ANGELITA RETTORE DE ARAUJO ZANELLA

UM ESQUEMA PARA ENTREGA DE MENSAGENS CODIFICADAS EM REDES DTNS

Dissertação Apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Luiz Carlos Pessoa Albini

CURITIBA

2014

ANGELITA RETTORE DE ARAUJO ZANELLA

UM ESQUEMA PARA ENTREGA DE MENSAGENS CODIFICADAS EM REDES DTNS

Proposta de Dissertação de Mestrado Apresentada ao Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Luiz Carlos Pessoa Albini

CURITIBA

2014

Dedico esta dissertação ao meu esposo,

companheiro de todas as horas e eterna fonte de inspiração.

AGRADECIMENTOS

Agradeço a Deus, por ter me guiado nesta caminha e por me dar forças para seguir nos momentos difíceis.

Ao meu querido esposo, Fabrício, pela força, confiança e por estar sempre presente incentivando para seguir em frente. O seu apoio incondicional, seu companheirismo, amizade, paciência e compreensão foram fundamentais para que eu pudesse conquistar mais essa vitória. Aos meus pais, pelo exemplo de vida e à família pelo inestimável apoio.

Agradeço ao Professor Albini, por ter acreditado na minha capacidade e por aceitar orientar este trabalho. Obrigada por sua compreensão e imensa ajuda durante todo o processo.

Aos colegas do IFC, especialmente aos meus coordenadores e diretores, pelo incentivo, pela força e ajuda, organizando meus horários de trabalho, acreditando no meu potencial e incentivando para a continuidade do trabalho. Obrigada pela confiança e apoio durante todos esses meses.

Aos amigos do mestrado, pelos momentos e experiências compartilhadas, pelas palavras de conforto e de carinho. Agradeço ao Rafael, Saulo, Ricardo, Robson, Yuri e Rodrigo por compartilharem seus conhecimentos e perspectivas, pela ajuda e incentivo. Em especial, agradeço à Cinara, pela imensurável ajuda nos momentos mais importantes e por participar tão alegremente dos momentos especiais.

Aos amigos de caminhada, que mais do que momentos, compartilharam suas casas. Obrigada Naiára e Elisa pelo carinho e confiança. Em especial, agradeço à Eloisa, pelo agradável convívio, pelas inúmeras discussões acerca dos mais variados assuntos e pela amizade, que tornou essa caminhada mais suave e animada.

Aos funcionários do programa de pós-graduação, pela disponibilidade, simpatia de gentileza. Obrigada pela ajuda!

Agradeço também ao FUMDES, pelo apoio financeiro e à Universidade Federal do Paraná, pela oportunidade de realizar o sonho do mestrado.

A verdadeira viagem do
descobrimento não consiste em
procurar novos horizontes, mas em
ver com novos olhos.

Marcel Proust

RESUMO

As redes tolerantes a atrasos e desconexões (DTN) são formadas por nós móveis ad hoc cujas características de mobilidade impõem sérias restrições para o encaminhamento e entrega de mensagens. Uma característica importante das DTNs é a conectividade intermitente, resultado das frequentes desconexões causadas pela mobilidade e topologia esparsa. Nesses cenários, a entrega de mensagens torna-se um desafio, pois elas podem ser retidas por longos períodos ou nunca serem entregues ao destinatário. As propostas existentes para melhorar a taxa de entrega de mensagens nesses cenários alteram os protocolos de roteamento, fazendo verificação de integridade a cada salto. Isso resulta em sobrecarga no roteamento e em um custo computacional muitas vezes impraticável. Este trabalho propõe um esquema, denominado EMCOD, que visa reduzir o tempo para entrega de mensagens e a sobrecarga de processamento, em redes caracterizadas por longos atrasos e que sofrem perdas de pacotes. O esquema utiliza codificação de rede e intercalação de dados para criação de mensagens menores que são encaminhadas pela rede. A reconstrução dos dados originais é feita a partir da recepção de algumas mensagens, não sendo necessário aguardar o recebimento de todas. Utilizando Reed-Solomon para codificação de dados, o EMCOD é capaz de reduzir o tempo para recuperação dos dados originais em mais de 50%, em cenários que sofrem altas taxas de perdas de pacotes. A sobrecarga computacional adicionada pelo processo de codificação é compensada pela capacidade de recuperação dos dados originais, sem que seja necessário reenviar as mensagens perdidas. Devido a essa capacidade, é possível reduzir a sobrecarga na rede em mais de 60%, em cenários que possuem altos índices de perda de mensagens. O EMCOD altera a estrutura da camada de Agregação, mas não interfere no funcionamento das demais camadas. Assim, é possível realizar o roteamento das mensagens através de nós que não implementem o esquema proposto.

Palavras-chave: DTN, codificação de rede, redução de tempo, baixa sobrecarga.

ABSTRACT

Delay and Disruption Tolerant Networks (DTN) are made up of mobile ad hoc nodes, and it is exactly that mobility that imposes major message routing and delivery restrictions. Another important characteristic of DTNs is its intermittent connectivity, resulting from frequent disconnections, which in turn are caused by mobility and scattered topologies. In these scenarios, message delivery becomes a challenge, considering they can be detained for long periods or never get delivered to its destination. Existing solutions to improve message delivery rates in such scenarios modify routing protocols to perform integrity verification with each hop. This results in routing overhads and, too often, unrealistic processing costs. This research proposes a schema, named EMCOD, that decreases message delivery times, and also minimizes processing overhads in networks burdened by long delays and packet losses. The schema uses data encoding and interleaving to create smaller messages, which are then routed through the network. The original data is then reassembled from some of the messages received, without the need to wait for the retrieval of all messages. Using Reed-Solomon codes to encode the data, EMCOD is capable of reducing original data retrieval times by more than 50%, in scenarios with high packet loss rates. The processing overhad resulting from the encoding procedures is offset by the data retrieval capabilities, without the need to resend lost messages. This capability effectively decreases network overhads by more than 60%, in scenarios subject to high message loss rates. EMCOD modifies the structure of the Bundle Layer, without interfering with the remaining layers, making it possible to route the messages through nodes that don't implement the proposed schema.

Keywords: DTN, network encoding, time reduction, low overhead.

SUMÁRIO

RESUMO **iv**

ABSTRACT **v**

LISTA DE FIGURAS **ix**

LISTA DE TABELAS **x**

LISTA DE ABREVIATURAS E SIGLAS **xi**

NOTAÇÃO **1**

1 INTRODUÇÃO **1**

1.1 Problema 3

1.2 Objetivo 4

1.2.1 Objetivo Geral 4

1.2.2 Objetivos Específicos 5

1.3 Estrutura do Trabalho 5

2 ROTEAMENTO EM REDES DTN **6**

2.1 Roteamento em Cenários Determinísticos 6

2.2 Roteamento em Cenários Estocásticos 8

2.2.1 Roteamento Epidêmico 8

2.3 Roteamento Estocástico com Técnicas de Codificação 9

2.3.1 Protocolo de Roteamento vCF 10

2.3.2 Protocolo NER-DRP 11

3 CÓDIGOS CORRETORES DE ERRO **14**

3.1 Códigos de Correção Direta de Erros 15

3.1.1 Códigos de Bloco 16

3.1.2 Códigos Convolucionais 17
3.2 Códigos Reed-Solomon 19
3.2.1 Processo de Codificação 20
3.2.2 Processo de Decodificação 22
3.2.3 Complexidade Computacional 28

4 UM ESQUEMA PARA ENTREGA DE MENSAGENS CODIFICADAS EM REDES 33

4.1 Visão Geral da Subcamada CCD 34
4.2 Construção das Mensagens 35
4.3 Codificação de Dados 38
4.4 Intercalador 38
4.5 Funcionamento da Subcamada CCD com Reed-Solomon 39
4.6 Recepção das Mensagens e Reconstrução dos Dados Originais 40

5 LIMITES ANALÍTICOS 43

6 ANÁLISE DE DESEMPENHO 47

6.1 Sobrecarga de Mensagens 48
6.2 Capacidade de Recuperação 52
6.3 Sobrecarga de Energia 54
6.4 Atraso para Entrega das Mensagens 56
6.5 Análise dos Resultados 59

7 ANÁLISE DE DESEMPENHO CONSIDERANDO PERDAS 60

8 CONSIDERAÇÕES FINAIS 65

8.1 Entrega de Mensagens Codificadas em Redes DTN 65
8.2 Trabalhos Futuros 67

BIBLIOGRAFIA 68

Apêndices 68

Apêndice A CAMPOS FINITOS **69**

Apêndice B CÓDIGOS-FONTE **72**

B.1 Desempenho em ambientes desconsiderando perdas 72

B.2 Desempenho em ambientes com perda de dados 79

LISTA DE FIGURAS

1.1 A Camada de Agregação 2

2.1 Entrega de mensagens com perda de pacotes [**?**]. 11
2.2 Processo de divisão da mensagem na origem [**?**]. 12

3.1 Estrutura de uma mensagem gerada pelo código de Hamming 14
3.2 Criação dos códigos de bloco 16
3.3 Codificador convolucional com n=2 e K=3 18
3.4 Estrutura de um bloco de dados codificado com Reed-Solomon 20

4.1 Divisão da Camada de Agregação 33
4.2 Divisão da subcamada CCD 34
4.3 Transmissão das mensagens até o destinatário 35
4.4 Composição da Matriz fxb 36
4.5 Processo de construção da mensagem 36
4.6 Agrupamento de unidades 37
4.7 Processo de Intercalação e criação da unidadeCCD 39
4.8 Reconstrução dos blocos 41

6.1 Quantidade de mensagens criadas 49
6.2 Número de mensagens adicionais 51
6.3 Capacidade de recuperação de mensagens utilizando 53
6.4 Sobrecarga de Energia 56
6.5 Comparação das sobrecargas de energia 57
6.6 Atraso para entrega de mensagens 58

7.1 Quantidade de mensagens enviadas 61
7.2 Atraso para entrega de mensagens 62

LISTA DE TABELAS

3.2.1 Complexidade Computacional para construção de blocos 30

3.2.2 Custo computacional dos operadores GF 31

3.2.3 Custo computacional dos operadores dos códigos RS para m=8 31

3.2.4 Energia consumida pelos códigos analisados 32

6.1 Códigos RS utilizados para análise de desempenho 48

6.1.1 Tamanho das mensagens criadas pelos códigos Reed-Solomon 49

7.1 Capacidade de recuperação de mensagens 60

A.0.1Elementos do campo denotado por $GF(2^3)$ 71

CAPÍTULO 1

INTRODUÇÃO

Nas últimas décadas, novas tecnologias de rede têm sido desenvolvidas para dar suporte à comunicação em cenários que ultrapassam a abrangência da Internet tradicional [**?** **?** **?** **?** **?** **?** **?**]. Os novos cenários envolvem ambientes em que não é possível instalar uma infraestrutura que permita a comunicação entre os nós[**?** **?** **?** **?**]. Nesses ambientes, os nós podem formar redes móveis ad hoc para se comunicar.

Alguns ambientes formam cenários desafiadores para a criação de redes, devido às restrições impostas pelo ambiente ou às características dos nós. Exemplos desses cenários são as redes submarinas, comunicação espacial e em áreas rurais, cujos nós possuem densidade esparsa, restrições quanto ao fornecimento de energia, comunicação intermitente ou sofrem falhas frequentes [**?** **?** **?**]. Essas redes formam um subgrupo das redes móveis, as Redes Tolerantes a Atrasos e Desconexões (DTN, *Delay and Disruption Tolerant Networks*).

Khabbaz [**?**] define DTN como uma "arquitetura de sobreposição"criada para operar sobre protocolos de redes heterogêneas que podem não ter conectividade permanente. Elas são formadas por dispositivos móveis cuja comunicação pode sofrer desconexões frequentes, longos atrasos e possuir altas taxas de perdas de pacotes. São redes que não possuem um caminho ativo entre origem e destino, o que é um desafio para os protocolos tradicionais. Protocolos como o TCP são inviáveis nesse tipo de rede, devido à ausência de um caminho fim-a-fim [**?** **?**].

