determinado tipo de canal, outras nem tanto.

Existem, felizmente, duas formas básicas de se evitar que uma informação seja perdida durante o processo de comunicação: ARQ (Automatic Repeat Request) e FEC. Ambas são técnicas de codificação digitais, mais especificamente, são técnicas de codificação para controle de erros. A figura 2.2 mostra um diagrama, no transmissor, sobre onde deve ser feita esta codificação. Numa visão mais global do processo, o diagrama da figura 2.2 se insere no bloco "Codificador na Fonte" da figura 2.1. É evidente que um processo semelhante se faz necessário no receptor.

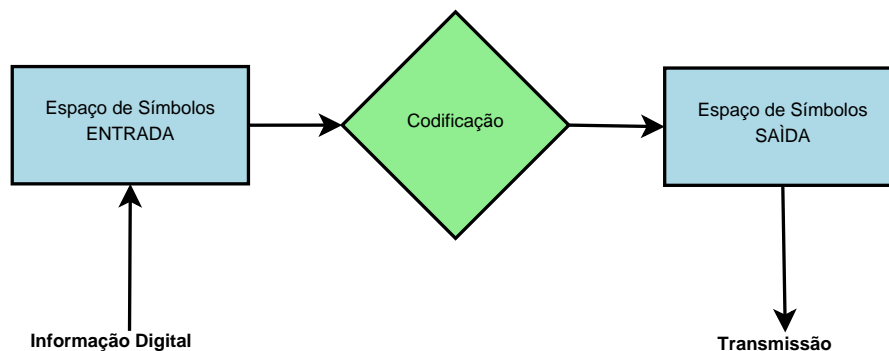


Figura 2.2: Codificação para Controle de Erros no Emissor

## 2.3 Modelo MR-OSI

Para um melhor entendimento dos conceitos envolvidos neste trabalho, é interessante uma breve explicação do modelo MR-OSI, do nível mais alto (abstrato) para o mais baixo. Um diagrama do modelo MR-OSI pode ser visto na figura 2.3.

A camada de aplicação converte a representação da informação para um formato universal, facilitando as comunicações entre aplicações que estão em ambientes diferentes.

As camadas de apresentação e sessão serão ignoradas neste momento. Geralmente não são explicitamente implementadas no mundo real e suas funcionalidades se distribuem nas camadas vizinhas.

A camada de transporte adiciona serviços fim-a-fim. Implementar, se for o caso, mecanismos de controle de erros, de fluxo e de sequência. Estes mecanismos devem ser suficientes para que as camadas superiores não tenham de se preocupar com a disponibilidade e confiabilidade das comunicações. Nos casos dos protocolos TCP e UDP, esta camada oferece serviços, respectivamente, com e sem confiabilidade.

A camada de rede ocupa-se da transferência de dados *host-to-host*, uma vez que trata as transferências de dados desde o emissor até ao receptor final. No caso do protocolo IP, utilizado neste trabalho, ela limita-se a detectar a ocorrência de erros, sem tratamento algum.

A camada de enlace garante a comunicação entre pontos de uma rede que estão diretamente ligados entre si e usam o mesmo tipo de camada física.

Na camada física, os dados provenientes do camada de enlace, na sua forma mais elementar (bits) são transformados em sinais adequados ao meio responsável pela propagação do mesmo desde o emissor até receptor.

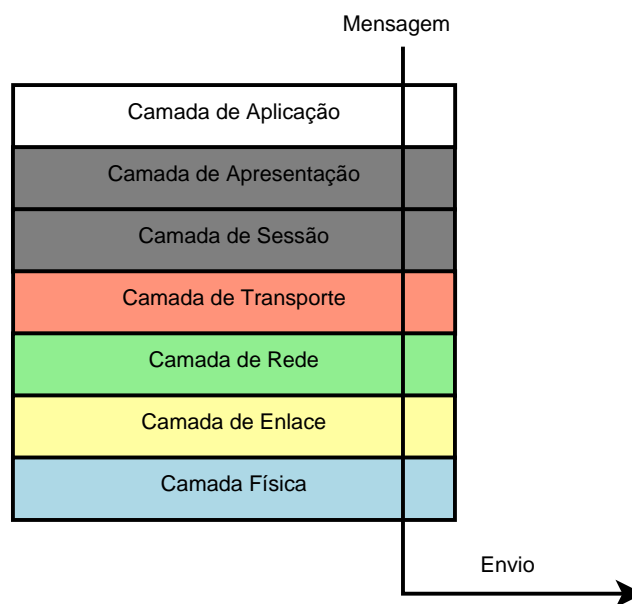


Figura 2.3: Diagrama do Modelo MR-OSI

## 2.4 Protocolo UDP

Este protocolo é extremamente simples e amplamente utilizado na Internet. Não é orientado a conexão, não tem garantia de entrega de mensagens nem controle de duplicações e também não garante o ordenamento (CARISSIMI, 2009, pág 287).

O formato do datagrama UDP é apresentado na figura 2.4. O campo porta de origem, de 16 bits, indica a partir de que porta, no emissor, que a mensagem foi enviada. Estes valores podem variar de 0 a 65535. O campo porta de destino é similar, mas indica a porta que a mensagem deve ser entregue. O campo tamanho do datagrama é calculado sobre os dados da mensagem e o cabeçalho UDP. O checksum é calculado sobre o cabeçalho UDP e os dados da mensagem justaposto ao pseudo-cabeçalho imaginário composto pelos

dados do cabeçalho do protocolo imediatamente abaixo, no caso o IP. Neste sentido, nota-se que a independência das camadas do modelo MR-OSI foi quebrada.

É necessária uma observação especial sobre o campo checksum. Se o receptor conferir o checksum e este não conferir com o datagrama recebido, este será descartado silenciosamente. Isso impede que a implementação de FEC na camada de aplicação funcione para corrupção de dados dentro do pacote. Felizmente, para os objetivos deste trabalho, este checksum é opcional, fazendo com que pacotes corrompidos por ruídos sejam entregues para a aplicação, onde se dará a recuperação das informações corrompidas, através do mecanismo de decodificação FEC. Para se desabilitar o uso do checksum pode-se utilizar uma opção específica do *socket* do receptor ou, no emissor, enviar todos os bits deste campo em zero.

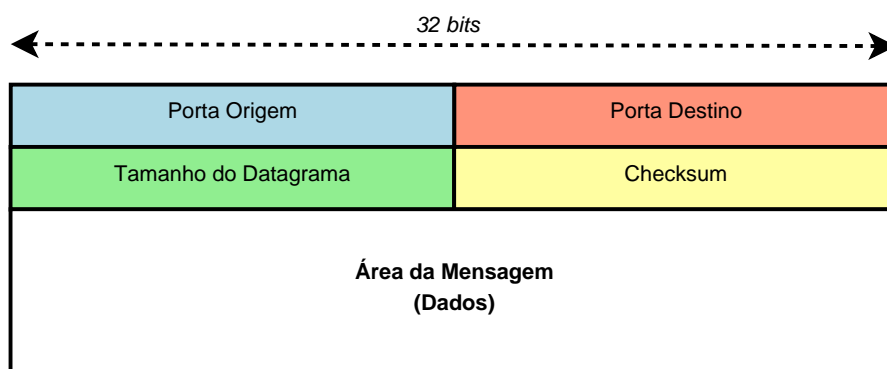


Figura 2.4: O Formato do Datagrama UDP

## 2.5 Protocolo TCP

Este protocolo é bastante sofisticado. É orientado a conexão, tem garantia de entrega de mensagens, controle de duplicações e também garante o ordenamento (CARISSIMI, 2009, pág 291).

O formato do pacote TCP é apresentado na figura 2.5. Como se pode ver, existem mais campos do que no datagrama UDP. Todos estes campos extras são necessários para adicionar as funcionalidades relacionadas a confiabilidade do protocolo TCP. Além de ser maior, o pacote TCP requer mais processamento que o UDP, já que todas estas informações precisam ser processadas.

## 2.6 Automatic Repeat Request

ARQ, sigla em inglês, significa Pedido Automático de Repetição. Como o nome já diz, o receptor, ao verificar que a mensagem recebida não está contida no conjunto das mensagens esperadas, pede a sua retransmissão (COSTELLO, 1983, pág 458). Evidentemente, é necessário implementar mecanismos para a verificação das informações recebidas. Um método para esta verificação da correção da mensagem bastante utilizado é o CRC. Deixa-se claro que CRC não é um método de ARQ, mas apenas um mecanismo de se detectar se os dados recebidos estão íntegros.

Vale ressaltar que tanto o transmissor quanto o receptor devem ser modificados com o intuito de suportar ARQ. Para o ARQ, na figura 2.2, o bloco de codificação apenas adiciona informações para que, no destino, seja possível verificar se a mensagem não foi

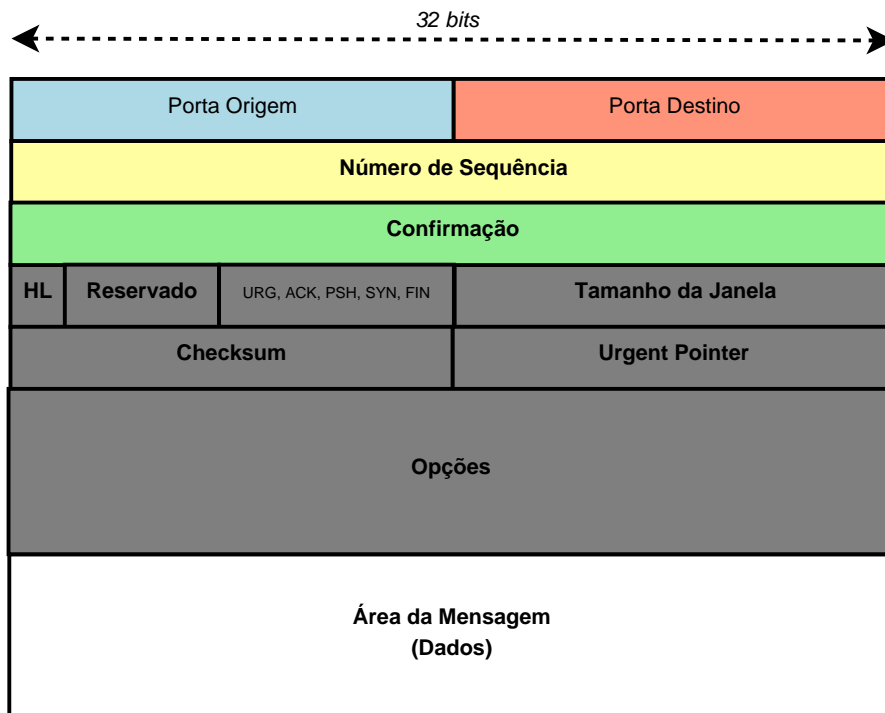


Figura 2.5: O Formato do Pacote TCP

corrompida durante a transmissão.

Por mais interessante que seja, o estudo desta modalidade não é o foco principal deste trabalho. Entretanto, será interessante comparar o protocolo TCP, que usa um esquema de ARQ, ao UDP instrumentado com FEC.

## 2.7 Forward Error Correction

Ao contrário de ARQ, quando uma mensagem é julgada corrompida pelo ruído, não é pedida a retransmissão, por opção ou por necessidade. Em um sistema de comunicação com FEC, o transmissor adiciona redundância à mensagem, bits adicionais, para que, no caso desta mensagem ser corrompida por ruído, a mesma possa ser recuperada pelo receptor (MOREIRA, FARRELL, pág 65). Aqui, não existe a possibilidade do receptor pedir a retransmissão da mensagem, embora existam mecanismos de controle de erros híbridos, que usam ARQ e FEC concomitantemente. É importante ressaltar que a quantidade de informação transmitida é a mesma da mensagem original, pois a probabilidade desta mensagem ser selecionada para transmissão continua a mesma (SWEENEY, 2002, pág 3). Portanto é fácil inferir que, utilizando FEC, a eficiência da transmissão diminui. Isso se deve ao fato de ser necessário adicionar bits de redundância, não apenas para verificação de integridade mas também para recuperação de erros. Estes aumentam o tamanho da mensagem, mas não aumentam a quantidade de informação transportada. Um diagrama para ilustrar as diferenças de um sistema de comunicação com FEC para um sistema de comunicação típico é mostrado na figura 2.6. Para o FEC, na figura 2.2, o bloco de codificação adiciona mais informações redundantes do que o ARQ, ao se incluir, também, redundância suficiente para que possam ser recuperadas partes da mensagem corrompidas por ruído.