Para possibilitar a comunicação em redes DTNs, utiliza-se comutação por mensagens e armazenamento persistente [**?** **?** **?**]. A utilização dessas técnicas não exige o estabelecimento de nenhum circuito entre origem e destino. Ao invés disso, a mensagem completa é encaminhada para o próximo salto, que a armazena e tem sua custódia até o seu repasse para o salto seguinte[1]. A ação se repete até que a mensagem seja entregue para

[1]Um nó só deixa de possuir a custódia de uma mensagem quando outro nó aceita o repasse dessa custódia.

o nó destinatário. O processo de encaminhamento e custódia da mensagem é realizado por uma nova camada, denominada camada de *Agregação* (*Bundle layer*), destacada na Figura 1.1, que passa a ser responsável pela comunicação salto-a-salto. Para garantir a interoperabilidade com redes que utilizam a pilha de protocolos TCP/IP, a camada de Agregação foi criada em um nível imediatamente acima da camada de Transporte.

Figura 1.1: A Camada de Agregação

As mensagens encaminhadas pelas redes DTNs são denominadas Unidade de Dados de Aplicação (ADU, *Application Data Unit*) [**?**]. A camada de Agregação transforma as ADUs em Unidade de Dados de Protocolo (PDU, *Protocol Data Unit*), denominados agregados (*bundles*), os quais são encaminhados e armazenados pelos nós das DTNs. Os agregados possuem todas as informações necessárias para encaminhamento da mensagem até o destinatário.

Cada agregado é composto por blocos com formato definido e tamanhos variados, semelhantes aos cabeçalhos, mas que não precisam estar, obrigatoriamente, no início da unidade de dados. Dois blocos são obrigatórios, sendo apenas um definido como primário: o primeiro contém informações para o encaminhamento do agregado até o destino, e o seguinte contém a carga útil de dados. Além desses, podem existir os blocos de extensão, que contêm campos adicionais.

Uma vez que em redes DTN nem todos os nós estão ativos a todo instante, a transmissão dos agregados é realizada quando ocorre um contato entre dois nós vizinhos. Um contato é definido como uma ocasião favorável para a troca de dados. Oliveira et. al.[**?**] classifica os contatos de acordo com as características de conexão:

- Contatos persistentes: estão sempre disponíveis e ocorrem em redes cujas desco-

nexões são esporádicas.

- Contatos sob demanda: precisam de uma ação para que sejam instanciados e, após isso, permanecem ativos até que o contato seja encerrado. Podem ocorrer em redes de sensores, por exemplo, em que um grupo de nós fica ativo, enquanto outro grupo de nós está dormindo para poupar energia.

- Contatos programados: são contatos estabelecidos em horários e com duração previamente agendados. Ocorrem em redes cujos nós mantêm um padrão de movimento e periodicamente se aproximam o suficiente para permitir a troca de mensagens.

- Contatos previsíveis: é possível prever o horário e a duração do contato, baseado no histórico de contatos prévios, mas não é possível ter a certeza de que o contato ocorrerá conforme o previsto. Podem ocorrer em comunicações rurais, em que a comunicação entre duas regiões pode ser feita através de um nó intermediário, como um ônibus, por exemplo.

- Contatos oportunistas: o contato ocorre ao acaso, devido à aproximação dos nós. Não há qualquer informação sobre os nós conectados e a conexão pode ser interrompida a qualquer momento. Ocorre, geralmente, entre nós autônomos.

Em redes DTNs, a entrega de uma mensagem corrompida é um grande problema, pois não é possível estabelecer uma conexão confiável. A confiabilidade deve ser implementada no nível de aplicação, o que exige que o destinatário envie uma mensagem ao remetente solicitando sua retransmissão. Em diversos cenários, todo esse processo pode causar excessivos atrasos que inviabilizam a comunicação. Assim, estratégias que permitam a recuperação da mensagem corrompida, tornam-se uma alternativa necessária.

1.1 Problema

Buscando evitar a entrega de mensagens corrompidas e a perda de mensagens durante a transmissão, algumas propostas utilizam sistemas de codificação [**?** **?** **?** **?** **?**]. Uma dessas propostas é o vCF [**?**], um protocolo de roteamento que utiliza codificação linear

aliada ao protocolo de roteamento Epidêmico. Outro protocolo de roteamento proposto com o mesmo objetivo é o NER-DRP [**?**], que une o protocolo Epidêmico ao Reed-Solomon para recuperar possíveis corrupções na mensagem. Ambas as propostas dividem a mensagem original em blocos que são codificados e realizam a recuperação de erros a cada salto. Apesar dessas propostas realizarem a entrega íntegra da mensagem, elas fazem alterações em protocolos de roteamento, restringindo a utilização dos novos protocolos a cenários específicos.

Visando garantir a entrega íntegra de mensagens com o menor tempo possível, foi elaborado um esquema que, além de entregar a mensagem íntegra ao destinatário, gera o mínimo de sobrecarga na rede visando reduzir o impacto causado. Esta dissertação propõe um esquema que utiliza codificação de rede, divisão da mensagem em partes menores e o seu encaminhamento através de caminhos possivelmente diferentes. O esquema proposto não altera o roteamento, trabalhando na camada de Agregação. Desta forma, não está atrelado a um protocolo de roteamento específico e pode ser aplicado a diferentes cenários. Em redes que sofrem longos atrasos e perda de mensagens, esse esquema reduz o tempo para a entrega da mensagem com uma sobrecarga de processamento menor que a de outras propostas similares.

1.2 Objetivo

Este trabalho visa desenvolver um esquema que reduza o tempo para entrega de mensagens e gere uma sobrecarga reduzida em redes DTN. Para alcançar esse objetivo é utilizada codificação de rede associada a uma nova estratégia para construção das mensagens. O novo esquema é adicionado à camada de Agregação e permite ao destinatário recuperar a mensagem original, mesmo que parte dela tenha sido perdida.

1.2.1 Objetivo Geral

O objetivo deste trabalho é desenvolver um esquema que reduza o tempo para entrega de mensagens e gere uma sobrecarga de processamento reduzida em redes que sofrem longos

atrasos quando ocorre perda de pacotes.

1.2.2 Objetivos Específicos

- Reduzir a taxa de perda de pacotes em redes DTN;
- Diminuir o atraso para entrega da mensagem ao destinatário;
- Permitir a recuperação da mensagem mesmo que ocorra perda de mensagens;
- Minimizar a sobrecarga computacional;
- Reduzir o tráfego de rede.

1.3 Estrutura do Trabalho

Esta dissertação está organizada em oito capítulos. O Capítulo 2 apresenta os fundamentos de redes DTNs, abordando principalmente o roteamento de mensagens. O Capítulo 3 fornece uma visão geral sobre os códigos de correção de erros e suas principais categorias, com ênfase nos códigos Reed-Solomon. O Capítulo 4 descreve o esquema, denominado EMCOD, e a criação da subcamada CCD. O Capítulo 5 apresenta a a prova da redução do tempo para entrega da mensagem ao destinatário e a baixa sobrecarga gerada pelo processo. O Capítulo 6 mostra a análise de desempenho da subcamada utilizando Reed-Solomon como código de correção de erros e o Capítulo 7 analisa o desempenho da subcamada considerando perdas de mensagens. Por fim são apresentadas as considerações finais no Capítulo 8.

CAPÍTULO 2

ROTEAMENTO EM REDES DTN

Devido às características peculiares das redes DTNs é comum não existir um caminho fim-a-fim. Por esse motivo, um dos desafios dessas redes é a realização de um roteamento eficiente, que considere suas características, tais como longos atrasos e frequentes desconexões. Os protocolos desenvolvidos para as redes convencionais e para as redes sem fio ad hoc não são adequados para redes DTNs, por isso, novos protocolos estão sendo desenvolvidos [**?**]. Como existem vários tipos de DTN, devem ser desenvolvidos protocolos que considerem as particularidades de cada uma delas. Assim, esses protocolos podem ser divididos de acordo com o cenário, que pode ser determinístico ou estocástico [**?**].

2.1 Roteamento em Cenários Determinísticos

Em cenários determinísticos, pressupõe-se que algumas informações são conhecidas e, a partir delas, é possível desenvolver protocolos que consideram as peculiaridades de cada cenário. As diferentes informações são classificadas e modeladas por oráculos, que são uma abstração para o tipo de informação ao qual estão relacionados. Um oráculo fornece uma informação específica ou um conjunto delas.

Oliveira [**?**] descreve quatro grupos distintos de oráculos. O *Oráculo de Resumo de Contatos* contém um resumo sobre as informações não variantes de um determinado nó. Ele é capaz de informar, por exemplo, qual é o tempo médio necessário para o estabelecimento de um novo contato. O *Oráculo de Contatos* possui informações mais detalhadas sobre cada um dos contatos estabelecidos pelo nó, como o tempo de duração do contato. Já o *Oráculo de Ocupação* informa qual o nível de ocupação dos *buffers* de transmissão dos nós, o que pode ser útil para maximizar o desempenho da rede. Por fim, o *Oráculo de Demanda de Tráfego* conhece a demanda de tráfego dos nós da rede a cada instante de tempo. Essa informação é baseada no conhecimento sobre todas as mensagens

que os nós desejam encaminhar.

Tomando como base as informações obtidas através dos oráculos, diversos protocolos foram desenvolvidos para cenários determinísticos. Entre os principais protocolos desenvolvidos para cenários determinísticos, Oliveira [**?**] cita:

- Primeiro Contato (FC, *First Contact*) [**?**]: O FC é um protocolo muito simples, que não utiliza nenhuma informação e não realiza nenhum processamento adicional para o encaminhamento da mensagem. Seu funcionamento está baseado no encaminhamento da mensagem a um de seus vizinhos, independentemente de quais sejam suas características. Ao utilizar esse protocolo, o nó simplesmente encaminha a mensagem para o primeiro nó com o qual venha a estabelecer um contato.

- Atraso Mínimo Esperado (MED, *Minimum Expected Delay*) [**?**]: Com a utilização de informações do Oráculo de Resumo de Contatos pode-se determinar qual o tempo médio restante para que um novo contato seja estabelecido. A partir dessa informação é possível calcular uma rota, minimizando o tempo médio de espera, mesmo que o contato demore um pouco a ser estabelecido.

- Entrega Mais Rápida (ED, *Earliest Delivery*) [**?**]: Através do Oráculo de Contatos, o ED obtém informações relacionadas ao estabelecimento de contatos, sendo possível saber em que momento um determinado enlace estará disponível. A partir dessa informação, o ED determina qual caminho tem maior probabilidade de entregar a mensagem mais rapidamente.

- Entrega Mais Rápida com Todas as Filas (EDAQ, *Earliest Delivery with All Queues*) [**?**]: O EDAQ utiliza-se de informações sobre os contatos e os tamanhos das filas de transmissão de dados para determinar o melhor caminho. As informações sobre as filas de transmissão agregam mais eficiência e permitem determinar o caminho mais rápido. No entanto estas informações são difíceis de serem obtida na prática, o que torna o protocolo inviável para muitos cenários.

Os protocolos criados para cenários determinísticos não foram utilizados neste estudo, pois neste não é considerada a existência de informações sobre a rede. Por esse motivo,

considerou-se suficiente a apresentação geral de seus conceitos. Para mais informações sobre os protocolos para cenários determinísticos, o leitor deve seguir estas referências [**?** **?** **?**].

2.2 Roteamento em Cenários Estocásticos

Em cenários estocásticos são consideradas pouca ou nenhuma informação adicional sobre a rede, pois em muitos casos é difícil possuí-las ou sua obtenção gera sobrecarga. Para esses casos foram criados os protocolos baseados em inundação e replicação [**?** **?** **?**], os quais utilizam pouca [**?** **?**] ou nenhuma [**?** **?**] informação sobre a rede para encaminhamento das mensagens. A ausência de informações sobre a rede dificulta o roteamento e impossibilita a determinação de "melhores" rotas.