Uma desvantagem do FEC (detecção e correção) puro em relação ao ARQ (detecção



e retransmissão) é que, caso a mensagem seja perdida ou contenha mais erros que a capacidade de correção da codificação FEC empregada, perdem-se completamente os dados transmitidos.

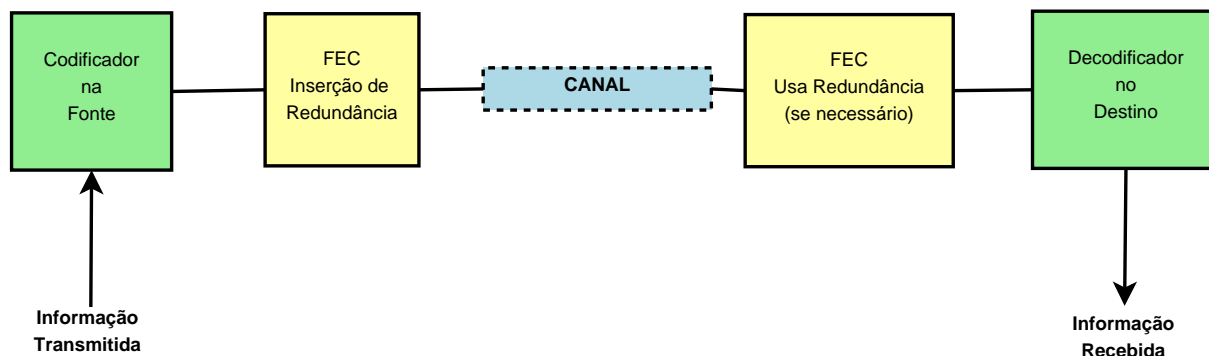


Figura 2.6: Sistema Digital de Comunicação com FEC

A maioria dos códigos que servem para realizar FEC se encaixam em sete categorias. Códigos de bloco, códigos cíclicos, códigos BCH (Bose e Ray-Chaudhuri), códigos Reed-Solomon, códigos turbo, códigos LDPC (Low Density Parity Check) e códigos de convolução.

Agora será dada uma visão geral de codificações que servem para realizar FEC. Ou seja, métodos para codificar redundância nas mensagens de forma que erros possam ser recuperados no destino da mensagem. Foram utilizados neste trabalho apenas os códigos BCH e Reed-Solomon.

### 2.7.1 Redundância Tripla

A redundância tripla (MOREIRA, FARRELL, pág 42) é extremamente simples de entender e fácil de implementar. A figura 2.7 mostra como funciona esta técnica. A mensagem foi repetida três vezes. É aconselhável, na maioria dos casos, concatenar a mensagem inteira três vezes ao invés de repetir, sucessivamente, cada bit três vezes pelo fato que ruídos mais longos podem alterar a mesma posição de bit em mais de uma réplica, mudando o resultado da decodificação.

Esta técnica consiste em transmitir três vezes cada bit, concatenando três vezes a mensagem. No receptor, a mensagem é recebida normalmente. Então é separada em três partes (as três réplicas da mensagem original). Estas réplicas são comparadas bit a bit e a mensagem final é montada por votação entre os três bits das três mensagens replicadas, um bit de cada mensagem. A figura 2.7 também mostra a decodificação.

Não é difícil notar que este código não é muito eficiente. De fato, aumentando três vezes o tamanho da mensagem, aumentamos proporcionalmente a probabilidade de que um erro ocorra na transmissão desta mensagem. Outra limitação grave é o fato da informação ser interpretada como correta do lado do receptor ao se inverter um bit de mesma posição em duas das réplicas, estando ela, neste caso, errada. Por estas características, a redundância tripla foi deixada de fora dos testes realizados.

### 2.7.2 Código de Hamming

O código de Hamming é um dos mais conhecidos. Talvez pelo fato de ser relativamente eficiente e bastante simples (MOREIRA, FARRELL, pág 64). É bastante usado em memórias. Existem diversas variações dele, aqui será explicado o mais usado de-

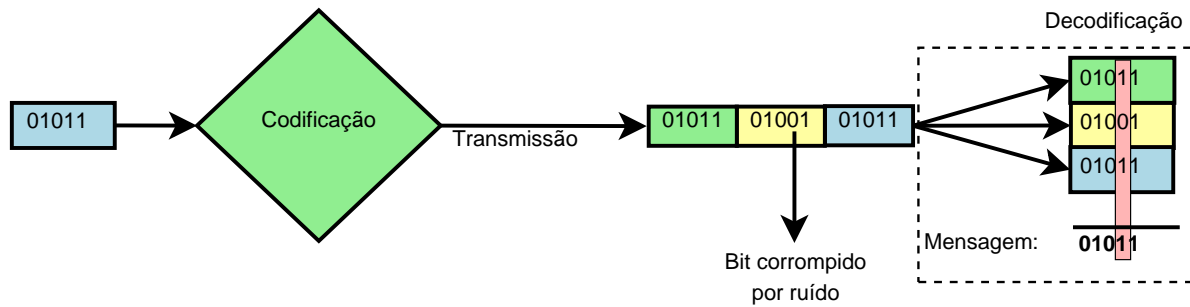


Figura 2.7: Redundância Tripla

les (SEC). Esta variação é capaz de corrigir um bit errado em cada bloco, não impondo restrições sobre o tamanho deste bloco.

Codificar uma sequência de bits com o código de Hamming é relativamente simples: são inseridos bits de paridade em posições pré-determinadas. Estas posições são as de potência de 2, ou seja, 1, 2, 4, 8, 16, 32... (HUFFMANN, 2003, pág 29). A figura 2.8 ilustra esta inserção para um bloco de 8 bits.

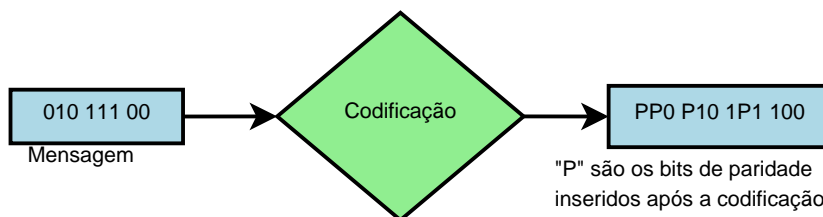


Figura 2.8: Código de Hamming: Inserção dos Bits de Paridade

Feito isso, temos um novo bloco de 12 bits. Mas estes bits de "P" devem assumir valores. Isto também é feito de uma forma simples. Cada bit "P" de paridade receberá o resultado do cálculo de paridade cobrindo os bits que a função binária E entre a posição do bit de dado e a posição do bit de paridade é não-nula, numerando-os da esquerda para direita, de forma crescente e começando em um. Por exemplo:

- Bit de Paridade na posição 1: Cálculo da paridade dos bits 1,3,5,7,9
- Bit de Paridade na posição 2: Cálculo da paridade dos bits 2,3,6,7,10,11
- Bit de Paridade na posição 4: Cálculo da paridade dos bits 4,5,6,7,12
- Bit de Paridade na posição 8: Cálculo da paridade dos bits 8,9,10,11,12

Note que cada bit de paridade está incluído no cálculo de apenas um bit de paridade, ele mesmo. Isso é de extrema importância e será explicado mais adiante.

Definido os valores de cada bit "P", a mensagem já está codificada pelo código de Hamming. Vamos agora analisar o que acontece no receptor: ao receber o bloco de 12 bits, o receptor já deve saber que se tratam, na verdade, de 8 bits de dados e 4 de paridade, assim como sabe, da mesma forma que o transmissor, em que posição estão os bits de dados e os de paridade. O decodificar deve calcular as paridades novamente e então compará-las com os bits "P" recebidos. Caso sejam idênticos, as chances são de que a mensagem não sofreu interferências. Caso sejam diferentes a soma das posições dos bits de paridade nos mostram a posição do bit errado, bastando invertê-lo para corrigir este

erro. Quando apenas um bit de paridade é diferente do calculado, o mesmo algoritmo aponta para que este bit de paridade, na mensagem recebida, esteja errado. Este importante fato é consequência de que cada bit de paridade está incluído apenas no cálculo da paridade dele mesmo, como dito anteriormente.

Entre as limitações desta codificação, na variação apresentada, se encontra a incapacidade de corrigir erros duplos (dois bits alterados durante a transmissão) ou maiores.

Estes códigos também foram excluídos dos testes realizados neste trabalho pelo fato de suas generalizações (BCH) estarem presentes.

### 2.7.3 Códigos Cíclicos

Uma característica importantíssima destes códigos é sua fácil implementação em hardware, usando lógica sequencial e *shift registers* (WICKER, 1994, pág 100). Portanto, são bastante eficientes, populares e estudados. Uma definição de códigos cíclicos é dada a seguir.

Seja  $C$  um código linear, dentro de um conjunto finito  $A$ , de tamanho de bloco  $n$ .  $C$  é um código cíclico se, para cada sequência de símbolos  $c = (c_1 \dots c_{n-1}, c_n)$  de  $C$ , a sequência de símbolos rodada à direita  $c = (c_n \dots c_1, c_n)$  também é uma sequência de símbolos de  $C$  (HUFFMANN, 2003, pág 121).

#### 2.7.3.1 Códigos BCH

Os códigos BCH são códigos cíclicos multi-nível, uma generalização dos códigos Reed-Müller e BCH (CARRASCO, 2008, pág 84).

Foram desenvolvidos em 1959 por Hocquenghem, e desde lá são objeto de muita atenção. Um dos seus pontos altos é a facilidade de decodificação, chamada de *syndrome decoding*. Também é uma classe caracterizada por ser bastante flexível, podendo-se definir o limiar de erros e o tamanho do bloco. Isso significa que é bastante interessante para projetos customizados.

São complicados de se implementar e pesados computacionalmente.

Estes códigos foram incluídos nos testes deste trabalho.

#### 2.7.3.2 Códigos Reed-Solomon

Este código de correção de erros funciona sobre-amostrando um polinômio construído a partir dos dados de entrada. Este polinômio é amostrado em vários pontos, e estas amostras são transmitidas. Existe um número mínimo de pontos que devem ser enviados para que o receptor possa recompor o polinômio, recuperando os dados. Se enviarmos mais pontos, estamos adicionando redundância, ponto de interesse deste trabalho (MOREIRA, FARRELL, pág 65).

São também códigos de tamanho de bloco fixo. Numa aplicação típica, se codificam 223 símbolos de 8 bits em 255 símbolos de saída. A capacidade de recuperação de erros neste caso é de 16 símbolos.

Códigos Reed-Solomon são usados em diversas aplicações comerciais: CDs, DVDs, Blu-ray, DSL, DVB e RAID 6, só para citar alguns.

Por serem muito expressivos, estes códigos foram incluídos nos testes deste trabalho.

### 2.7.4 Códigos Convolucionais

Este tipo de código difere dos códigos de bloco no sentido que o primeiro possui memória, o processo de codificação é dito contínuo. Ou seja, as sequências codificadas

para transmissão dependem não apenas dos dados a serem codificados, mas também da ordem que estes dados chegam ao codificador (COSTELLO, 1983, pág 287).

Estes códigos são bastante usados para transmissões tanto via rádio quanto cabeadas. Na década de 1970, estes códigos foram usados para transmissões espaciais e de satélites.

Estes códigos deram origem aos Códigos Turbo, que estão substituindo os Convolucionais em diversas aplicações.

Estes códigos, por terem como característica um processo de codificação contínuo, não são muito apropriados para serem aplicados nos testes pretendidos deste trabalho. Embora seja tecnicamente possível, os resultados não seriam completamente justos, diminuindo a confiabilidade dos resultados. Portanto, foram excluídos dos testes.

### **2.7.5 Códigos Turbo**

Relativamente recentes, de 1993, os códigos Turbo consistem na concatenação, em paralelo, de dois códigos de convolução. Eles atingem excelente desempenho, próximo ao limite de Shannon.