Na tentativa de melhorar o desempenho do roteamento, alguns protocolos [**?** **?** **?**] utilizam a troca de informações entre os nós durante um contato, buscando realizar estimativas que auxiliem na determinação de uma rota [**?** **?** **?** **?**]. Outros buscam determinar um padrão de mobilidade e o controle de movimento para maximizar a eficiência [**?** **?**]. Há, ainda, os que incorporam técnicas de codificação para melhorar a taxa de entrega e reduzir a sobrecarga na rede [**?** **?**]. Em todos os casos, o desempenho do protocolo depende do cenário em que é utilizado.

Uma comparação realizada entre alguns dos protocolos desenvolvidos para cenários estocásticos [**?**] mostra que aqueles baseados em inundação alcançam maiores taxas de entrega, os baseados em conhecimento consomem menos recursos e os baseados em codificação geram menor sobrecarga. Entretanto, cada técnica apresenta resultados melhores ou piores de acordo com o cenário.

2.2.1 Roteamento Epidêmico

Em muitos cenários que envolvem redes DTNs, a característica mais marcante é a baixa densidade de nós, o que causa sérios problemas de comunicação, uma vez que não existe um caminho que possibilite o encaminhamento da mensagem entre origem e destino. A

densidade esparsa da rede, aliada à ausência de informações acerca dos nós e da rede como um todo, dificultam o processo de roteamento das mensagens. Sobretudo para cenários com essas características, o roteamento Epidêmico [**?**] apresenta bons resultados [**?**].

O protocolo Epidêmico foi um dos primeiros a serem desenvolvidos para redes DTNs [**?**]. Criado para cenários estocásticos, considera a entrega eventual de mensagens e as encaminha para destinos arbitrários [**?**], partindo do princípio de que apenas algumas pequenas porções da rede permanecem conectadas [**?**]. Seu principal objetivo é maximizar a probabilidade de entrega de uma mensagem ao destinatário. Contudo, pode-se adicionar a esse propósito a intenção de minimizar a latência [**?**].

O protocolo de roteamento Epidêmico não utiliza informações sobre topologia ou conectividade da rede. Seu funcionamento é baseado no encaminhamento das mensagens para o próximo salto, fazendo com que cada nó mantenha uma cópia de todas as mensagens sob sua custódia. A mobilidade dos nós é suficiente para que as mensagens sejam entregues, porém podem ocorrer elevados índices de retransmissões, pois não são considerados limites quanto ao tamanho da memória de armazenamento [**?**].

Devido às características do protocolo, considera-se adequada a sua utilização na avaliação do esquema proposto. Por ser utilizado apenas como protocolo de roteamento neste trabalho, considera-se suficiente apresentar uma visão geral sobre o seu funcionamento. Uma descrição mais detalhada pode ser obtida em [**? ? ? ?**].

2.3 Roteamento Estocástico com Técnicas de Codificação

Uma vez que em redes DTNs não existe uma conexão fim-a-fim, não é possível ter certeza de que uma mensagem será recebida pelo destinatário e, ainda que a mensagem seja recebida, é possível que ela esteja corrompida. Como essas redes não permitem a utilização de protocolos de transporte com garantia de entrega, é muito difícil detectar a ocorrência de erros durante o encaminhamento.

Alguns dos esforços existentes para resolver esse problema empregam técnicas de codificação. Essas técnicas têm sido utilizadas em sistemas de comunicação, visando corrigir erros decorrentes do processo de transmissão de dados a partir da adição de *bits* de pari-

dade. Devido à sua eficiência, essas técnicas passaram por um grande avanço nas últimas décadas, resultando na criação de códigos cada vez mais robustos. O capítulo 3 apresenta uma visão geral sobre os códigos de correção de erros.

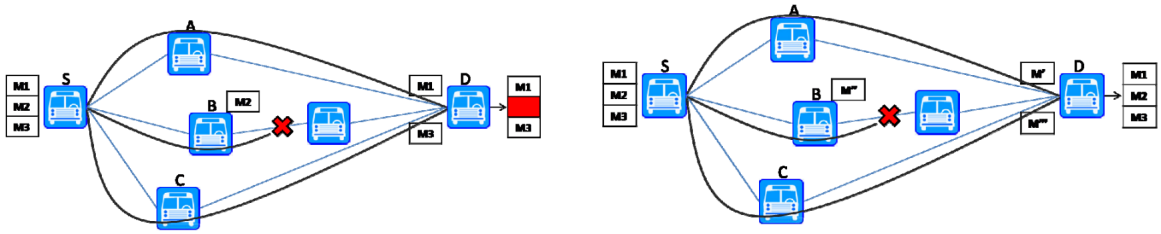
Ao considerar os benefícios proporcionados por esses códigos, Chung, Li e Liao propuseram um protocolo de roteamento, denominado vCF [**?**], que utiliza codificação linear para redes DTNs. O objetivo dos pesquisadores é maximizar a taxa de entrega, sem prejudicar a reconstrução da mensagem original. Já Li *et al.* [**?**] optam por utilizar os códigos Reed-Solomon (RS) para resolver o mesmo problema e utilizam o protocolo de roteamento Epidêmico como base para a criação de um novo protocolo, chamado NER-DRP.

2.3.1 Protocolo de Roteamento vCF

O trabalho publicado por Chung, Li e Liao em 2010 [**?**] trata do problema de entrega confiável e otimização dos recursos em redes veiculares formadas por ônibus que transitam em áreas rurais. Essas redes são caracterizadas pela conectividade intermitente e sofrem longos atrasos para a entrega de pacotes.

A solução proposta pelos autores divide a mensagem em blocos menores como alternativa para otimizar o uso dos recursos de rede. Essa divisão pode inviabilizar a reconstrução da mensagem no destinatário devido a falta de alguns blocos, como mostra a Figura 2.1(a) [**?**]. Para resolver o problema os pesquisadores propõem um novo protocolo que utiliza a codificação linear no processo de construção dos blocos, o vCF. Ao dividir a mensagem em blocos, o protocolo codifica-os de forma que seja possível reconstruir a mensagem original a partir de um subconjunto de blocos, o que admite a perda de alguns blocos durante o processo de roteamento, como mostrado na Figura 2.1(b) [**?**].

As mensagens recebidas pelo vCF são divididas em blocos. A cada bloco é atribuído um coeficiente $\vec{g}$ único, pertencente aos corpos finitos, que é utilizado para codificação do bloco. Um nó intermediário recebe uma quantidade de blocos codificados que permite a recuperação da mensagem. Caso a mensagem seja enviada para mais de um nó, seus blocos devem ser codificados com coeficientes diferentes. O processo de codificação é realizado a cada salto até que a mensagem seja recebida pelo destinatário.



(a) Entrega de blocos sem codificação de rede (b) Entrega de blocos com codificação de rede

Figura 2.1: Entrega de mensagens com perda de pacotes [**?**].

Ao receber os blocos, o destinatário utiliza Eliminação Gaussiana [**?**] para verificar se os pacotes são linearmente independentes e descartar blocos duplicados que tenham sido recebidos. Uma vez descartados os blocos duplicados, o vCF verifica se foi recebida uma quantidade suficiente de pacotes para reconstruir a mensagem original. Em caso negativo, o destinatário aguarda o recebimento de mais blocos.

Resultados obtidos através de simulação demonstram que o protocolo possui taxa de entrega em média 67% superiores ao Epidêmico. Em relação às taxas de atraso, o vCF é aproximadamente 32% mais eficiente que o Epidêmico. Porém, codificar dados a cada salto gera sobrecarga computacional desnecessária sobre a rede. Além disso, a codificação linear não é o método mais utilizado para codificação de rede, pois apresenta resultados inferiores se comparados a outros códigos amplamente utilizados. Por este motivo, o vCF não será utilizado para comparação com o esquema proposto por este trabalho.

2.3.2 Protocolo NER-DRP

Li *et. al.* [**?**] propõem a utilização de um método para recuperação de erros em redes com enlaces unidirecionais, o qual denominaram Recuperação de Erros de Camada de Rede (NER, *Network-layer Error Recovery*). O método divide a mensagem em blocos e os codifica utilizando um código de correção de erros, como Reed-Solomon (RS). O código de correção de erros é utilizado durante o processo de roteamento para correção de possíveis erros nos blocos. Devido ao processo adicional de correção de erros, foi necessário realizar modificações no protocolo de roteamento originalmente utilizado, o Epidêmico, o que resulta em um novo protocolo denominado NER-DRP.

O processo de encaminhamento da mensagem inicia com a sua divisão em pacotes de dados[1] e a subdivisão de cada pacote em blocos. Cada bloco é formado por uma parte de dados e uma parte de paridade, que é utilizada para verificação do bloco, como mostra a Figura 2.2 [**?**]. A paridade é gerada pelo código corretor de erros, sendo que os autores optaram pela utilização de um código Reed-Solomon, o RS(127,117)[2]. O RS será detalhado na Seção 3.2.

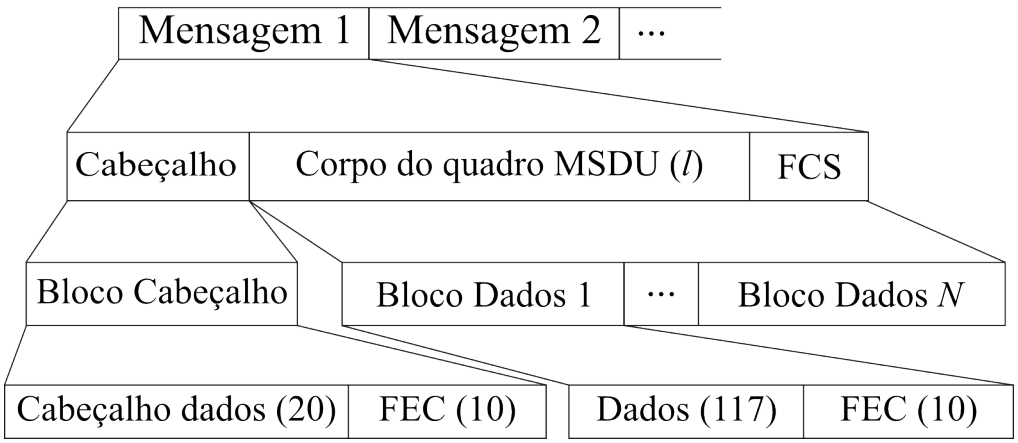


Figura 2.2: Processo de divisão da mensagem na origem [**?**].

Nos nós intermediários, o NER é responsável pela recuperação e correção de erros em pacotes possivelmente corrompidos. A paridade adicionada a cada bloco permite aos nós intermediários a correção de cada bloco individualmente e também dos demais blocos. A recuperação dos demais blocos é feita a partir de uma recombinação de múltiplas cópias, o que é feito em uma subcamada da camada de Rede e diferencia o protocolo dos demais. Ao receber um bloco, a camada adicional não o descarta em caso de detecção de corrupção, mas o armazena para recombiná-lo com outros blocos do mesmo pacote em uma tentativa de recuperar o pacote corrompido.

O NER-DRP apresenta um desempenho melhor quando comparado ao protocolo Epidêmico original, alcançando taxas de entrega aproximadamente 68% superiores ao Epidêmico, com redução da média de atraso fim-a-fim em cerca de 3%. Os resultados foram obtidos comparando os espaços de armazenamento por pacote, variações quanto à

[1]Denominação utilizada pelos autores

[2]Um código Reed-Solomon é definido pela expressão RS(n, k) em que n representa o número de *bytes* que constituem o bloco codificado e k o número de *bytes* provenientes da mensagem original.

mobilidade dos nós e raio de transmissão.

Entretanto, o processo de detecção e correção de erros efetuado nos nós intermediários pode ser indesejado em diversas redes DTNs, pois o processamento adicional acarreta em sobrecarga computacional e pode atrasar o encaminhamento das mensagens. Além disso, em cenários com topologia esparsa, um nó intermediário com blocos corrompidos pode nunca conseguir recuperar a mensagem original, por falta de blocos íntegros para recombinar. Por não conseguir recombinar os blocos e realizar a correção do erro, a mensagem pode nunca ser encaminhada, resultando em subutilização da rede, degradação do desempenho além do desperdício de memória. Ademais, essas redes geralmente estabelecem enlaces bidirecionais e o processo de verificação e correção de erros salto-a-salto é realizado pela camada de Enlace.

CAPÍTULO 3

CÓDIGOS CORRETORES DE ERRO

As transmissões de dados e o seu armazenamento através de sistemas computacionais estão sujeitas a erros introduzidos por ruídos ou falhas no sistema [**?** **?** **?** **?**]. Esses erros corrompem a informação, gerando problemas durante sua decodificação [**?**]. Na tentativa de resolver esse problema e permitir a recuperação da informação original após a sua exposição a erros, Richard W. Hamming propôs, em 1950, um Código de Correção de Erros (ECC, *Error Correcting Code*) [**?**]. Os ECCs são caracterizados pela inserção de redundância, responsável pela detecção e correção de erros introduzidos na mensagem.

O código proposto por Hamming adiciona m *bits* de paridade em um bloco de k *bits* de dados, gerando a transmissão de uma mensagem de tamanho $n = m + k$ *bits*. Sua capacidade de correção depende da sua distância mínima, sendo possível detectar até $d_{min} - 1$ erros e de corrigir até $\frac{d_{min}-1}{2}$ erros [**?**]. A distância mínima[1] ou distância da Hamming refere-se ao número de posições em que os elementos de um par de vetores são diferentes [**?**] (d_{min}). Em virtude da forma como o código é construído, a capacidade de correção está diretamente ligada ao tamanho da mensagem original e à quantia de *bits* de paridade adicionados. A Figura 3.1 representa a estrutura da mensagem gerada pelo código de Hamming. Neste exemplo, é possível detectar dois *bits* e corrigir um *bit* corrompido.

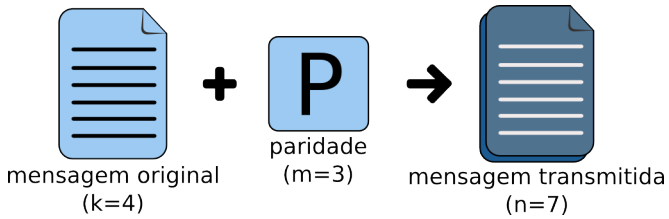


Figura 3.1: Estrutura de uma mensagem gerada pelo código de Hamming

A partir do desenvolvimento do primeiro ECC, buscou-se a melhoria das técnicas

[1]Mais detalhes sobre distância mínima de Hamming podem ser obtidos em [**?**]

visando à criação de códigos mais eficientes. Os avanços na codificação foram possíveis após a publicação dos estudos de Shannon [**?**], que lançou as bases da Teoria dos Códigos e possibilitou um desenvolvimento acelerado dos sistemas de armazenamento de dados em memória de acesso aleatório; em discos rígidos, usando a técnica RAID; discos compactos (CD) ou digitais (DVD); compressão de textos e imagens; reprodução de áudio e vídeo em computador; comunicação de voz sobre IP; comunicação móvel; vídeo digital em banda larga e transmissão de dados através de rede de telefonia (DSL). Nos sistemas modernos de comunicação móvel, os ECCs estão presentes principalmente na codificação de canal, permitindo não apenas a correção e detecção de erros, mas também a melhoria na eficiência de uso do canal.

Os ECCs relacionados à codificação de canal podem ser classificados em duas categorias: Código de Correção Direta de Erros (FEC, *Forward Error-Correction*) e Esquema de Solicitação e Repetição Automática (ARQ, *Automatic-Repeat reQuest*) [**?**]. O primeiro aplica a estratégia de adição de *bits* redundantes em uma palavra código e a utilização desta palavra tanto para detecção quanto para correção de erros, sem a necessidade de processamento posterior. O segundo utiliza uma palavra código apenas para a detecção de erros e, caso a informação recebida contenha erros, é necessário que a informação original seja retransmitida [**?**]. Esta dissertação não apresenta maiores detalhes sobre o ARQ, porém o leitor pode encontrá-las em [**?**].

3.1 Códigos de Correção Direta de Erros

Os códigos de correção direta de erros são amplamente utilizados para detecção e correção de erros em transmissões de dados. Sua utilização está relacionada ao fato de possuírem latência inferior ao esquema de repetição automática. Para que seja possível realizar a detecção e correção de erros sem a necessidade de retransmissão, os códigos FEC adicionam dados redundantes à mensagem original, o que gera sobrecarga na transmissão.

O processo de detecção e correção de erros está diretamente relacionado aos *bits* de paridade inseridos na mensagem. A ocorrência de um único erro em uma sequência de *bits* pode ser verificada através da utilização de um único *bit* de paridade [**?**]. Entretanto,

para corrigir um erro é preciso saber qual *bit* está errado para, então, inverter o seu estado de 0 para 1 ou vice-versa. Para isso, é necessário adicionar um número maior de *bits*, cuja quantidade varia de acordo com o código de detecção utilizado e influencia na eficiência do processo.

Os códigos FEC podem ser classificados em dois grupos: os *códigos de bloco* e os *códigos convolucionais* [**?** **?**]. Os códigos de bloco são caracterizados pela divisão da mensagem em blocos, compostos pela mensagem original e alguns *bits* redundantes, chamados *palavra código*. Os códigos convolucionais aceitam uma sequência de *bits*, de comprimento limitado ao tamanho da memória do codificador. A sequência é codificada segundo uma convolução realizada entre os *bits* de entrada e a resposta ao impulso do codificador. Por isso, o codificador pode ser visto como uma "convolução discreta no tempo", que realiza a operação utilizando uma janela deslizante de duração proporcional ao tamanho da sua memória [**?** **?**].

3.1.1 Códigos de Bloco

Os códigos de bloco utilizam palavras de mesmo tamanho combinadas para formar a mensagem ou sequência de dados. Como pode ser visto na Figura 3.2, uma mensagem é dividida em blocos menores, formando palavras de tamanho k que são codificadas de acordo com um padrão predeterminado para gerar a paridade (m). O resultado é um bloco de tamanho n que será transmitido através do meio. O bloco resultante é denominado bloco (n, k), sendo $n > k$. Os $n - k$ *bits* restantes são *bits* de paridade, utilizados para a detecção e correção de erros.

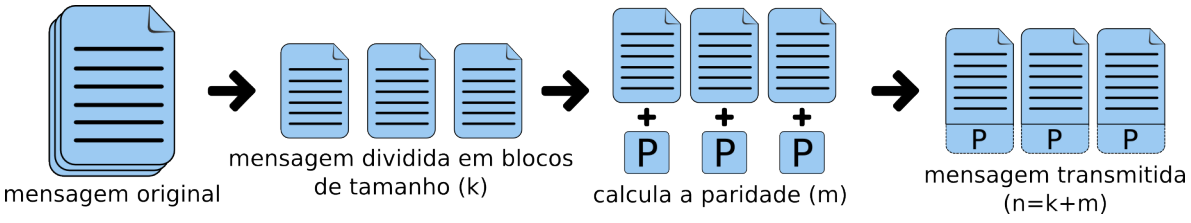


Figura 3.2: Criação dos códigos de bloco

Uma subclasse dos códigos de bloco são os códigos de bloco lineares. Estes utilizam aritmética de módulo 2 (OU exclusivo - XOR) para gerar os *bits* de paridade, calculada

a partir da soma de duas palavras código do próprio bloco, da seguinte forma: $A \oplus B = C|A, B \in k$ e $C \in n$, ou seja, a soma de duas palavras código (A e B) gera uma terceira palavra código (C).

No conjunto de códigos de bloco lineares, existem os códigos cíclicos [**?**]. Estes são caracterizados pela facilidade de codificação e estrutura matemática bem definida, o que os torna bastante eficientes para detecção de erros. Além de ser um código de bloco linear, nos códigos cíclicos qualquer deslocamento dentro de uma palavra código resulta em uma palavra código. Os códigos cíclicos utilizam um polinômio gerador, denominado $G(x)$, para a geração de palavras código.

Entre os mais poderosos e importantes códigos de bloco estão os pertencentes à subclasse denominada Código *Bose-Chaudhuri-Hocquenghem* (BCH). Os códigos BCH são cíclicos e capazes de corrigir até t erros aleatórios em cada palavra código. O número de erros que pode ser corrigido varia em função do polinômio gerador utilizado pelo código. Um importante código pertencente a essa subclasse é o Reed-Solomon [**?**], que é detalhado na Seção 3.2.

3.1.2 Códigos Convolucionais

Na década de 1950 um grande número de cientistas estudava a transmissão de dados em canais ruidosos e buscava soluções para corrigir erros em dados corrompidos. Os primeiros estudos introduziram os códigos de bloco e utilizaram a verificação de paridade para a correção de erros. Estudos desenvolvidos por Peter Elias [**?**] mostram que os códigos baseados em verificação de paridade são eficientes, porém não utilizam toda a capacidade do canal, gerando desperdício. A partir dessa conclusão, Elias propôs, em 1955, uma nova técnica de codificação, denominada codificação convolucional [**?**], que permite a utilização de toda a capacidade do canal.

A codificação convolucional, assim como a codificação de bloco, é uma técnica de correção direta de erros. Mas, ao invés de trabalhar com os blocos de símbolos de tamanho fixo, trata séries de dados [**?**], o que é especialmente conveniente para comunicações que transmitem dados de forma serial. A decodificação serial torna desnecessária a uti-

lização de memória para armazenar toda a mensagem no decodificador, sendo suficiente armazenar uma pequena quantidade de símbolos da mensagem [**?** **?** **?**].

O codificador convolucional, ilustrado pela Figura 3.3, pode ser visto como uma Máquina de Estados Finitos, em que a palavra código é gerada em função dos *bits* de entrada e do estado atual. Os *bits* da mensagem são inseridos no codificador em grupos de tamanho k e são deslocados k posições para a direita. Dentro do codificador existem somadores módulo 2 (portas OU exclusivo), onde os *bits* são combinados para formar a palavra código. A saída é resultado de operações lineares, executadas sobre o símbolo de entrada atual e o estado do registrador de deslocamento. Uma vez que para cada k *bits* de entrada são produzidos n *bits* de saída, então a taxa de codificação (r) é dada por $r = k/n$ [**?** **?**].

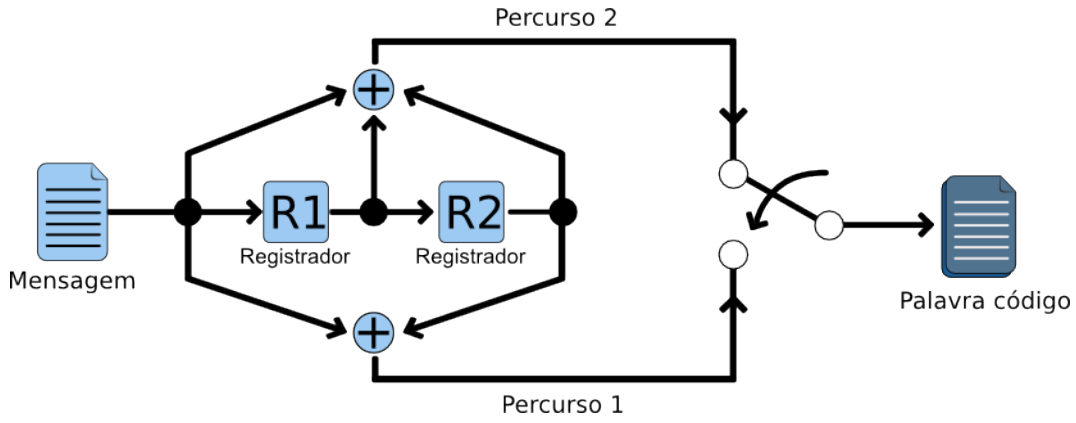


Figura 3.3: Codificador convolucional com n=2 e K=3

Os códigos convolucionais geralmente são descritos a partir da taxa de codificação e do comprimento de restrição. O comprimento de restrição, expresso em termos de *bit* de mensagem, representa o número de *bits* de entrada que influenciam para a formação de n *bits* de saída. Um codificador com eD estágios de deslocamento possui memória com tamanho equivalente ao número de estágios. Após a inserção de um *bit* no registrador, são necessários $eD + 1$ deslocamentos para que este saia do registrador. O comprimento de restrição é equivalente ao número de deslocamentos realizados pelo código [**?**].