Em uma visão superficial, o emissor envia a mensagem dividida em três blocos. O primeiro contém a mensagem em si. O segundo bloco contém as paridades do primeiro, computada usando o código de convolução RSC (Recursive Systematic Convolutional Code). O último bloco contém uma permutação conhecida do primeiro bloco, novamente computada usando RSC (CARRASCO, 2008, pág 281).

O receptor trabalha em paralelo, calculando a probabilidade de cada bit ser '0' ou '1', até convergir.

Hoje em dia são usados para comunicação de satélites e espacial. Outra aplicação muito popular destes códigos é em telefonia móvel 3G.

Estes códigos competem, atualmente, com os códigos LDPC, que também possuem características interessantíssimas.

Também foram excluídos dos testes deste trabalho. Isso se deu ao fato de serem muito pesados de serem implementados em software para transmissões em alta velocidade. Ressalta-se que este tipo de código é mais propício para ser implementado, em hardware, pois é bastante paralelizável.

### **2.7.6 Códigos LDPC**

E, por último, os códigos LDPC, de 1962, atingem um excelente desempenho, muito próximo ao limite de Shannon, mas são relativamente pesados computacionalmente. Por conta deste custo computacional elevado, apenas em 1996 começaram a ser estudados novamente. Também são conhecidos como códigos Gallager (CARRASCO, 2008, pág 201), em homenagem a Robert G. Gallager, que desenvolveu o conceito destes códigos em sua tese de doutorado em 1960.

Um dos seus principais atrativos em relação aos códigos turbo é que não estão ligados a patentes.

São utilizados na transmissão de TV Digital (DVB-S2) e no padrão 10 Gigabit Ethernet (10GBase-T).

Desenvolvimentos recentes levaram estes códigos a ter melhor desempenho que os códigos turbo em aplicações com altas taxas de codificação.

Também foram excluídos dos testes deste trabalho, por razões idênticas aos códigos turbo.

## 3 TRABALHOS RELACIONADOS

Foram estudados 5 trabalhos relevantes com assuntos relacionados a este trabalho de graduação. O primeiro mostra as vantagens de se utilizar FEC para streaming de vídeo. O segundo apresenta um mecanismo de FEC para UDP, semelhante aos testados neste trabalho, porém adaptativo. O terceiro testa FEC contra ACC (Frame Accumulator) para VoIP em redes congestionadas. Já o quarto tem objetivos semelhantes ao primeiro, mas aplica FEC uma camada abaixo, na camada de rede. O último estuda o comportamento do protocolo TCP na presença de erros.

### 3.1 FEC em streaming de Vídeo

O trabalho de Chen (CHEN, 2006) mostra como usar FEC adaptativo para proteger dados importantes na transmissão de vídeo no formato H.264, sobre o protocolo UDP. Ele utiliza os códigos do tipo Reed-Solomon. O FEC é adaptativo no sentido de alternar o mecanismo de inserção de redundância dos dados enviados, dependendo das condições do canal de transmissão.

O trabalho é voltado para transmissão de vídeo para telefonia móvel, com taxas de erros de até  $10^{-2}$ , mas geralmente da ordem de  $10^{-3}$ . Os testes foram realizados utilizando o protocolo UDP-Lite (UDP com checksum reduzido e cabeçalho compactado, RFC 3828) numa rede Bluetooth.

Os resultados finais foram bastante positivos, diminuindo as perdas de pacotes por erros residuais e melhorando o SNR (Signal to Noise Ratio) do vídeo transmitido. O SNR é melhorado entre 10% e 50%, dependendo do caso.

### 3.2 FEC Adaptativo para UDP

O trabalho de Kwon (KWON, 2005) implementa Forward Error Correction sobre o protocolo UDP. O algoritmo de FEC muda de acordo com as condições de canal.

O problema desta implementação, assim como o primeiro trabalho relacionado, é que o emissor depende de respostas enviadas pelo receptor para inferir o tipo de FEC mais apropriado naquele momento. Isto aumenta o tráfego gerado e a complexidade do protocolo UDP, tornando-o comparável ao protocolo TCP em alguns aspectos. Em redes de alta latência (como comunicação no espaço ou satélites), as estatísticas de perda e corrupção de pacotes podem chegar muito tarde ao emissor.

Um ponto positivo deste tipo de implementação, segundo os autores, é que a escolha dinâmica do tipo de codificação FEC mais adequada preserva a banda do canal de comunicação, não enviando informações redundantes quando não é necessário.

### 3.3 FEC vs. ACC para VoIP em redes congestionadas

Diferentemente dos dois primeiros, no trabalho de Praestholm (PRAESTHOLM, 2007), o objetivo não é implementar codificação FEC sobre UDP. Ele compara FEC com ACC para descobrir qual dos dois métodos é mais eficiente para tráfego VoIP, numa rede congestionada. Ambos os métodos são comparados nos quesitos de perda e atrasos de quadros. A rede é modelada de acordo com uma fila do tipo M/M/1/K.

O trabalho conclui que FEC é melhor para redes não congestionadas, mas quando o grau de congestionamento cresce, existe um ponto que ACC passa a ser melhor.

### 3.4 FEC na camada de Rede

O trabalho de Hui (HUI, 2001) implementa um mecanismo FEC para protocolos não confiáveis (como UDP) abaixo do nível de socket, mais precisamente, na camada de rede. O interessante desta implementação é que programas existentes podem, sem qualquer alteração, tirarem proveito dos mecanismos FEC. Isso se deve ao fato da camada de rede no modelo MR-OSI ser independente da camada de aplicação.

Além disso, o mecanismo FEC pode ser ajustado de dois modos: manual e automático. No modo manual o mecanismo FEC deve ser especificado no sentido de quanto de capacidade de correção de erros se deseja. No modo automático o mecanismo é ajustado de acordo com as estatísticas de erro enviadas pelo receptor.

### 3.5 Comportamento do TCP com perda de quadros

O trabalho de Yang (YANG, 2009) apresentou um modelo para estimar o comportamento do protocolo TCP no caso da rede perder pacotes em ambos os sentidos de transmissão (emissão e recepção). Também foi feita uma simulação no ns-2 (Network Simulator, v2 ). Os resultados mostram que a penalidade, em termos de desempenho, para perda de pacotes é alta. O desempenho cai bruscamente a medida que se ocorrem problemas de transmissão na rede. Dependendo das características da rede, uma taxa de perda de pacotes de 5% diminui a vazão média em mais de 30%.

### 3.6 Considerações sobre os Trabalhos Relacionados

Chen e Praestholm mostram que, de fato, utilizar codificação FEC produz resultados bons para determinadas aplicações, embora nem sempre seja melhor que outras técnicas. Contudo, são os trabalhos de Kwon e Hui que abordam questões mais delicadas e interessantes para este trabalho.

Kwon decidiu implementar FEC na camada de transporte, enquanto Hui na camada de rede. Estas duas abordagens diferem da utilizada neste trabalho, onde a codificação FEC é feita na camada de aplicação. Codificar todo o *payload* do quadro ethernet, ou seja, o pacote inteiro IP com FEC, é restringir o universo de possíveis aplicações. Por exemplo, durante um streaming de vídeo através do site *www.youtube.com*, os pacotes IP passam por diversos roteadores. Estes roteadores, para realizar sua função, precisam saber o endereço IP de destino destes pacotes. Esta implementação possivelmente requer que cada roteador decodifique o *payload* da camada de enlace para descobrir como rotear o pacote. Isto gera um sobrecusto que pode ser excessivo nas redes atuais.

Praestholm faz a codificação FEC na camada de transporte. Embora do ponto de vista

de desempenho seja muito melhor do que fazer isso na camada de rede, ainda teremos sérios problemas. NATs (Traditional NATs) estão espalhados por toda internet e realizam tradução de endereços e portas. Para realizar estas traduções, evidentemente, eles precisam das informações de endereços IP e de portas. Ao se codificar o *payload* do pacote IP (datagrama UDP no caso), a informação de porta fica escondida, sendo necessário que o equipamento NAT decodifique o datagrama UDP antes de realizar seu trabalho. Não é razoável incumbi-los de mais esta tarefa, que acarreta em um sobrecusto de processamento.

Yang mostra que, de fato, existe vantagem e se utilizar FEC para transmissão de dados já que o protocolo TCP não demonstra um bom desempenho em redes com perda e/ou corrupção de quadros.

## 4 IMPLEMENTAÇÃO

O trabalho é composto por 4 partes de software: Codificadores FEC, decodificadores FEC, injetor de falhas e o software de gerenciamento de testes.

Este projeto utiliza tanto novas codificações quanto o reuso de software. Certamente não seria possível implementar todos os módulos necessários, em tempo hábil, partindo do início. O reuso de software, principalmente no que tange as codificações FEC, ajuda a se concentrar na análise dos conceitos envolvidos e na apuração dos dados de desempenho.

### 4.1 Visão Geral

A implementação básica é composta basicamente por 2 partes essenciais: os codificadores FEC e os decodificadores FEC. Com estas duas partes é possível se utilizar UDP instrumentado com FEC para transmissão de dados quaisquer. Isto é mostrado na figura 4.1.

Entretanto, para realizar os testes são necessários o injetor de falhas e o software de gerenciamento de testes.

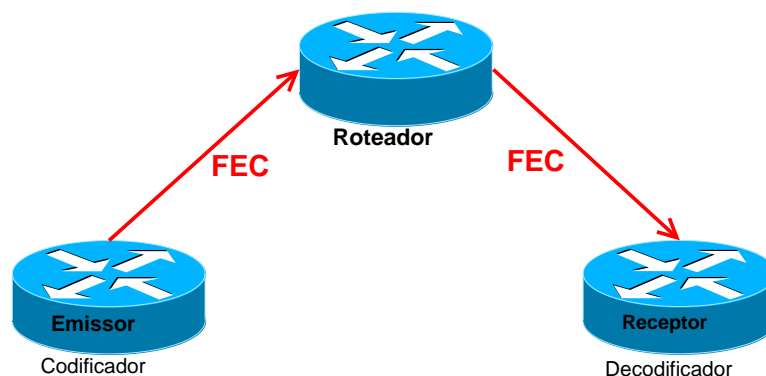


Figura 4.1: Transmissão com FEC

### 4.2 Codificador e Decodificador FEC

Os mecanismos de codificação FEC foram implementados na camada de aplicação do modelo MR-OSI. A mensagem codificada com FEC será transmitida no *payload* do protocolo UDP. Portanto, foram usados sockets UDP da forma usual. Na figura 4.2 podemos ver o formato de quadro deste protocolo UDP e na figura 4.3 podemos ver o mesmo quadro carregando dados com FEC.





Figura 4.2: Quadro do Protocolo UDP

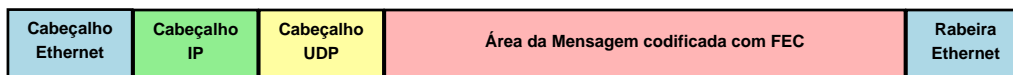


Figura 4.3: Quadro do Protocolo UDP, codificado com FEC na área de mensagem

Foram utilizados duas codificações FEC neste trabalho. Reed-Solomon e BCH.

O codificador Reed-Solomon utilizado foi baseado no projeto RSCODE (RSCODE, v1.3). O código original foi modificado para atender aos requisitos deste trabalho. Entre as modificações mais importantes estão a divisão de codificador e decodificador com dois programas diferentes e a inclusão de suporte a medição de tempo de codificação/decodificação.

O codificador BCH foi baseado no simulador BCH3 (BCH3, 1997). O código original foi modificado para atender aos requisitos deste trabalho. Entre as modificações mais importantes estão a divisão de codificador e decodificar com dois programas diferentes e a inclusão de suporte a medição de tempo de codificação/decodificação.

### 4.3 Software de Gerenciamento dos Testes

O Software de Gerenciamento dos Testes foi implementado em Python. Este software cria mensagens aleatórias, de tamanho pré-definido, chama o codificador FEC, implementado em linguagem C, para codificá-las, e as transmite para o receptor. O mesmo software de gerenciamento de testes recebe estas mensagens no receptor, chama o decodificador FEC para decodificar a mensagem e a mesma é enviada ao emissor onde é feita a conferência com a mensagem original. No emissor, as estatísticas sobre o teste são geradas. Isso pode ser visto na figura 4.4.

Também faz parte das atribuições do software de gerenciamento de testes controlar a execução dos testes assim como o número de repetições dos mesmos.

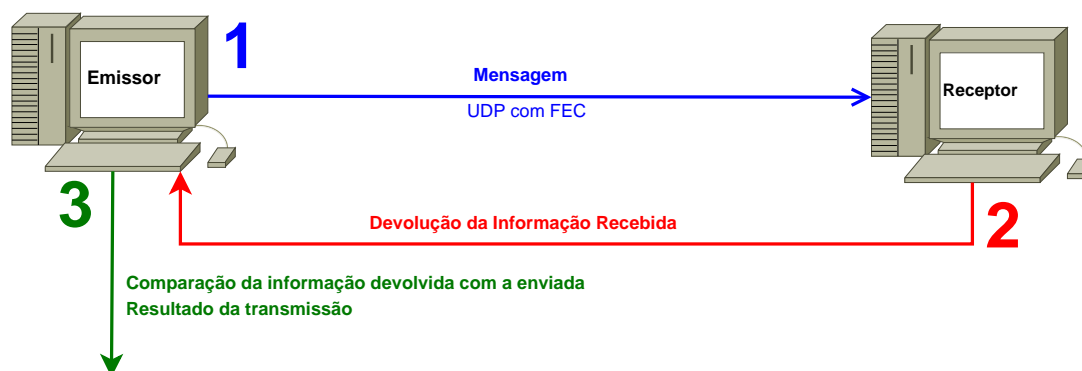


Figura 4.4: Esquema das Funcionalidades do Software de Gerenciamento de Testes

#### 4.4 Injetor de Falhas

Para possibilitar análise do comportamento dos dois mecanismos FEC estudados e estabelecer a comparação entre o protocolo TCP e o UDP instrumentado com FEC Reed-Solomon, foi inserido no sistema um injetor de falhas, assim como mostrado na figura 4.5. Nela pode-se ver que um pacote, ao deixar o emissor e ser codificado, passa pelo injetor de falhas antes de chegar ao decodificador e receptor. Este injetor possui a função de corromper pacotes UDP e TCP.

Foi utilizado o *Firmament*, desenvolvido no Grupo de Tolerância a Falhas do Instituto de Informática da UFRGS (Drebes, R. J.). Este injetor foi utilizado com sucesso por (Siqueira, T.) para testar o protocolo SCTP (Stream Control Transmission Protocol).

O injetor de falhas foi configurado para corromper 1 byte, de posição fixa, dos pacotes TCP e UDP, com probabilidades de 2%, 5% ou 10%. A escolha de corromper apenas um byte em posição fixa se deve ao fato de que os decodificadores Reed-Solomon e BCH se comportam, do ponto de vista de desempenho, da mesma forma para corrigir qualquer erro, desde que dentro de sua capacidade de correção. Também não interfere, para estas codificações, a posição destes erros. Não importa se forem 8 erros espalhados pela área de dados, ou em sequência, ou apenas 1 sempre na mesma posição, os resultados do ponto de vista de desempenho, serão os mesmos. Como este trabalho busca dados de desempenho destes códigos, optou-se pela configuração mais simples do injetor de falhas. Foi simulado então, um canal com erros do tipo sem memória, conforme explicado na seção 2.2.

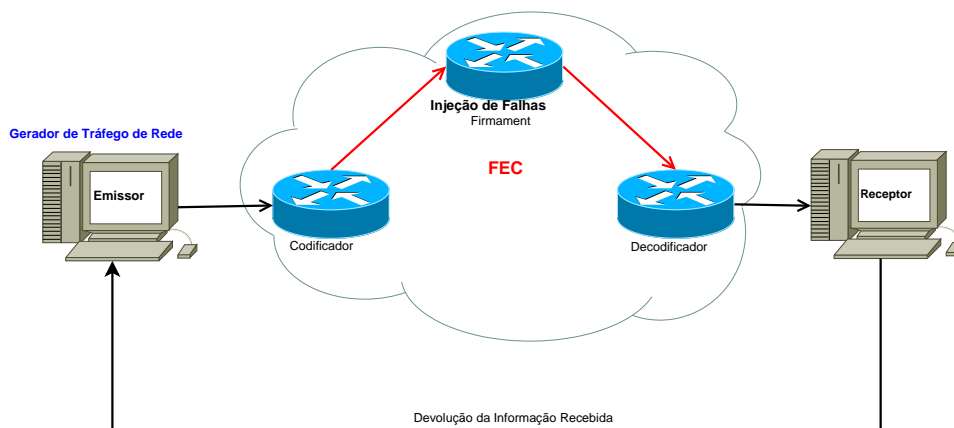


Figura 4.5: Esquema de Avaliação da Implementação

## 5 ESPECIFICAÇÃO DOS TESTES

### 5.1 Descrição

Os testes realizados podem ser divididos em três categorias independentes. Ou seja, os resultados de um teste não interfere nos outros nem é necessária uma ordenação da condução dos testes.

As categorias são:

1. Transmissão sem erros, BCH e Reed-Solomon
2. Transmissão com erros, BCH e Reed-Solomon
3. Transmissão com erros, TCP e UDP+Reed-Solomon

Para as primeiras duas categorias, utilizou-se um tamanho fixo de mensagem: 250 bytes. Ou seja, o *payload* útil de cada datagrama UDP instrumentado com FEC é de 250 bytes. Este tamanho de mensagem foi escolhido de forma arbitrária tentando chegar a um número que represente a média. Ressalta-se que a redundância das codificações FEC não está incluída nestes 250 bytes.

Já para os testes envolvendo TCP, fez-se a transferência de um arquivo de 100 Megabytes através de um programa implementado em *Python*, que utiliza os usuais sockets TCP. Comparou-se isso a transferência do mesmo arquivo utilizando-se UDP instrumentado com Reed-Solomon de 8 Bytes de capacidade de correção.

Equipamento Utilizado:

- Linux Kernel 2.6.33
- GCC 4.5.0
- Python 2.6.5
- Hardware: AMD Phenom X4 Deneb 3,8Ghz. 4GB RAM, Dual NIC
- Hardware Injetor de Falhas: AMD Athlon X2 2,6Ghz. 2GB RAM, Dual NIC.

### 5.2 Transmissão sem erros, BCH e Reed-Solomon

Os códigos foram configurados para suportar 1, 2, 4, 8 e 16 bytes alterados durante a transmissão e ainda permitirem que a informação original fosse recuperada. Para que as interferências do escalonador do sistema operacional e das cargas de outros aplicativos quaisquer nos resultados fossem minimizadas ao máximo, optou-se por um intervalo de

5 segundos entre cada mensagem do emissor para o receptor, e repetiu-se este processo 1000 vezes.

O intervalo de 5 segundos serve para impedir que alguma tarefa do sistema operacional execute e prejudique um número maior de testes. As 1000 repetições foram escolhidas de forma arbitrária após alguns ciclos inconclusivos de cálculo do número de amostras necessárias para as estatísticas do teste.

Com este teste buscou-se medir o custo computacional dos codificadores e decodificadores contra suas capacidades de correção de erros. Além do impacto que a informação redundante adicionada aos dados causa na transmissão até o receptor.

Portanto:

- Número de repetições: 1000
- Intervalo entre cada repetição: 5 segundos
- Variações dos Códigos: 5 (1, 2, 4, 8 e 16 bytes)
- Sem erros no canal

Dados Obtidos:

- Tempo de codificação de cada quadro
- Tempo de decodificação de cada quadro
- Tempo total de transmissão de cada quadro (codificação + decodificação + transmissão)

Para cada dado obtido, foram calculadas as médias assim como os intervalos de confiança, para uma confiança de 95%.

### 5.3 Transmissão com erros, BCH e Reed-Solomon

Os códigos foram configurados para suportar 8 bytes corrompidos durante a transmissão e ainda permitirem que a informação original fosse recuperada. O canal de comunicação introduziu erros (injetor de falhas), com probabilidade 1, de 1, 2, 4 e 8 bytes nas mensagens enviadas pelo emissor para o receptor. Para que as interferências do escalonador do sistema operacional e das cargas de outros aplicativos quaisquer nos resultados fossem minimizadas ao máximo, optou-se por um intervalo de 5 segundos entre cada mensagem do emissor para o receptor, e repetiu-se este processo mil vezes.

Não foi possível realizar os testes com os códigos configurados para suportarem 16 bytes corrompidos por questões de desempenho, em especial o tempo de codificação do código Reed-Solomon para o codificador utilizado.

Com este teste buscou-se medir o custo computacional dos codificadores e decodificadores contra a quantidade de erros introduzidas pelo canal.

Portanto:

- Número de repetições: 1000
- Intervalo entre cada repetição: 5 segundos
- Capacidade de Correção: 1 (8 bytes)

- Variações dos Erros: 4 (1, 2, 4, 8 bytes)
- Sempre ocorre erro na transmissão (dentro da capacidade de correção do código)

Dados Obtidos:

- Tempo de decodificação de cada quadro

Não foi necessário medir novamente tempos de codificação pois estes valores seriam os mesmos obtidos nos testes da categoria anterior. O tempo de execução do codificador no emissor independe do que acontece com o datagrama após a codificação.

Para o dado obtido, foi calculado a média assim como o respectivo intervalo de confiança, para uma confiança de 95%.

## 5.4 Transmissão com erros, TCP e UDP com Reed-Solomon

Transmitir um arquivo de 100MB de um ponto A para o ponto B, utilizando tanto o protocolo TCP quanto o UDP instrumentado com Reed-Solomon. O canal de transmissão foi configurado com erros (alteração de byte) introduzidos pelo injetor de falhas, nas taxas de 0, 1, 2, 5 e 10 quadros a cada 100. Foram feitas 100 repetições, com intervalos de 60 segundos entre elas para cada taxa de erro. As taxas de injeção de erros são as usualmente utilizadas na área de Tolerância a Falhas. O intervalo de 60 segundos segue o mesmo princípio dos testes anteriores, evitar que processos não previstos influenciem mais nos testes. As 100 repetições foram escolhidas de forma arbitrária, era o maior número possível de repetições para se executar em tempo hábil para o término do trabalho.

O erro consiste em corromper 1 byte do *payload* do pacote TCP. Com isso, este pacote é descartado no momento em que se calcula seu checksum. Não faz diferença corromper 1 ou mais bytes, já que o código FEC se comporta do mesmo jeito, do ponto de vista de desempenho, frente ao número de bits corrompidos, desde que estes estejam dentro de sua capacidade de correção.

O vazão máxima da rede foi limitada para 5 Megabytes/segundo a fim de impedir que qualquer flutuação na capacidade da rede ou escalonamento de processo interferisse no resultado. Isto foi feito através de uma política de fila HTB (Hierarchical Token Bucket) na interface de saída do emissor.

Portanto:

- Transmissão de 100MB
- Intervalo entre cada repetição: 60 segundos
- Reed-Solomon para 8 bytes vs TCP
- Erro no canal com probabilidades de 0, 0.01, 0.02, 0.05 e 0.1

Dados Obtidos:

- Tempo de transmissão do arquivo

Para o dado obtido, foi calculada a média assim como o respectivo intervalo de confiança, para uma confiança de 95%.