O desempenho de um codificador convolucional está relacionado ao algoritmo decodificador e às propriedades de correção de erro do código. Essas propriedades estão associadas a parâmetros como o peso de Hamming, que indica o número de elementos não nulos no

vetor de um código, e com a distância Hamming, referente ao número de posições em que um par de vetores possui elementos, respectivamente, diferentes. A distância livre (d_l) é o elemento mais importante quando se trata de desempenho e, em códigos convolucionais, d_l é definida como a "distância mínima de Hamming" entre dois vetores quaisquer. Um código convolucional é capaz de corrigir até $d_l/2$ erros.

Uma importante evolução dos FECs ocorreu em 1993, com o desenvolvimento dos *códigos turbo* [**?**]. Eles utilizam codificadores convolucionais separados por intercalador e detecção interativa. Essa combinação torna-os poderosos, quando comparados a outros códigos [**?**].

3.2 Códigos Reed-Solomon

Os códigos Reed-Solomon foram desenvolvidos em 1960 por Irvin Reed e Gustave Solomon [**?**]. Entre as principais aplicações do RS está o aumento da confiabilidade de dispositivos de armazenamento digital de dados, como discos rígidos e discos compactos (CDs) [**?**]. Por ser um tipo de código de bloco, o RS é capaz de detectar e corrigir erros em dados transmitidos em rajadas ou armazenados em blocos. Em ambos os casos, o decodificador deve possuir memória suficiente para armazenar o bloco de *bits* durante o processo de decodificação, a fim de realizar a correção de possíveis erros.

Um código RS é um código cíclico, não binário, capaz de codificar uma sequência de k símbolos de dados em n símbolos codificados. Aos k símbolos originais, são adicionados $n - k$ símbolos de paridade, utilizados para a detecção e correção de erros. A paridade adicionada aos dados originais indica a capacidade de correção de erros. Um código $RS(n, k)$ é capaz de corrigir até t símbolos corrompidos de um bloco, em que $t = \frac{n-k}{2}$ [**?** **?**].

A construção de um código Reed-Solomon tem como base a aritmética de corpos finitos, também conhecidos como Corpos de Galois (GF, *Galois Fields*) [**?** **?**]. Conforme mostra a Figura 3.4, a mensagem é codificada através de um polinômio gerador ($G(x)$) que, aplicado à mensagem original, resulta em uma palavra código.

A decodificação é realizada em quatro etapas: (i) cálculo da síndrome, que indica se

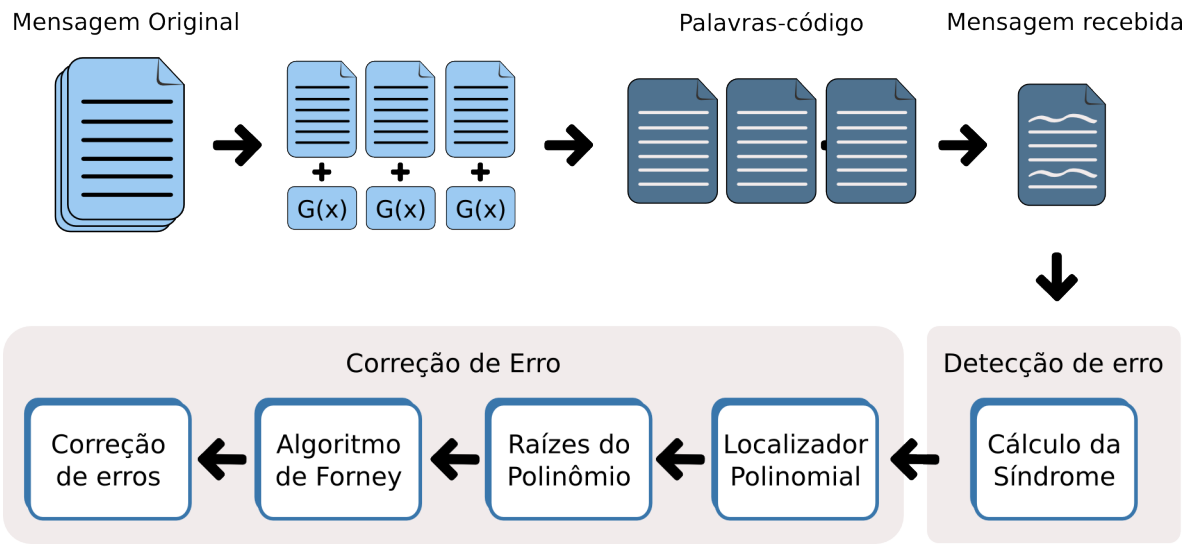


Figura 3.4: Estrutura de um bloco de dados codificado com Reed-Solomon

a palavra código recebida possui erros; (ii) obtenção do polinômio localizador através do Algoritmo Euclidiano Extendido (EEA), do Algoritmo Berlekamp-Massey (BMA) ou do algoritmo Peterson-Gorenstein-Zierler (PGZ) [**?** **?** **?**]; (iii) localização das raízes do polinômio, por meio do algoritmo de busca de Chien e (iv) determinação do padrão de erro inserido na palavra código, utilizando o algoritmo de Forney. Uma vez encontrado o padrão de erro, é possível corrigí-lo.

3.2.1 Processo de Codificação

A codificação de uma mensagem envolve a geração de símbolos de paridade, utilizados para detecção e correção de erros. A quantidade de símbolos gerada afeta diretamente a capacidade de correção, de forma que, para cada $2t$ símbolos de paridade, é possível corrigir até t símbolos errados. A paridade é gerada a partir de $G(x)$, que possui a forma genérica da Equação 3.1, em que X representa o símbolo de paridade [**?**].

$$G(x) = g_0 + g_1X + g_2X^2 + ... + g_{2t-1}X^{2t-1} \tag{3.1}$$

O polinômio gerador possui grau igual à quantidade de símbolos de paridade, isto é, existem $2t$ potências de α, que são raízes do polinômio. Apesar das raízes de $G(x)$ serem comumente designadas como $\alpha, \alpha^2, ..., \alpha^{2t}$, elas podem iniciar em qualquer outra potência

diferente de α. A Equação 3.2 ilustra o cálculo do polinômio gerador para um código RS(7,3) que possui 4 símbolos de paridade.

$$\begin{aligned} G(x) &= (X-\alpha)(X-\alpha^2)(X-\alpha^3)(X-\alpha^4) \\ &= (X^2-(\alpha+\alpha^2)X+\alpha^3)(X^2-(\alpha^3+\alpha^4)X+\alpha^7) \\ &= (X^2-\alpha^4X+\alpha^3)(X^2-\alpha^6X+\alpha^0) \\ &= X^4-(\alpha^4+\alpha^6)X^3+(\alpha^3+\alpha^{10}+\alpha^0)X^2-(\alpha^4+\alpha^9)X+\alpha^3 \\ &= X^4-\alpha^3X^3+\alpha^0X^2-\alpha^1X+\alpha^3 \end{aligned} \tag{3.2}$$

Ao considerar a equivalência entre operações de soma e subtração[2], pode-se substituir os sinais negativos por positivos. Portanto, o polinômio da Equação 3.3 é equivalente ao da Equação 3.2.

$$G(x) = X^4+\alpha^3X^3+\alpha^0X^2+\alpha^1X+\alpha^3 \tag{3.3}$$

Para exemplificar o processo de codificação é considerado envio de uma mensagem $m(x)$ representada por $\alpha^5\alpha^3\alpha^1$, que corresponde ao envio dos símbolos 111(α^5), 110 (α^3) e 010(α^1), cuja forma polinomial é a seguinte:

$$m(x) = \alpha^5X^2+\alpha^3X+\alpha^1 \tag{3.4}$$

Para obter os símbolos de paridade é necessário calcular o resto da divisão polinomial de $m(x)$ por $G(x)$, o que é feito multiplicando os valores de $m(x)$ por X^{2t} e deslocando 4 estágios para a direita ($2t = 4$). O resultado desta operação é o polinômio $\alpha^5X^6 + \alpha^3X^5 + \alpha^1X^4$. Em seguida, divide-se a mensagem deslocada pelo polinômio gerador da

[2]Válido somente para $GF(2^n)$

Equação 3.3, que resulta no seguinte polinômio de paridade $p(x)$:

$$p(x) = \alpha^6 X^3 + \alpha^4 X^2 + \alpha^2 X + \alpha^0 \tag{3.5}$$

A soma do polinômio de mensagem ($\alpha^5 X^6 + \alpha^3 X^5 + \alpha^1 X^4$) com o polinômio de paridade (Equação3.5) resulta na sequência de símbolos que serão enviados. A sequência é representada pela Equação 3.6.

$$E(x) = \alpha^5 X^6 + \alpha^3 X^5 + \alpha^1 X^4 + \alpha^6 X^3 + \alpha^4 X^2 + \alpha^2 X + \alpha^0 \tag{3.6}$$

3.2.2 Processo de Decodificação

Ao receber uma mensagem codificada com Reed-Solomon é preciso verificar se a sequência recebida, $R(x)$, contém uma palavra código válida $E(x)$. Caso não possua, é necessário localizar e corrigir os erros. Todo o processo é descrito detalhadamente em [**?**]. Por conveniência, este trabalho apresenta uma visão geral do processo.

Como mostra a Equação 3.7, $E(x)$ é o produto entre mensagem original ($m(x)$) e o polinômio gerador ($G(x)$). Uma vez que $E(x)$ e $G(x)$ são múltiplos, as raízes de $G(x)$ também são raízes de $E(x)$, o que permite a verificação da mensagem recebida a partir de $G(x)$.

$$E(x) = m(x).G(x) \tag{3.7}$$

A aferição da mensagem recebida envolve o cálculo da síndrome, que é o resultado do cômputo das raízes do polinômio gerador sobre a sequência de *bits* recebida. Sabendo que a mensagem transferida pode ter sido corrompida durante a transmissão, que $R(x)$ é o resultado da soma entre a mensagem transmitida e um padrão de erro ($e(x)$), ou seja, $R(x) = E(x) + e(x)$, e que $E(x)$ é múltiplo de $G(x)$, deduz-se que é possível constatar a ocorrência de erros aplicando as raízes de $G(x)$ sobre o polinômio $R(x)$. Como resultado, serão obtidos valores diferentes de zero nos casos em que a mensagem recebida contiver erros. Desse modo, pode-se representar o cálculo da síndrome pela Equação 3.8.

$$S_i = R(x)|_{X=\alpha^i} = R(\alpha^i) \qquad i = 1, ..., n-k \tag{3.8}$$

Para exemplificar, a Equação 3.9 representa a mensagem $E(x)$ transmitida e uma sequência de *bits* diferente recebida. Aplicando a Equação 3.8 sobre o polinômio $R(x)$ para calcular a síndrome, conclui-se que a mensagem recebida contém dois símbolos corrompidos, $\alpha^0 X^3$ e $\alpha^6 X^4$.

$$\begin{aligned}
E(x) &= \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3 + \alpha X^4 + \alpha^3 X^5 + \alpha^5 X^6 \\
&= (100) + (001)X + (011)X^2 + (101)X^3 + (010)X^4 + (110)X^5 + (111)X^6 \\
R(x) &= (100) + (001)X + (011)X^2 + (100)X^3 + (101)X^4 + (110)X^5 + (111)X^6 \\
&= \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^0 X^3 + \alpha^6 X^4 + \alpha^3 X^5 + \alpha^5 X^6
\end{aligned} \tag{3.9}$$

A síndrome é calculada utilizando a Equação 3.8 aplicada sobre o polinômio recebido, que no exemplo (Equação 3.9) possui dois símbolos corrompidos. O desenvolvimento do cálculo da síndrome pode ser observado na Equação 3.10, cujo cômputo é feito substituindo-se X pelo valor de α^i na Equação $R(x)$. Após a substituição, os valores devem ser reduzidos através da operação "mod 7" para obter os expoentes corretos de α. O resultado final é obtido aplicando-se a função XOR aos símbolos que compõem a soma. Para toda palavra código válida, o resultado de S_i deve ser igual a 0. No entanto, como resultado da operação obtém-se todos os valores diferentes de zero, o que indica que a palavra código recebida contém erros.

$$\begin{aligned}
S_1 = R(\alpha) &= \alpha^0 + \alpha^3 + \alpha^6 + \alpha^3 + \alpha^{10} + \alpha^8 + \alpha^{11} \\
&= \alpha^0 + \alpha^3 + \alpha^6 + \alpha^3 + \alpha^2 + \alpha^1 + \alpha^4 \\
&= \alpha^3
\end{aligned} \tag{3.10a}$$

$$
\begin{aligned}
S_2 = R(\alpha^2) &= \alpha^0 + \alpha^4 + \alpha^8 + \alpha^6 + \alpha^{14} + \alpha^{13} + \alpha^{17} \\
&= \alpha^0 + \alpha^4 + \alpha^1 + \alpha^6 + \alpha^0 + \alpha^6 + \alpha^3 \\
&= \alpha^5 \qquad (3.10b) \\
S_3 = R(\alpha^3) &= \alpha^0 + \alpha^5 + \alpha^{10} + \alpha^9 + \alpha^{18} + \alpha^{18} + \alpha^{23} \\
&= \alpha^0 + \alpha^5 + \alpha^3 + \alpha^2 + \alpha^4 + \alpha^4 + \alpha^2 \\
&= \alpha^6 \qquad (3.10c) \\
S_4 = R(\alpha^4) &= \alpha^0 + \alpha^6 + \alpha^{12} + \alpha^{12} + \alpha^{22} + \alpha^{23} + \alpha^{29} \\
&= \alpha^0 + \alpha^6 + \alpha^5 + \alpha^5 + \alpha^1 + \alpha^2 + \alpha^1 \\
&= 0 \qquad (3.10d)
\end{aligned}
$$

Uma vez detectada a corrupção da palavra código, é preciso identificar os valores errados e suas localizações. Esse processo utiliza um polinômio de erro ($e(X)$), cujo padrão pode ser representado pela Equação 3.11, em que e_n representa o erro de índice n e X^n sua respectiva localização.

$$
e(X) = \sum_{n=0}^{6} e_n X^n \qquad (3.11)
$$

Supondo que para a mensagem enviada o padrão de erros seja representado pela Equação 3.12, pode-se dizer que a paridade foi corrompida em 1 *bit* (α^2) e um símbolo da mensagem foi corrompido em 3 *bits* (α^5).

$$
\begin{aligned}
e(x) &= 0 + 0X + 0X^2 + \alpha^2 X^3 + \alpha^5 X^4 + 0X^5 + 0X^6 \\
&= (000) + (000)X + (000)X^2 + (001)X^3 + (111)X^4 + (000)X^5 + (000)X^6 \qquad (3.12)
\end{aligned}
$$

Sabendo que existem v erros na sequência, é indispensável que se encontrem os símbolos corrompidos e suas respectivas localizações. Para isso, define-se uma variável correspondente à localização do erro, por exemplo, $\beta_l = \alpha^{jl}$. Em seguida, obtêm-se os $2t$ símbolos da síndrome, através da substituição de α^i no polinômio de erro, conforme a

Equação 3.13. A solução é obtida por meio de um sistema de equações, conhecido como "algoritmo de decodificação Reed-Solomon", pois o sistema não é linear, o que impede a resolução através equações de sistemas lineares.

$$\begin{aligned} S_1 &= R(\alpha) = e_{j1}\beta_1 + e_{j2}\beta_2 + ... + e_{jv}\beta_v \\ S_2 &= R(\alpha^2) = e_{j1}\beta_1^2 + e_{j2}\beta_2^2 + ... + e_{jv}\beta_v^2 \\ &... \\ S_{2t} &= R(\alpha^{2t}) = e_{j1}\beta_1^{2t} + e_{j2}\beta_2^{2t} + ... + e_{jv}\beta_v^{2t} \end{aligned} \tag{3.13}$$

O cálculo inicia pela identificação das posições em que se encontram os símbolos corrompidos. Isso é feito utilizando-se o polinômio localizador, representado por $\sigma(x)$, definido na Equação 3.14. A localização do erro é dada pela inversa das raízes de $\sigma(x)$, que são $1/\beta_1$, $1/\beta_2$, ..., $1/\beta_v$

$$\begin{aligned} \sigma(X) &= (1 + \beta_1 X)(1 + \beta_2 X)...(1 + \beta_v X) \\ &= 1 + \sigma_1 X + \sigma_2 X^2 + ... + \sigma_v X^v \end{aligned} \tag{3.14}$$

Os coeficientes de σ são determinados pela modelagem auto-regressiva que utiliza uma matriz de síndromes. As t primeiras síndromes da matriz são utilizadas para prever as demais, como demonstra a Equação 3.15. Ao calcular os coeficientes de σ para o código RS(7,3), cuja capacidade de correção é de até 2 erros, obtém-se e Equação 3.16.

$$\begin{bmatrix} S_1 & S_2 & ... & S_t \\ S_2 & S_3 & ... & S_{t+1} \\ ... & ... & ... & ... \\ S_t & S_{t+1} & ... & S_{2t-1} \end{bmatrix} \begin{bmatrix} \sigma_t \\ \sigma_{t-1} \\ ... \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} -S_{t+1} \\ -S_{t+2} \\ ... \\ -S_{2t} \end{bmatrix} \tag{3.15}$$

$$\begin{bmatrix} S_1 & S_2 \\ S_2 & S_3 \end{bmatrix} \begin{bmatrix} \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} S_3 \\ S_4 \end{bmatrix}$$
$$\begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix} \begin{bmatrix} \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} \alpha^6 \\ 0 \end{bmatrix} \tag{3.16}$$

Os valores de σ_1 e σ_2 são obtidos pelo cálculo da inversa da matriz, conforme a Equação 3.17. Os detalhes para o cálculo dos coeficientes de σ_1 e σ_2 podem ser encontrados em [**?**]. A partir dos coeficientes de σ_1 e σ_2 e do polinômio de $\sigma(X)$, pode-se representar o polinômio localizador de erro pela Equação 3.18.

$$\begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix}^{-1} \begin{bmatrix} \alpha^6 \\ \alpha^0 \end{bmatrix} = \begin{bmatrix} \sigma_2 \\ \sigma_1 \end{bmatrix}$$
$$\begin{bmatrix} \alpha^1 & \alpha^0 \\ \alpha^0 & \alpha^5 \end{bmatrix} \begin{bmatrix} \alpha^6 \\ 0 \end{bmatrix} = \begin{bmatrix} \alpha^0 \\ \alpha^6 \end{bmatrix} \tag{3.17}$$

$$\sigma(X) = \alpha^0 + \alpha^6 X + \alpha^0 X^2 \tag{3.18}$$

As inversas das raízes de $\sigma(x)$ são obtidas aplicando o polinômio para todos os elementos do campo. Resultados iguais a zero indicam localização de erro. Como demonstra a Equação 3.19, os elementos α^3 e α^4 são raízes de $\sigma(X)$, que caracterizam erros nas localizações $\beta_1 = \alpha^3$ e $\beta_2 = \alpha^4$.

$$\sigma(\alpha^0) = \alpha^0 + \alpha^6 + \alpha^0 = \alpha^6 \neq 0$$
$$\sigma(\alpha^1) = \alpha^0 + \alpha^7 + \alpha^2 = \alpha^2 \neq 0$$
$$\sigma(\alpha^2) = \alpha^0 + \alpha^8 + \alpha^4 = \alpha^6 \neq 0$$
$$\sigma(\alpha^3) = \alpha^0 + \alpha^9 + \alpha^6 = 0 \Rightarrow ERRO$$

$$\sigma(\alpha^4) = \alpha^0 + \alpha^{10} + \alpha^8 = 0 \Rightarrow ERRO$$
$$\sigma(\alpha^5) = \alpha^0 + \alpha^{11} + \alpha^{10} = \alpha^2 \neq 0$$
$$\sigma(\alpha^6) = \alpha^0 + \alpha^{12} + \alpha^{12} = \alpha^0 \neq 0 \tag{3.19}$$

Uma vez localizadas as posições de erro, é necessário identificar seus respectivos valores. O cálculo utilizará a notação e_l, em que l representa a ordem do erro. Os valores de erros para as localizações encontradas na Equação 3.19 podem ser determinados através de uma das quatro síndromes calculadas na Equação 3.13. Utilizando as síndromes S_1 e S_2 e aplicando as incógnitas, obtém-se a Equação 3.20.

$$S_1 = e(\alpha) = e_1\beta_1 + e_2\beta_2$$
$$S_2 = e(\alpha^2) = e_1\beta_1^2 + e_2\beta_2^2 \tag{3.20}$$

O resultado é alcançado através da Equação 3.17, convertendo a Equação anterior para sua forma polinomial, conforme a Equação 3.16. Após as devidas conversões e substituições, obtém-se a Equação 3.21. Conhecendo os valores e erro, α^2 e α^5, pode-se estimar o polinômio de erro $e(x)$, como demonstra a Equação 3.22.

$$\begin{bmatrix} \alpha^3 & \alpha^4 \\ \alpha^6 & \alpha^1 \end{bmatrix}^{-1} \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$$
$$\begin{bmatrix} \alpha^2 & \alpha^5 \\ \alpha^0 & \alpha^4 \end{bmatrix} \begin{bmatrix} \alpha^3 \\ \alpha^5 \end{bmatrix} = \begin{bmatrix} \alpha^2 \\ \alpha^5 \end{bmatrix} \tag{3.21}$$

$$e(x) = e_1X^{j1} + e_2X^{j2}$$
$$e(x) = \alpha^2X^3 + \alpha^5X^4 \tag{3.22}$$

Tendo em vista a capacidade de correção de RS(7,3) e que a mensagem recebida sofreu corrupção de exatamente dois símbolos, pode-se afirmar que o algoritmo é capaz de decodificar corretamente a mensagem. A prova da decodificação correta pode ser obtida estimando a sequência transmitida, $\hat{E}(x)$, e verificando sua equivalência com a mensagem transmitida, $E(x)$. A estimativa é obtida pela soma de $R(x)$ e $e(x)$, como apresentado na Equação 3.23. Uma vez que $E(x)$ é igual a $\hat{E}(x)$, conclui-se que a mensagem foi decodificada corretamente.

$$\begin{aligned}
\hat{E}(x) &= R(x) + e(x) \\
\hat{E}(x) &= \alpha^0 + \alpha^2 X + \alpha^4 X^2 + (\alpha^0 + \alpha^2) X^3 + (\alpha^6 + \alpha^5) X^4 + \alpha^3 X^5 + \alpha^5 X^6 \\
\hat{E}(x) &= \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3 + \alpha^1 X^4 + \alpha^3 X^5 + \alpha^5 X^6
\end{aligned} \tag{3.23}$$

3.2.3 Complexidade Computacional

O processo de codificação e decodificação exige a execução de uma série de operações de adição e multiplicação, além de algumas inversões. Essas operações matemáticas possuem uma complexidade, que pode variar de acordo com a quantia de cálculos implementados pelo algoritmo. Visando desenvolver um codificador RS com o menor consumo de energia possível, Biard e Noguet [**?**] realizaram um estudo que apresenta a complexidade computacional e a energia consumida durante o processo de codificação e decodificação do Reed-Solomon.