## 5.5 Faultlet para o Firmament

Foi utilizado o faultlet abaixo no injetor de falhas Firmament. A taxa de erros é ajustada de acordo com o teste a ser realizado, conforme comentado no código.

```

SET 2 R0
READS R0 R10 Le o tamanho total do pacote
SET 1 R0
SUB R0 R10
SET 100 R0
RND R0 R1
SET 80 R0  $(100 - x) / 2 = \text{Probabilidade de ser selecionado em \%}$ .
ADD R0 R1
JMPN R1 SELECTED
ACP
SELECTED: Pacote foi sorteado
Descobrimo localização do cabeçalho UDP/TCP
SET 0 R1
READB R1 R0
SET 0xf R1
AND R1 R0
SET 4 R1
MUL R1 R0 Localização colocada em R0
Descobrimo qual o protocolo de transporte
SET 9 R3
READB R3 R2
SET 6 R3  $6 = \text{TCP}$ 
SUB R3 R2
JMPZ R2 TCP
UDP: Presumindo que, se não é TCP, é UDP
SET 8 R1
ADD R1 R0 O cabeçalho de UDP tem sempre 8 bytes
JMP ALTERAR
TCP: O protocolo é TCP
Descobrimo o tamanho do cabeçalho TCP
MOV R0 R2
SET 12 R1
ADD R1 R2
READB R2 R1
SET 0xf0 R2
AND R2 R1
SET 4 R2
DIV R2 R1
ADD R1 R0
JMP ALTERAR
ALTERAR:
Se não tiver área de dados, deixar passar sem alterar
SUB R0 R10
JMPN R10 NODATA

```

*R0 aponta para o início da área de dados*

READB R0 R1

MOV R1 R3

SET 1 R2

AND R2 R3

SUB R3 R2

SET 0xfe R3

AND R3 R1

OR R2 R1

WRTEB R0 R1 *Insero Erro*

**NODATA:**

ACP

## 6 EXECUÇÃO E RESULTADOS DOS TESTES

Os testes realizados neste trabalho estavam sujeitos a alguns fatores que podem ter afetado as medidas. Fatores estes dificilmente contornáveis e comuns em experimentos deste tipo. É impossível garantir que os tempos medidos reflitam os números reais, por três motivos.

O primeiro é a ordem de grandeza dos números medidos. São, de fato, tempos muito pequenos. Mesmo se utilizando contadores de tempo de alta resolução, a precisão destes não é garantida.

O segundo fator é o escalonador do sistema operacional. Como os tempos são muito pequenos, decisões do escalonador podem ter impactos profundos nos tempos medidos.

O último fator é a interferência dos mecanismos de medição no desempenho dos códigos: é impossível determinar exatamente quanto dos valores medidos pode ser atribuído aos trechos de código utilizados para realizar as medições de tempo.

Mesmo assim, o objetivo destes testes pôde ser cumprido: o de traçar os perfis dos códigos.

### 6.1 Transmissão sem erros, BCH e Reed-Solomon

#### 6.1.1 Detalhamento da Execução

Na figura 6.1 são apresentados os pontos onde foram monitorados os tempos de execução. No emissor, uma mensagem aleatória (string de 250 bytes) foi gerada a cada 60 segundos pelo Software de Gerenciamento de Testes. Escolheu-se também uma codificação FEC (BCH ou Reed-Solomon) e a sua capacidade de correção (1, 2, 4, 8 ou 16 bytes), chamando o codificador FEC correspondente. O tempo gasto pelo codificador FEC foi registrado. Para cada mensagem, quando o codificador FEC terminou, o controle foi retomado pelo Software de Gerenciamento de Testes, que enviou a mensagem codificada para um socket UDP tendo o receptor como destino. O receptor então recebeu esta mensagem e a decodificou, de forma análoga ao que aconteceu no emissor, mas utilizando, naturalmente, um decodificador FEC. O tempo gasto para decodificação foi registrado. Então o receptor enviou a mensagem decodificada de volta para o emissor, via uma conexão confiável TCP, para simples conferência e validação da transmissão.

Para cada combinação de código e capacidade de correção foram realizadas 1000 repetições. Foram feitas, portanto, 10.000 transmissões e 30.000 dados coletados. Ao se chegar neste número, a mensagem aleatória deixou de ser gerada e o teste completado. Tanto o emissor quanto o receptor estavam na mesma máquina real, cada um com uma placa de rede dedicada, para tirar proveito do mesmo relógio e facilitar as medições. Portanto, o emissor e receptor mostrados nas figuras 4.4 e 4.5 foram simplificados para



apenas uma máquina real.

Todo o software envolvido, retirando-se a parte do sistema operacional e do codificador FEC, faz parte do software de gerenciamento de testes.

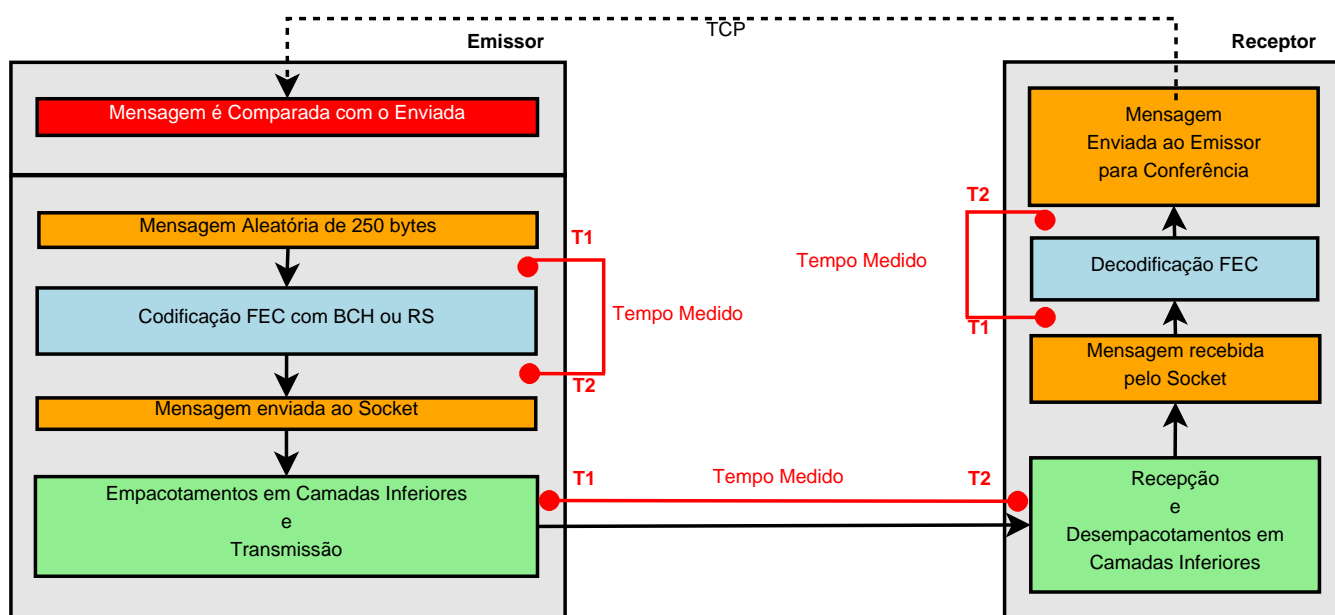


Figura 6.1: Detalhes da Execução do Teste - Sem Erros na Rede

### 6.1.2 Resultados

Coletados os dados de tempo de codificação para os dois códigos, criou-se o gráfico mostrado na figura 6.2. Percebe-se que o processo de codificação BCH, para mensagens de 250 bytes é menos custoso computacionalmente que um Reed-Solomon. Conforme a capacidade de correção dos códigos aumenta, esta diferença de custo computacional fica maior ainda.

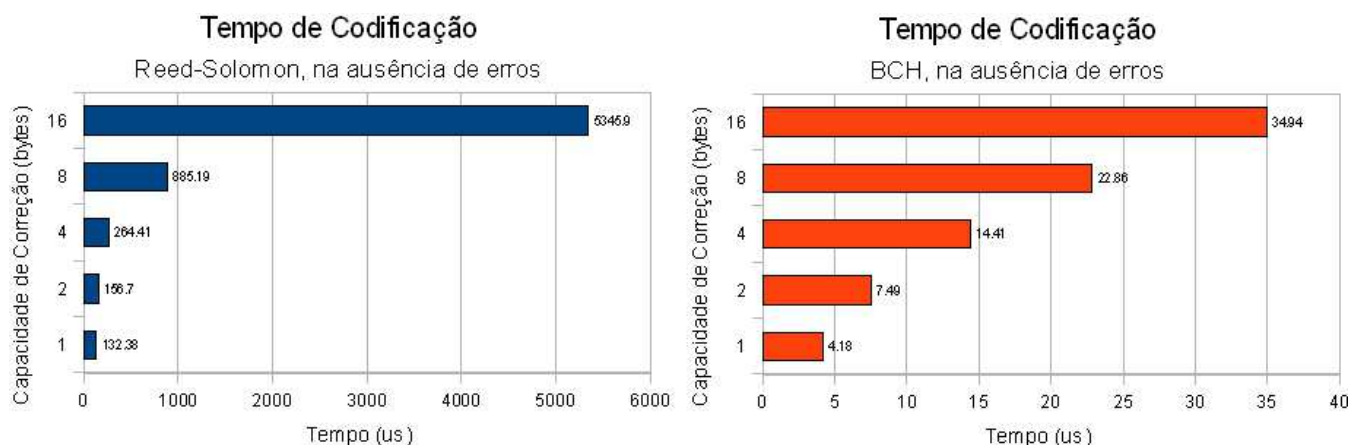


Figura 6.2: Tempo de Codificação na Ausência de Erro

Por exemplo, enquanto um BCH de capacidade de correção de 1 byte demora 4,18 microssegundos para ser gerado, a mesma mensagem com Reed-Solomon demora 132,38 microssegundos. Uma diferença de quase 32 vezes. No outro extremo, um BCH consome 33,94 microssegundos e um Reed-Solomon 5345,9 microssegundos, uma diferença de

mais de 150x. Portanto, no quesito custo computacional para codificação, o BCH se sai muito melhor.

Já o gráfico mostrado na figura 6.3 ilustra os dados recolhidos no decodificador. Percebe-se que o processo de decodificação Reed-Solomon, para mensagens de 250 bytes é menos custoso computacionalmente que um BCH. Diferentemente do que aconteceu no codificador, o custo do BCH é sempre por volta de 2 vezes maior que o do Reed-Solomon.

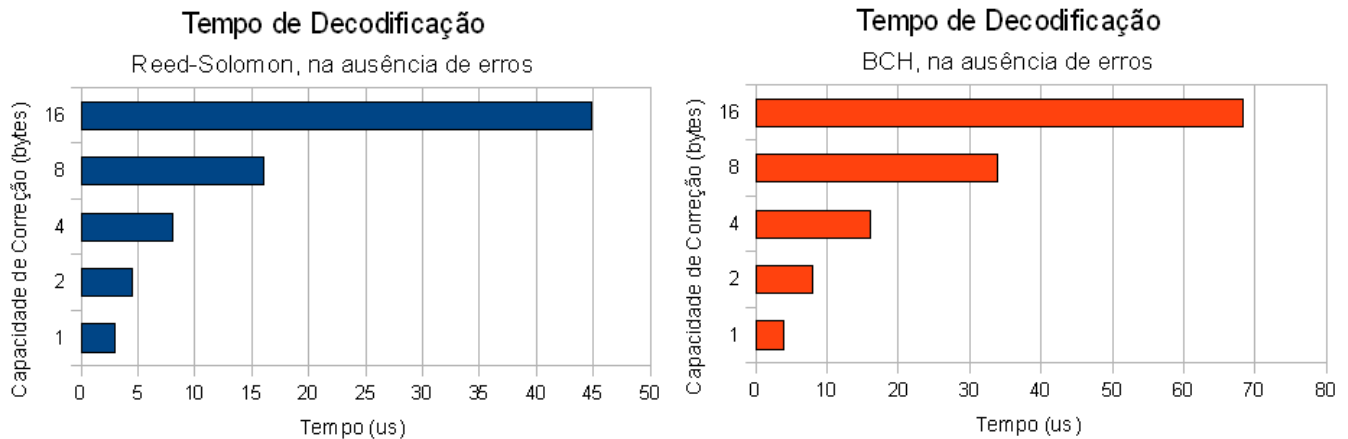


Figura 6.3: Tempo de Decodificação na Ausência de Erro

Por exemplo, enquanto uma mensagem codificada com BCH de capacidade de correção de 4 Bytes demora 16,1 microssegundos para ser decodificado, a mesma mensagem com Reed-Solomon demora 8,02 microssegundos. Portanto, no quesito custo computacional para decodificação, o Reed-Solomon se saiu melhor.

Finalmente, combinando os tempos de codificação, decodificação e juntando-os ao tempo de transmissão no canal de comunicação, obtemos o gráfico mostrado na figura 6.4. Nele, fica evidente que a vantagem obtida pelo BCH no quesito tempo de codificação é mais do que o suficiente para superar, por uma grande margem, o Reed-Solomon quando se somam todos os tempos.

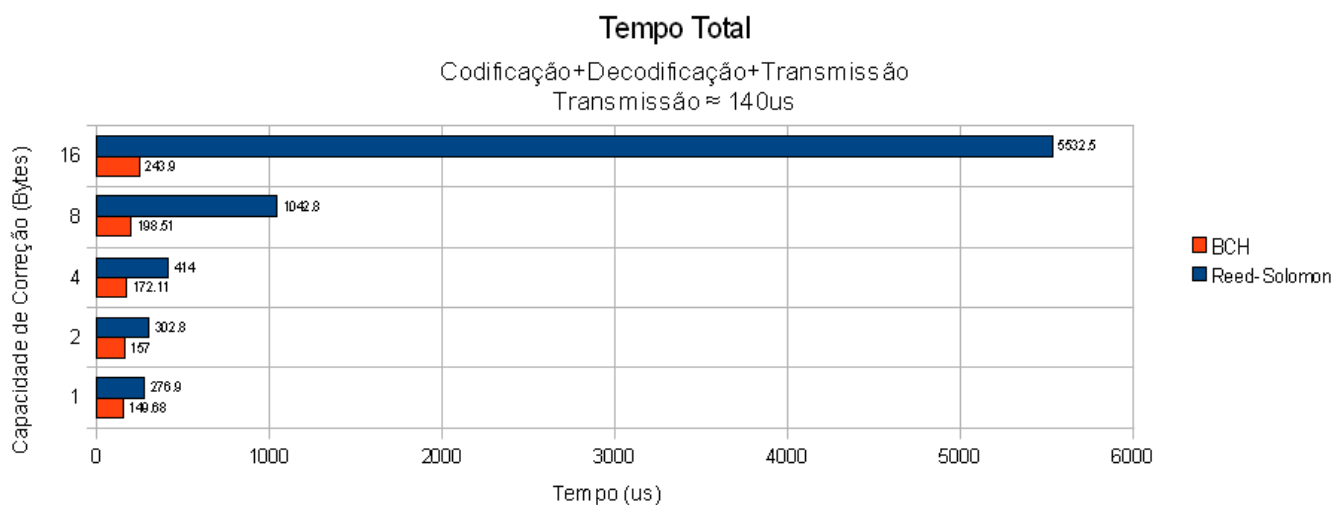


Figura 6.4: Tempo Total na Ausência de Erro

Nota-se que o tempo de transmissão ficou por volta de 140 microssegundos em todos os testes. Isso se deve ao fato de não ser utilizado toda capacidade do datagrama do

quadro UDP (relativo a MTU) e, portanto, os bytes extras de redundância que surgem com o aumento da capacidade de correção de erros não precisam ser enviados em um segundo datagrama. Além disso, frente a quantidade de bytes enviados, esses bytes extras são pouco significativos.

## 6.2 Transmissão com erros, BCH e Reed-Solomon

### 6.2.1 Detalhamento da Execução

Na figura 6.5 são apresentados os pontos onde foram monitorados os tempos de execução. No emissor, uma mensagem aleatória foi gerada a cada 60 segundos pelo Software de Gerenciamento de Testes. Escolheu-se também uma codificação FEC (BCH ou Reed-Solomon) e a sua capacidade de correção (1, 2, 4 ou 8 bytes), chamando o codificador FEC correspondente. O tempo gasto pelo codificador FEC foi registrado. Quando o codificador FEC terminou, o controle foi retomado pelo Software de Gerenciamento de Testes, que enviou a mensagem codificada para um socket UDP tendo o receptor como destino. Na rede, entre o emissor e o receptor, estava o injetor de falhas, que corrompeu um byte do *payload* do protocolo UDP. O receptor então recebeu esta mensagem corrompida e a decodificou, recuperando-a. O tempo gasto para decodificação foi registrado. Então o receptor enviou a mensagem decodificada de volta para o emissor, via uma conexão confiável TCP, para simples conferência e validação da transmissão.

Para cada combinação de código e capacidade de correção são realizadas 1000 repetições. Foram feitas, portanto, 2.000 transmissões e coletadas 2.000 medidas. Ao se chegar neste número, a mensagem aleatória deixou de ser gerada e o teste completado. Tanto o emissor quanto o receptor estavam na mesma máquina real, cada um com uma placa de rede dedicada, para tirar proveito do mesmo relógio e facilitar as medições. O injetor de falhas estava em outra máquina real.

Todo o software envolvido, retirando-se a parte do sistema operacional e do codificador FEC, faz parte do software de gerenciamento de testes.

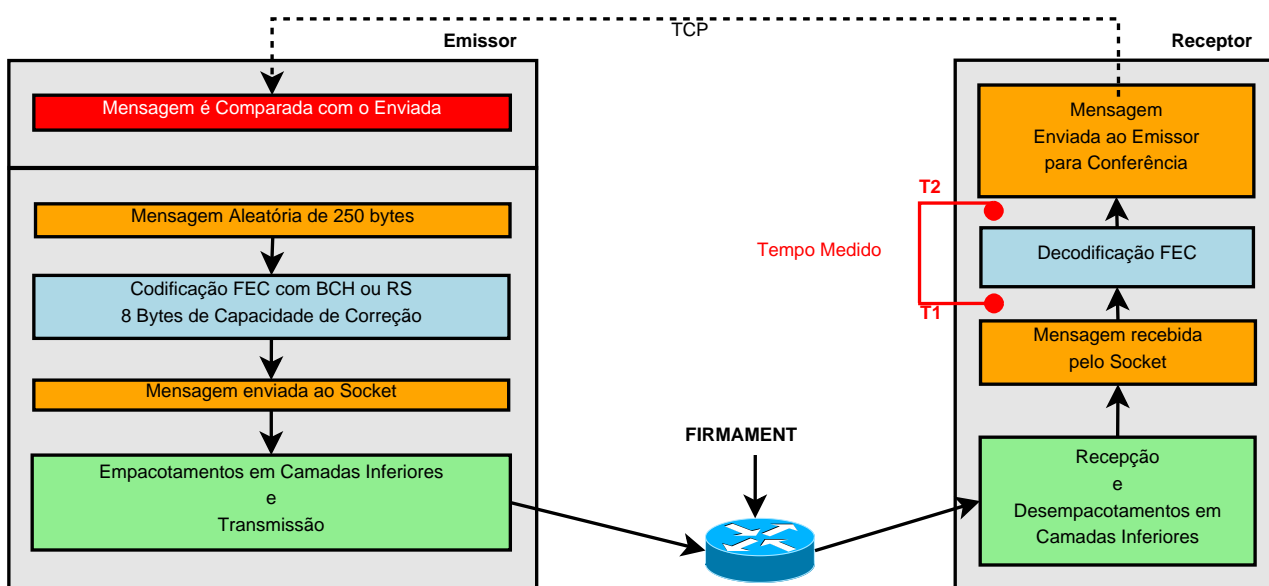


Figura 6.5: Detalhes da Execução do Teste - Com Erros na Rede

## 6.2.2 Resultados

Com a coleta dos dados da segunda categoria de testes, foi possível verificar os dados apresentados no gráfico da figura 6.6. Fica evidente que tanto para o decodificador BCH quanto para o Reed-Solomon, a quantidade de erros introduzida pelo canal de comunicação não interfere nos seus custos computacionais. O que determina o custo é a ausência ou a presença de erros, nada mais.

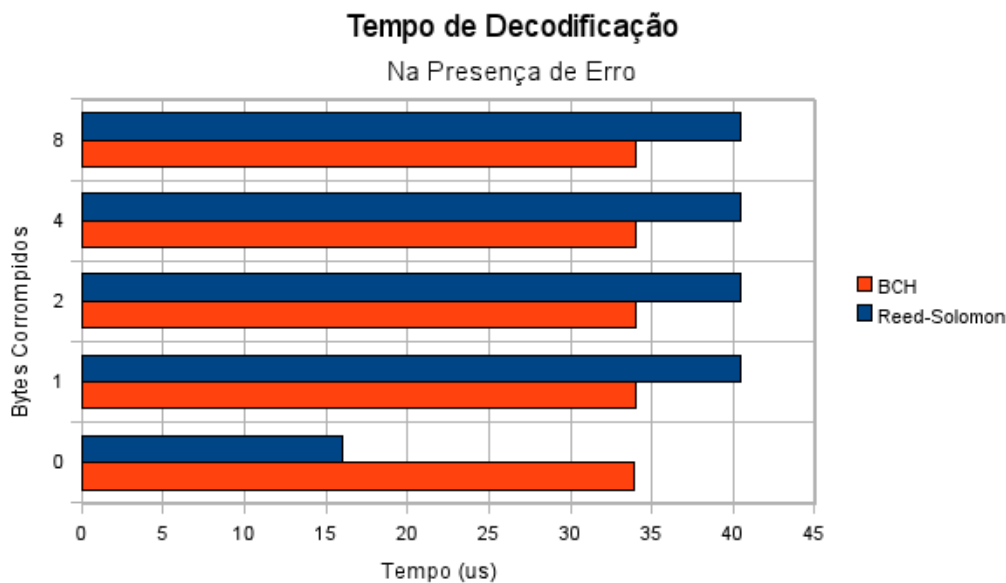


Figura 6.6: Tempo de Decodificação na Presença de Erro

Dito isso, podemos visualizar que o decodificador Reed-Solomon, na presença de erros, perde a vantagem competitiva que possuía na ausência de erros, onde era mais rápido que o BCH.

## 6.3 Transmissão com erros, TCP e UDP com Reed-Solomon

### 6.3.1 Detalhamento da Execução

Na figura 6.7 são apresentados os pontos onde foram monitorados os tempos de execução. No emissor, um arquivo de 100 Megabytes foi fragmentado em mensagens de 1000 bytes. Estas mensagens foram encaminhadas para um socket TCP ou para um codificador FEC (Reed-Solomon com capacidade de correção 8 bytes) e por conseguinte para um socket UDP. A transmissão foi feita, mas na rede encontrava-se o injetor de falhas capaz de corromper tanto pacotes TCP como datagramas UDP. Ele os corrompeu com probabilidades de 0, 0.01, 0.02, 0.05 ou 0.1. A probabilidade de corrupção não mudou durante a transmissão dos fragmentos do mesmo arquivo. No receptor, os fragmentos do arquivo foram decodificados (caso estivessem usando FEC) e os dados corrompidos, no caso, recuperados. Para TCP, o pacote corrompido foi descartado (falhou na detecção de erros em algum ponto da pilha TCP/IP) e o próprio protocolo TCP se encarregou de retransmitir estes dados perdidos. Foi medido o tempo entre o fim da codificação de todos fragmentos, no emissor, e o início da remontagem do arquivo, no receptor. Os fragmentos, no caso de FEC, foram todos codificados previamente, pois não havia tempo de processamento confortável o suficiente para codificar *just-in-time* mantendo o fluxo de 5

Megabytes/segundo na rede. A figura 6.4 mostra este problema de forma bastante clara. Para o TCP, o algoritmo de Nagle (RFC896, 1984) foi desabilitado.

Feito isso, o arquivo remontado no receptor foi transmitido para o emissor, a fim de ser conferido. Este teste foi repetido 100 vezes para cada probabilidade de corrupção pelo injetor de falhas e para cada protocolo.

Como são 10 combinações de protocolos e de probabilidades de injeção de erros, foram feitas 1000 transmissões de arquivos, gerando 1000 dados coletados. As transmissões usando FEC foram feitas separadas das TCPs.