A análise da complexidade parte do princípio de que as operações são realizadas seguindo as regras dos Corpos de Galois, cujos símbolos possuem valores limitados por 2^m. Os resultados obtidos apontam uma complexidade[3] de $O(t)$ para códigos $GF(2^8)$, com taxa de codificação de $r = 0.8$. Para chegar a esse resultado, foram analisados códigos com diferentes capacidades de correção.

Nesta análise, são consideradas as operações de adição (GFadd), multiplicação entre dois elementos quaisquer do corpo (GFmul), multiplicação entre dois elementos específicos

[3] "t" representa a capacidade de correção

(α^i) do corpo (GFmulαi), cálculo da inversa de um elemento (GFinv), armazenamento em registrador (GFreg) e armazenamento em memória (GFmem). Sendo assim, a complexidade computacional pode ser calculada a partir das informações da Tabela 3.2.1[**?**].

A partir dos dados contidos na tabela, pode-se obter a Complexidade Computacional por Bit de Informação (CCIB, *Computational Complexity per Information Bit*) e analisar o consumo de energia do codificador. O cálculo pode ser feito pela Equação 3.24, que considera a utilização do EEA como localizador polinomial, pois é o algoritmo mais utilizado na implementação do RS e requer menor quantidade de cálculos para a maioria dos casos [**?**].

$$\begin{aligned} CCIB = & \frac{1}{m}\left[\frac{1-r}{2r}(4t+1)GFmul + \frac{1}{r}(6t-1)GFmul\alpha i\right] + \\ & \frac{1}{m}\left[\left(\frac{1}{r}(6t-1) + \frac{1-r}{2r}(4t+1)\right)GFadd + \frac{1-r}{r}GFinv\right] + \\ & \frac{1}{m}\left[\left(\frac{1}{r}(6t-1) + \frac{1-r}{2r}(6t-1)\right)GFreg + 1GFmem\right] \end{aligned} \tag{3.24}$$

O valor de CCIB é calculado tomando como base o número de símbolos utilizados pelo código (m), sua taxa de codificação (r) e capacidade de correção de erros (t). A partir desses valores é estabelecida uma relação com o custo computacional das operações de adição, multiplicação e inversões realizadas pelo RS, apresentados na Tabela 3.2.2[**?**]. A Tabela apresenta a estimativa do consumo de energia desses operadores, obtida através de uma ferramenta de análise de consumo de energia e utilizando um Dispositivo Lógico Programável (FPGA, *Field-Programmable Gate Array*) otimizado. Utilizando os valores dados pelo código e calculando seu custo computacional é possível obter o consumo de energia e a complexidade computacional de um código específico.

Para exemplificar, são analisados os códigos $RS(255, 223)$, $RS(511, 479)$, $RS(127, 117)$, e $RS(512, 496)$, que serão utilizados posteriormente para análise de desempenho. Uma vez que neste trabalho os códigos são empregados na verificação e correção de erros em *bytes*, então todos utilizam conjuntos compostos por 8 símbolos ($m = 8$). O dois primeiros,

Tabela 3.2.1: Complexidade Computacional para construção de blocos

		Inversão	Muliplicação	Mult.(αi)	Adição	Arm. Registrador	Mem.
Codificador				$n(2t)$	$n(2t)$	$n(2t)$	
Síndrome				$n(2t)$	$n(2t)$	$n(2t)$	
BMA		$2t-1$	$(2t-1)(2t+1)+t^2$		$(2t-1)+(2t)+t^2$	$(2t-1)(5t-1)+t$	
EEA		t	$t(4t)$		$t(4t)$	$t(6t+1)$	
PGZ	$t=1$	1	1		2		
	$t=2$	1	9		4	4	
	$t=3$	1	27		6	15	
Busca Chien				$n(2t-1)$	$n(2t-1)$	$n(2t-1)$	
Alg. Forney		t	t				
Corr. Erro					t		
Atraso							k
Total (EEA)		$2t$	$t(4t+1)$	$n(6t-1)$	$n(6t-1)+t(4t+1)$	$n(6t-1)+(t(6t+1)$	k

Tabela 3.2.2: Custo computacional dos operadores GF

Operador GF	Complexidade Computacional	Consumo de Energia (pJ)
1GFadd	m XOR	$0.4m$
1 GFmulαi	$m(m-2)/2$ XOR	$0.4m(m-2)/2$
1 GFmul	m^2 AND $3(m-1)^2/2$ XOR	$0.4(m^2+3(m-1)^2/2)$
1 GFinv	m ROM read	$8m$
1 GFreg	m REG write	$2m$
1 GFmem	m RAM read e write	$10m$

$RS(255, 223)$ e $RS(511, 479)$, são capazes de corrigir 16 *bytes* corrompidos ($t = 16$), sendo que a taxa de codificação do primeiro é dado pela Equação 3.25 ($r = 0,87$) e o segundo possui valor de $r = 0,93$. O terceiro código, corrige até 5 *bytes* corrompidos e possui taxa de codificação igual a 0,92. O $RS(512, 496)$, cuja capacidade de correção é de 8 *bytes*, possui a maior taxa de codificação, sendo $r = 0,96$.

$$r = \frac{223}{255}$$
$$r = 0,87 \tag{3.25}$$

Dados os valores, pode-se calcular o custo computacional de cada operação realizada pelo código, que pode ser observado na Tabela 3.2.3. Uma vez que os códigos do exemplo possuem o mesmo valor de m, então o custo computacional para realizar uma única operação de adição, multiplicação, inversão e armazenamento, permanece o mesmo para ambos.

Tabela 3.2.3: Custo computacional dos operadores dos códigos RS para m=8

Operador GF	Complexidade Computacional	Consumo de Energia (pJ)
1GFadd	8 XOR	3.2
1 GFmulαi	24 XOR	9.6
1 GFmul	64 AND 74 XOR	55
1 GFinv	8 ROM read	64
1 GFreg	8 REG write	16
1 GFmem	8 RAM read e write	80

A partir dos valores apresentados na Tabela 3.2.3, pode-se calcular o CCIB. A energia consumida pelos códigos analisados varia devido às diferentes quantidades de operações

executadas por bloco. Sendo assim, apesar do custo para realizar uma única operação ser o mesmo, a energia consumida é diferente pois cada código executa uma determinada quantidade operações. A variação da quantidade de operações realizadas afeta a complexidade computacional, pois quanto mais operações forem necessárias para codificar um bloco, maior é a complexidade do código.

A Equação 3.26 apresenta o cálculo detalhado para obtenção do consumo de energia do $RS(255, 223)$. Este é o código que apresenta o maior consumo de energia entre os analisados, sendo seguido pelo $RS(511, 479)$, que consome aproximadamente 397,66pJ por *bit* de informação. Já o código $RS(127, 117)$ é o que menos consome energia, o qual gira em torno de 133,16pJ, sendo que o $RS(512, 496)$ gasta cerca de 190,37pJ. A Tabela 3.2.4 apresenta um resumo do consumo de energia desses códigos.

$$
\begin{aligned}
CCIB =& \frac{1}{8}\left[\frac{1-0.87}{2*0.87}*((4*16)+1)*55+\frac{1}{0.87}*((6*16)-1)*9.6\right]+ \\
& \frac{1}{8}\left[\left(\frac{1}{0.87}*((6*16)-1)+\frac{1-0.87}{2*0.87}*((4*16)+1)\right)*3.2+\frac{1-0.87}{0.87}*64\right]+ \\
& \frac{1}{8}\left[\left(\frac{1}{0.87}*((6*16)-1)+\frac{1-0.87}{2*0.87}*((6*16)-1)\right)*(16+1)*80\right] \\
CCIB =& 162.42+46.46+241.18 \\
CCIB =& 450.07pJ
\end{aligned}
\tag{3.26}
$$

Tabela 3.2.4: Energia consumida pelos códigos analisados

Código	Consumo de Energia (pJ)
RS(255,223)	450,07
RS(511,479)	397,66
RS(127,177)	133,16
RS(512,496)	190,37

O custo computacional dos códigos é compensado pela capacidade de recuperação da mensagem original, que pode ser crítica em muitos casos. Ao escolher o código a ser utilizado, é importante considerar sua taxa de codificação, pois ela impacta fortemente na complexidade.

CAPÍTULO 4

UM ESQUEMA PARA ENTREGA DE MENSAGENS CODIFICADAS EM REDES DTNs

Este capítulo apresenta um esquema, denominado Entrega de Mensagens CODificadas (EMCOD), para entrega de mensagens em redes DTNs. O EMCOD, visa reduzir o tempo e a sobrecarga para entrega de mensagens íntegras ao destinatário em redes que sofrem longos atrasos quando ocorre perda de mensagens. Funciona entre as camadas de Aplicação e Agregação e é implementado dentro de uma subcamada denominada Controle de Codificação de Dados (CCD), como mostra a Figura 4.1. A subcamada Encaminhamento e Custódia reúne todas as funcionalidades já implementadas na camada de Agregação.

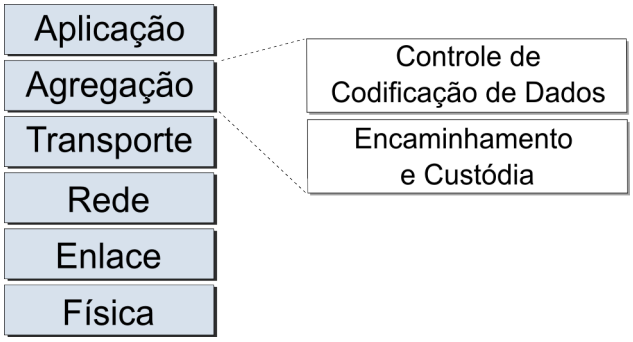


Figura 4.1: Divisão da Camada de Agregação

Este capítulo está organizado da seguinte forma: a Seção 4.1 apresenta uma visão geral da subcamada CCD. A Seção 4.2 explica como é realizada a construção das mensagens que serão enviadas para o destinatário. Na Seção 4.3 é descrito o o processo de codificação e decodificação. A Seção 4.4 descreve o funcionamento do Intercalador e sua importância para a subcamada CCD. Na Seção 4.5 é apresentado um exemplo da integração da subcamada CCD com o FEC Reed-Solomon. A Seção 4.6 discorre o processo recepção das mensagens, detalhando como é feita a reconstrução dos dados originais.

4.1 Visão Geral da Subcamada CCD

A subcamada de Controle de Codificação de Dados forma a parte superior da camada de Agregação. No nó emissor, a CCD recebe os *dados originais* da camada de Aplicação e os codifica gerando *mensagens*, que são enviadas para a subcamada inferior. No receptor, as mensagens são recebidas, decodificadas e a sequência de dados é reconstruída antes de ser enviada para a camada de Aplicação. A CCD é composta por dois módulos: o módulo codificação e o módulo interface, que podem ser vistos na Figura 4.2.

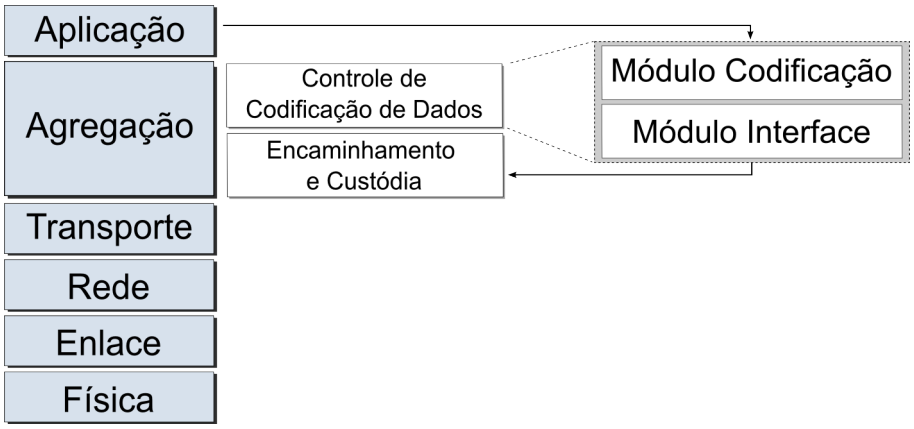


Figura 4.2: Divisão da subcamada CCD

Na origem, o *módulo codificação* recebe os dados originais da camada de Aplicação e os codifica utilizando um FEC[1]. Após a codificação, os dados são enviados para o *módulo interface* que monta as mensagens e as envia para a subcamada de Encaminhamento e Custódia para que sejam encaminhadas ao destinatário. Os processos de armazenamento, custódia e encaminhamento não sofrem alteração pelo EMCOD e não é adicionado nenhum processamento aos nós intermediários, a não ser aquele necessário para entregar as mensagens ao destinatário.