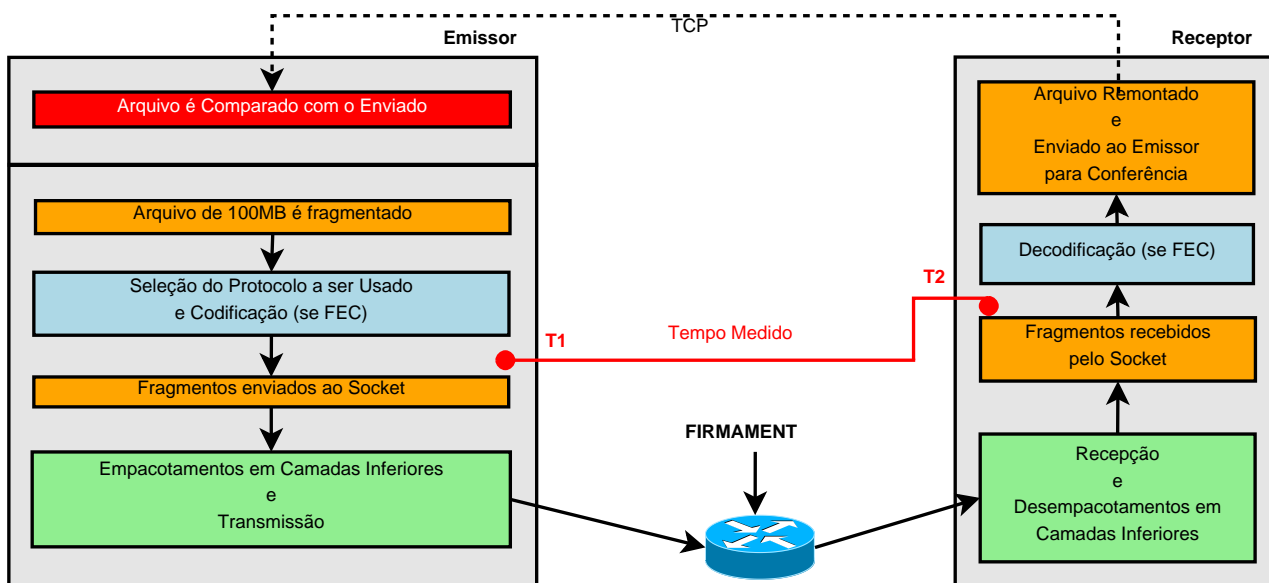


Figura 6.7: Detalhes da Execução do Teste com TCP e UDP+FEC

### 6.3.2 Resultados

Nota-se pela figura 6.8 que, na ausência de erros, se conseguem taxas de transferência muito parecidas entre Reed-Solomon com capacidade de correção de 8 bytes e TCP. Entretanto, na presença de erros, o tempo de transmissão do arquivo aumenta consideravelmente para o TCP enquanto permanece estável para a transmissão que utiliza FEC.

## 6.4 Resumo dos Resultados

São mostradas 5 tabelas. As primeiras três (6.1, 6.2 e 6.3) apresentam os resultados do experimento da seção "Transmissão sem Erros, BCH e Reed-Solomon". A primeira delas mostra os tempos de codificação dos codificadores FEC BCH e Reed-Solomon para 5 capacidades de correção diferentes. A segunda mostra os tempos de decodificação dos decodificadores FEC. E a terceira mostra a soma dos tempos da primeira, da segunda tabela e do tempo de transmissão.

A quarta (6.4) apresenta os tempos de decodificação para o experimento da seção "Transmissão com Erros, BCH e Reed-Solomon".

E por fim, a última (6.5) apresenta os resultados do experimento da seção "Transmissão com Erros, TCP e UDP com Reed-Solomon".

Tabela 6.1: BCH e Reed-Solomon - Tempo de Codificação

Código FEC	Capacidade de Correção	Tempo Médio de Codificação (us)	Intervalo de Confiança (us)
Reed-Solomon	1	132,38	130,53 a 134,24
Reed-Solomon	2	156,7	154,46 a 158,93
Reed-Solomon	4	264,41	260,56 a 268,26
Reed-Solomon	8	885,19	872,32 a 898,05
Reed-Solomon	16	5345,90	5267,19 a 5424,61
BCH	1	4,18	4,12 a 4,26
BCH	2	7,49	7,38 a 7,59
BCH	4	14,41	14,21 a 14,62
BCH	8	22,86	22,52 a 23,20
BCH	16	33,94	33,44 a 34,44

Tabela 6.2: BCH e Reed-Solomon - Tempo de Decodificação sem Erros

Código FEC	Capacidade de Correção	Tempo Médio de Decodificação (us)	Intervalo de Confiança (us)
Reed-Solomon	1	2,91	2,87 a 2,95
Reed-Solomon	2	4,51	4,44 a 4,58
Reed-Solomon	4	8,02	7,90 a 8,15
Reed-Solomon	8	16,05	15,81 a 16,28
Reed-Solomon	16	44,81	44,16 a 45,47
BCH	1	3,91	3,85 a 3,96
BCH	2	7,89	7,78 a 8,01
BCH	4	16,10	15,87 a 16,33
BCH	8	33,81	33,32 a 34,30
BCH	16	68,38	67,40 a 69,37

Tabela 6.3: BCH e Reed-Solomon - Tempo Total de Transmissão

Código FEC	Capacidade de Correção	Tempo Total (us)	Intervalo de Confiança (us)
Reed-Solomon	1	276,88	272,77 a 280,99
Reed-Solomon	2	302,83	298,44 a 307,22
Reed-Solomon	4	414,02	403,15 a 419,89
Reed-Solomon	8	1042,79	1027,47 a 1058,11
Reed-Solomon	16	5532,5	5452,67 a 5612,35
BCH	1	149,68	147,50 a 151,86
BCH	2	157	154,79 a 159,22
BCH	4	172,11	169,63 a 174,59
BCH	8	198,51	195,59 a 201,41
BCH	16	243,9	240,28 a 247,53

Tabela 6.4: BCH e Reed-Solomon - Tempo de Decodificação com Erros

Bytes Corrompidos	Código FEC	Tempo de Decodificação(us)	Intervalo de Confiança (us)
0	Reed-Solomon	16,05	15,81 a 16,28
0	BCH	33,81	33,32 a 34,30
1	Reed-Solomon	40,45	39,91 a 40,99
1	BCH	33,99	33,53 a 34,45
2	Reed-Solomon	40,43	39,87 a 40,99
2	BCH	33,98	33,52 a 34,42
4	Reed-Solomon	40,54	39,98 a 41,10
4	BCH	33,87	33,41 a 34,33
8	Reed-Solomon	40,51	39,97 a 41,07
8	BCH	33,95	33,50 a 34,41

Tabela 6.5: Transmissão com Erros, TCP vs. UDP com Reed-Solomon

Probabilidade de Erro na Rede (%)	Algoritmo ou Protocolo	Tempo de Transmissão(s)	Intervalo de Confiança (us)
0	UDP+RS(8)	21,41	20,57 a 22,24
0	TCP	21,35	20,45 a 22,24
1	UDP+RS(8)	21,42	20,62 a 22,21
1	TCP	30,06	28,91 a 31,22
2	UDP+RS(8)	21,38	20,55 a 22,21
2	TCP	34,39	33,03 a 35,75
5	UDP+RS(8)	21,40	20,56 a 22,24
5	TCP	62,68	60,27 a 65,09
10	UDP+RS(8)	21,43	20,55 a 22,31
10	TCP	121,08	116,50 a 125,66

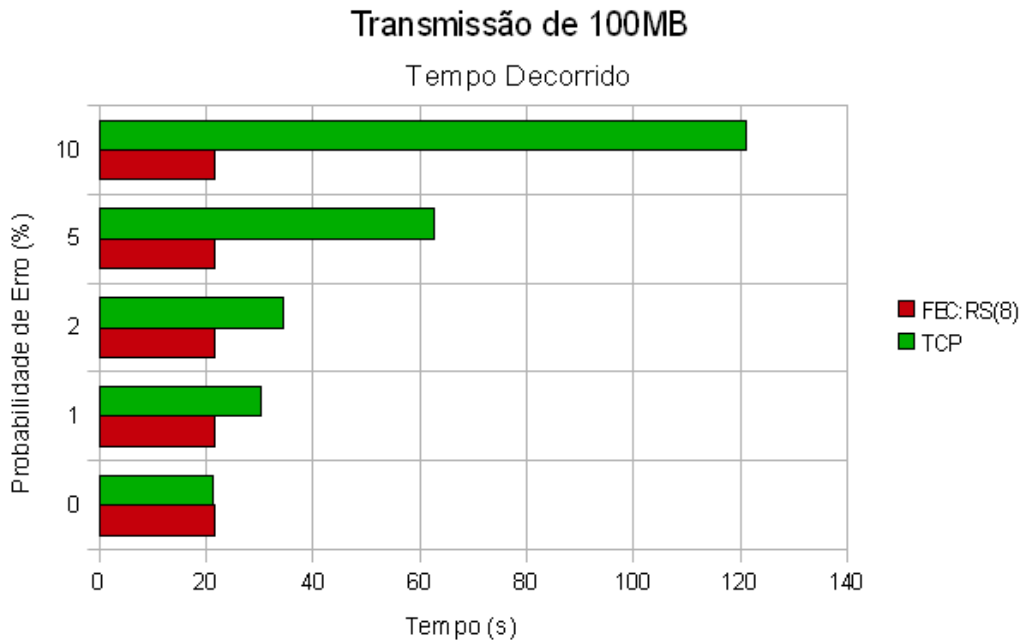


Figura 6.8: TCP vs FEC em Redes com Corrupção de Pacotes

## 6.5 Análise dos Resultados

Algumas conclusões podem ser tiradas da análise dos resultados. A primeira é que o protocolo TCP, embora confiável, só é adequado para redes onde é baixa a taxa de perda de pacotes pois seu desempenho cai muito com a introdução de erros. Por exemplo, com apenas 1% de probabilidade de corrupção, a taxa de transferência média cai em 28,97%. Já com 10%, a taxa de transferência média cai 82,37%. Este problema é ainda maior em redes com RTT (Round Trip Time) alto, segundo (YANG, 2009).

Na figura 6.9 é apresentado o valor da vazão obtida em um enlace que utiliza o protocolo TCP numa rede com 10% de probabilidade de corrupção de pacote. Note que a vazão varia significativamente ao longo tempo. Isso se deve ao mecanismo de controle de congestionamento do protocolo TCP que, ao perder pacotes (ou corrompê-los), infere que a rede está congestionada, diminuindo desta forma a vazão. Fazendo isso, o protocolo espera que o número de pacotes ou ACKs perdidos diminua, o que não ocorre, pois a probabilidade de erro continua a mesma. Assim, o TCP só consegue voltar a aumentar a vazão quando a taxa de erros diminuir (o que, no caso dos testes foi feito através do controle do injetor de falhas) no tráfego dirigido.

Com relação às codificações FEC, nota-se que códigos BCH em geral possuem custo de codificação bem menor do que Reed-Solomon, embora sejam um pouco menos eficientes no quesito quantidade de dados redundantes adicionados ao pacote. Na ausência de erros, o decodificador Reed-Solomon leva vantagem. Entretanto, quando ocorrem erros, o decodificador BCH é mais eficiente. Desta forma, uma vez que em condições normais, as redes devem conseguir entregar mais pacotes íntegros do que corrompidos, pode-se generalizar que o decodificador Reed-Solomon é menos custoso computacionalmente do que o BCH, para códigos com a mesma capacidade de correção de erros.

Entretanto, mesmo que o BCH seja geralmente mais rápido que o Reed-Solomon, existem alguns casos que deve-se preferir o último. Por exemplo, para streaming de vídeo, geralmente o emissor possui maior capacidade de processamento que o receptor. Esta capacidade maior do emissor pode ser usada para se utilizar FEC com Reed-



Solomon. Como o Reed-Solomon consome menos recursos que o BCH na decodificação, preservam-se recursos dos receptores. Isso acaba por ser uma espécie de balanceamento de carga, deixando as tarefas mais complexas para os que possuem maior poder computacional e as mais simples para os que não possuem tanto. Outro exemplo interessante será abordado agora, FEC para Multicast.

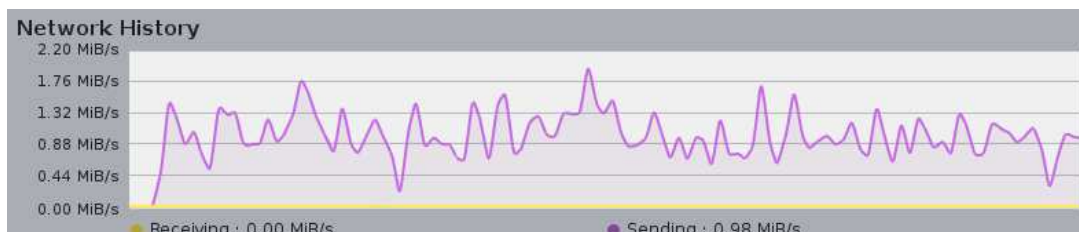


Figura 6.9: Comportamento do TCP com Erros no Canal

### 6.5.1 FEC para Multicast

Numa configuração multicast, um único emissor pode enviar uma única mensagem para qualquer número de receptores, desde que estes receptores estejam inscritos no grupo multicast criado por aquele emissor. As mensagens são entregues para múltiplos destinatários, mas só passam por um determinado link uma única vez. Isso independe do número de destinatários conectados ao emissor por aquele link. A duplicação das mensagens só ocorre quando o link para os destinatários se divide em duas direções. Uma ilustração destes conceitos é apresentada na figura 6.10.

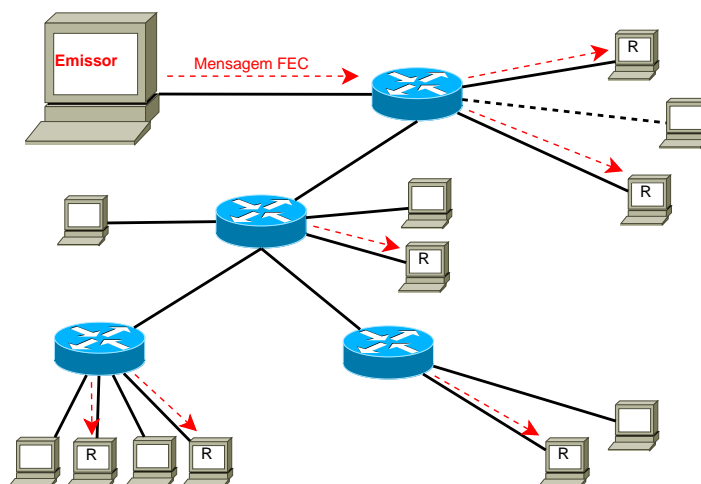


Figura 6.10: Visão de um Sistema Multicast

FEC é uma abordagem interessante para ser usada em multicast já que as mensagens se propagam por diversos canais, todos sujeitos a falhas e com características diferentes. Com isso, a probabilidade de ocorrer algum erro é maior do que em unicast. E na ocorrência de erro, pedir uma retransmissão seria muito custoso, pois a mensagem seria retransmitida para todo o grupo multicast. É, portanto, atraente utilizar codificação FEC e propiciar, a cada receptor, que recupere sua mensagem no caso de erro.

### 6.5.1.1 *Fatores Determinantes*

Do ponto de vista de desempenho, a mensagem é codificada apenas uma vez, no emissor e decodificada tantas vezes quanto receptores houverem.

Isso significa que o desempenho do decodificador assume um papel cada vez mais importante a medida que o número de clientes multicast aumenta. Pois a mensagem será decodificada por centenas, senão milhares de vezes.

Foi feita então uma previsão de qual seria o melhor algoritmo entre BCH e Reed-Solomon para ser utilizado em multicast.

### 6.5.1.2 *Previsão de Custo Computacional*

Valendo-se dos dados obtidos na primeira categoria de testes, podemos prever como seria o comportamento das codificações BCH e Reed-Solomon se utilizadas para multicast. O gráfico da figura 6.11 nos mostra que, apesar do Reed-Solomon ter desempenho inferior ao BCH para aplicações unicast, para multicast com um número elevado de inscritos as coisas mudam.

Este comportamento é explicado pelo fato do custo computacional do decodificador Reed-Solomon, na ausência de erros, ser mais leve que o BCH. Isso compensa o fato do codificador ser muito mais pesado mesmo nos piores casos (8 e 16 bytes).

Fica claro, portanto, que o Reed-Solomon preserva melhor os recursos do grupo multicast ao se ultrapassarem os 50 clientes para capacidade de correção de 8 bytes e 225 clientes para capacidade de correção de 16 bytes.

Entretanto, ao se introduzirem erros no canal de comunicação, espera-se novamente que o BCH apresente melhor desempenho, já que, neste caso, ele será mais leve também no decodificador.

Cada ponto deste gráfico foi gerado somando o tempo do codificador ao somatório de todos os tempos de decodificação de cada cliente. Não foram incluídos tempos de transmissão, pois o objetivo era analisar o custo de processamento.

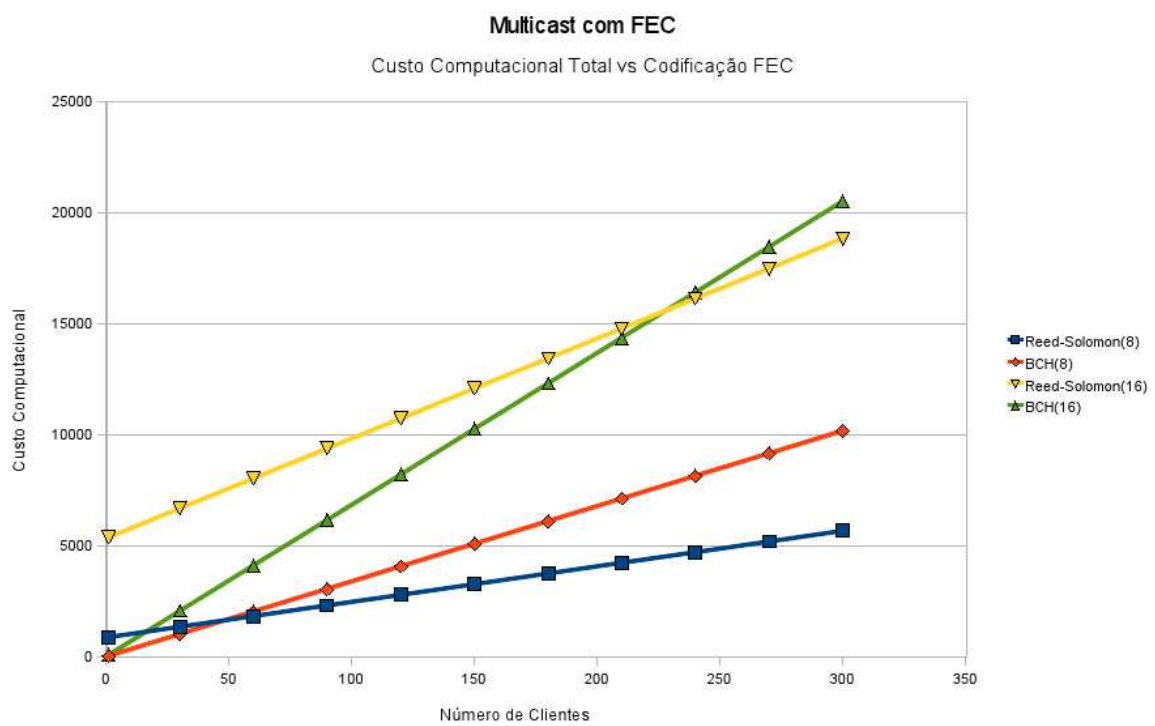


Figura 6.11: Previsão de Custo Computacional para Multicast na Ausência de Erros

## 7 CONCLUSÃO

Utilizar FEC nas transmissões de dados em redes de computadores, para evitar que pequenos erros diminuam ou até impeçam a transmissão de dados entre duas entidades comunicantes é, de fato, uma boa ideia. A utilização dos códigos para transmissão em rede se deu com sucesso, tanto para Reed-Solomon quanto para BCH. Foi possível traçar o perfil de cada implementação quanto ao seu desempenho no quesito uso de recursos de hardware, com resultados interessantes.

As taxas que o protocolo TCP consegue manter na presença de erros, mesmo que estes sejam poucos e extremamente simples, são muito baixas. Como mostrado, pequenas taxas de erro impõem quedas muito significativas nas taxas de transmissão deste protocolo. O uso de UDP instrumentado com FEC se mostrou mais eficiente que o TCP nos casos estudados, com relação as taxas de transmissão.

Em relação as implementações utilizadas, o decodificador BCH é ligeiramente mais eficiente que o Reed-Solomon, entretanto essa diferença pode ser desprezada e pode-se generalizar dizendo que os dois se comportam da mesma forma no quesito uso de recursos de hardware. Portanto, usar um ou outro não deve ser decidido levando em conta o desempenho dos decodificadores.

Com relação aos codificadores existe uma grande diferença: o Reed-Solomon se comporta de forma praticamente exponencial com o aumento da capacidade de correção do código, já o BCH pode ser dito linear. Do ponto de vista dos codificadores, deve-se preferir o BCH por apresentar maior eficiência.

No caso do uso prático de FEC, ressalta-se que quanto mais próxima do nível físico estiver a codificação FEC maior a proteção contra erros. E quanto mais próxima da camada de aplicação, menores serão as mudanças necessárias em protocolos e equipamentos existentes. Portanto, se deve escolher entre proteção e facilidade de implementação. A princípio, a camada em que a implementação melhor agrega estes dois quesitos é a de transporte, utilizando FEC para codificar seu *payload*.

Fica a sugestão de se fazer um estudo para o uso de FEC adaptativo no protocolo TCP. Adaptativo no sentido de escolher automaticamente a capacidade de correção dos códigos se baseando em dados fornecidos pelo receptor. O mecanismo de ACKs do TCP deve ser modificado para incluir estes dados. Nenhuma nova mensagem precisaria ser criada e, portanto, numa primeira análise, o sobrecusto desta modificação se daria apenas em tempo de CPU.

## REFERÊNCIAS

- Carissimi, Alexandre da S., J. Rochol, L. Granville. . **Redes de Computadores**. Bookman, 2009
- Carrasco, R. A., M. Johnston. . **Non-Binary Error Control Coding for Wireless Communication and Data Storage**. John Wiley I& Sons, Ltd, 2008
- Chen Zhi-kui. . **An adaptive FEC to protect RoHC and UDP-Lite H.264 video critical data**. Journal of Zhejiang University SCIENCE A, 7(5):910-918, 2006.
- Codificador e Decodificador BCH por Robert Morelos-Zaragoza **BCH3**. 1997, <http://www.eccpage.com/bch3.c>
- Codificador e Decodificador Reed-Solomon **RSCODE Versão 1.3**. <http://rscode.sourceforge.net/>
- Huffman, W. Cary,, Pless, Vera. . **Fundamentals of Error Correcting Codes**. Cambridge University Press, 2003
- Hui, Albert K.T., Chanson, Samuel T.. . **Design and Implementation of a Transparent Forward Error Correction Coding Daemon for Unreliable Transports**. 2001
- Moreira, J.C., Farrell, P. G.. . **Essentials of Error-Control Coding**. John Wiley I& Sons, Ltd, 2006.
- Nagle, John. . **Ford Aerospace and Communications Corporation Congestion Control in IP/TCP Internetworks**. IETF RFC 896, 1984.
- Præstholm, Steffen ., Schwefel, Hans-Peter,, Andersem, Søren Vang. . **A Comparative Study of Forward Error Correction and Frame Accumulation for VoIP over Congested Networks**. 2007
- Ran Yang, R.E. Kooij and R.D. Van der Mei. . **TCP Performance in case of bi-directional packet loss**. 2009
- Drebes, Roberto Jung,, Weber, Taisy **FIRMAMENT: Um Módulo de Injeção de Falhas de Comunicação para Linux** <http://sourceforge.net/projects/firmament/>
- Shannon, C. E.. **Communication in the presence of noise**. Proc. Institute of Radio Engineers, vol. 37, no.1, pp. 10-21, 1949.
- Shu Lin, Daniel J. Costello JR. . **Error Control Coding - Fundamentals and Applications**. Prentice Hall, 1983

Simulador de Redes **Network Simulator 2**. <http://www.isu.edu/nsnam/ns>

Siqueira, T. Fiss, B. Weber, R. Cechin, S. Weber, T. Univ. Fed. do Rio Grande do Sul . **Applying FIRMAMENT to test the SCTP communication protocol under network faults**. Test Workshop, 2009. LATW '09. 10th Latin American

Sweeney, P. . **Error Control Coding - From Theory to Practice**. John Wiley I& Sons, Ltd, 2002.

Wicker, Stephen B.. **Error Control Systems for Digital Communication and Storage**. Prentice Hall, 1994.

Young-Woo Kwon, Hyeyoung Chang, and JongWon Kim. . **Adaptive FEC Control for Reliable High-Speed UDP-Based Media Transport**. 2005