Como mostra a Figura 4.3, algumas mensagens podem ser perdidas durante a transmissão. A perda de mensagens pode ocorrer porque um nó saiu da rede antes de repassar a mensagem para o próximo salto. As mensagens retidas por algum nó da rede, devido à sua mobilidade, também podem ser tratadas como mensagens perdidas.

No destino, o módulo interface recebe as mensagens da subcamada Encaminhamento

[1]É possível utilizar qualquer FEC neste módulo, porém este trabalho utilizou somente o Reed-Solomon.

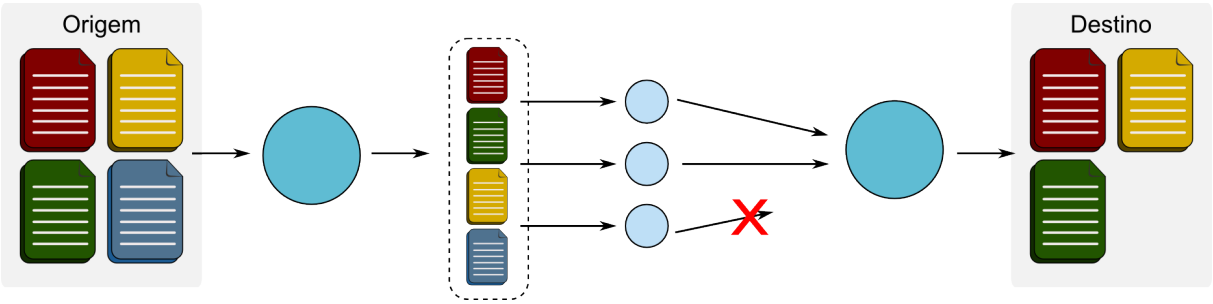


Figura 4.3: Transmissão das mensagens até o destinatário

e Custódia. Essas mensagens são processadas antes de serem decodificadas pelo módulo codificação. Caso as mensagens possam ser decodificadas com sucesso, os dados originais são reconstruídos e enviados para a camada de Aplicação. Caso não seja possível reconstruir os dados originais, é necessário solicitar o reenvio de algumas mensagens .

4.2 Construção das Mensagens

O módulo codificação recebe os dados da camada de Aplicação e os divide em uma quantidade pré-determinada de partes menores, chamadas *fragmentos*[2]. Os fragmentos são arranjados sequencialmente, formando uma matriz em que cada fragmento forma uma linha. Caso o último fragmento possua quantidade de *bytes* inferior ao das linhas anteriores, ele é completado com zeros. A matriz formada pelos fragmentos é dividida em colunas de forma que cada *byte* de um fragmento compõe uma coluna. Todos os *bytes* de uma mesma coluna formam um bloco.

Um *bloco* é formado pelos n-ésimos *bytes* de cada fragmento. A união de fragmentos e blocos forma uma matriz fxb. Esse processo é ilustrado pela Figura 4.4. Os blocos obtidos são codificados por um FEC e geram um *bloco codificado*, que é composto pelos dados do bloco e pela paridade, adicionada pelo FEC. Os blocos codificados são enviados para o módulo interface para construção da mensagem.

A simples codificação dos dados permite a detecção e recuperação de alguns *bytes* corrompidos, geralmente por erros ocorridos durante o processo de transmissão ou armazenamento, mas não permite que uma mensagem completa seja perdida. Em redes DTNs

[2]A quantidade de fragmentos varia de acordo com o FEC utilizado.

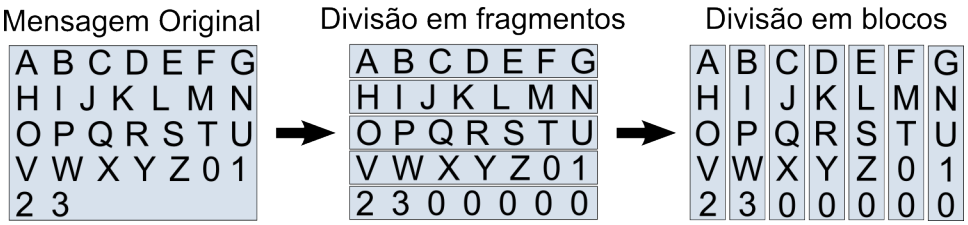


Figura 4.4: Composição da Matriz fxb

é comum ocorrer a perda de uma mensagem completa, seja porque o nó que possuía sua custódia saiu da rede, seja por falta de memória para armazenamento. Para minimizar os efeitos causados por esse problema, a CCD possui um Intercalador, que mistura os dados dos blocos codificados e os distribui em diferentes unidades. As unidades são agrupadas em mensagens que serão enviadas para o destinatário. A Figura 4.5 mostra os passos para a criação das mensagens.

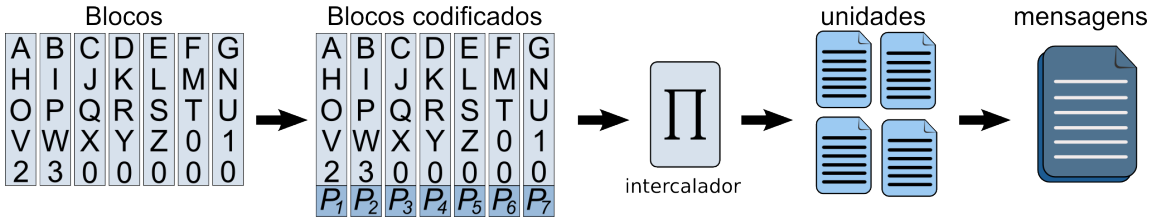


Figura 4.5: Processo de construção da mensagem

A construção de mensagens busca não apenas agrupar unidades para envio, mas também organizá-las de forma a maximizar a capacidade de recuperação dos dados. Para atingir este objetivo é preciso considerar que uma determinada sequência de unidades possui dados de um mesmo bloco codificado. Por exemplo, se o bloco codificado possuir tamanho 100 *bytes*, então em uma sequência de 100 unidades, cada uma irá conter um *byte* de um determinado bloco codificado. Agrupar essas unidades em uma mesma mensagem reduz a capacidade de recuperação dos dados, em caso de perda ou corrupção. Por esse motivo, as mensagens agrupam unidades sequencialmente distantes. O agrupamento de unidades pode ser observado na Figura 4.6.

Para realizar o agrupamento, as unidades recebem um identificador de sequência[3](id). Cada mensagem (M) contém unidades com os identificadores mais distantes possíveis. Para calcular quais unidades formam uma mensagem deve-se utilizar a Equação 4.1, que

[3]O identificador não é adicionado como campo da mensagem.

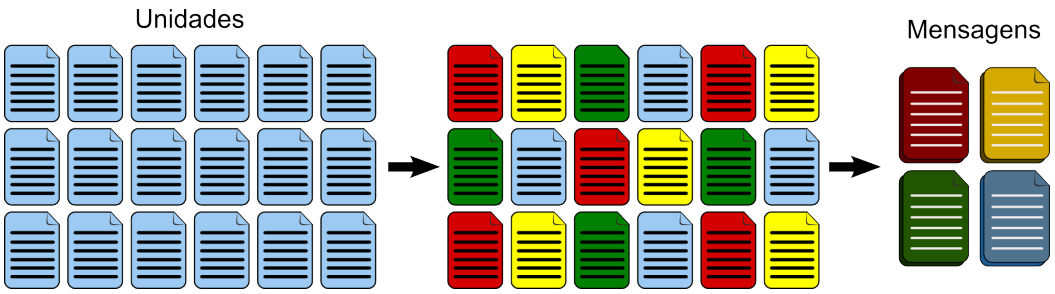


Figura 4.6: Agrupamento de unidades

recebe a quantidade de dados originais k e retorna o identificador (id) da mensagem que irá recebê-la.

$$id = k \mod \mid M \mid \tag{4.1}$$

Na Equação 4.1, o valor $\mid M \mid$ refere-se ao número total de mensagens criadas, cujo valor deve ser calculado considerando a quantidade de dados, o tamanho dos blocos e o tamanho das mensagens. A distribuição alcança maior eficiência quanto maior for a quantidade de unidades geradas. Portanto, para que a distribuição seja eficiente é preciso que a quantidade mínima de mensagens seja superior à capacidade de correção do FEC. Por exemplo, se o FEC for capaz de corrigir até 10 *bytes* corrompidos, então é necessário que sejam criadas mais que 10 mensagens. De qualquer forma, conforme demonstrado no Capítulo 5, se as unidades não fossem distribuídas em diferentes mensagens, a eficiência alcançada seria similar àquela obtida quando são geradas pequenas quantidades de mensagens.

Para que a reconstrução da sequência de dados seja possível, é preciso enviar uma informação sobre o identificador da mensagem e total de mensagens criadas. Para isso, pode-se criar um bloco de extensão[4] para o agregado, composto por dois campos: um deles possui o número total de mensagens e o outro o identificador da mensagem que está sendo encaminhada. A partir desta informação, o destinatário é capaz de verificar quais mensagens foram perdidas ou corrompidas e reorganizar as unidades na sequência correta.

[4]Compõe o "cabeçalho"do agregado. Um agregado é uma unidade de dados da camada de Agregação.

4.3 Codificação de Dados

A subcamada CCD codifica blocos de dados através de um FEC para permitir a sua recuperação em caso de corrupção de dados. A codificação, realizada no módulo codificador, é aplicada a cada *bloco* individualmente. A subcamada CCD permite a utilização de qualquer código corretor de erros, desde que o mesmo permita a codificação de blocos de dados.

Na origem, cada bloco é processado pelo FEC e resulta em um *bloco codificado*. Por exemplo, um bloco composto por 3 *bytes* que é processado pelo $RS(7, 3)$, tem 4 *bytes* de paridade adicionados ao bloco. Assim, o bloco codificado é formado pelos 3 *bytes* de dados e pelos 4 *bytes* de paridade.

No destino, o módulo codificador recebe os blocos codificados do Intercalador e verifica a sua integridade. Caso o bloco codificado não esteja íntegro, o FEC tenta corrigí-lo. No exemplo, se forem identificados até dois *bytes* corrompidos, o $RS(7, 3)$ é capaz de corrigir. Caso sejam identificados três *bytes* corrompidos é preciso solicitar o reenvio de pelo menos uma mensagem que contenha um dos *bytes* corrompidos. É possível saber qual mensagem deve ser enviada a partir dos identificadores adicionados ao cabeçalho do agregado.

Como a subcamada CCD distribui os dados de um bloco codificado em diferentes mensagens, é preciso verificar se todas foram recebidas. Caso positivo, é preciso enviar um NACK para que a mensagem contendo o *byte* corrompido seja reenviada. Se nem todas as mensagens foram recebidas e o dado corrompido é um espaço preenchido com zero, é possível aguardar o recebimento de novas mensagens por um tempo determinado. Após o esgotamento deste tempo é preciso solicitar o reenvio da mensagem.

4.4 Intercalador

A criação de unidades com dados intercalados permite a perda mensagens[5] durante o processo de transmissão. Como a mensagem perdida possui apenas alguns *bytes* de um mesmo bloco codificado, então o processo de decodificação recupera os que foram perdidos.

[5]O número de mensagens que podem ser perdidas depende do tamanho da mensagem original.

Desta forma, a capacidade de recuperação é maximizada e a sobrecarga da rede é reduzida.

O Intercalador opera sobre os blocos codificados, deslocando seus *bytes* em um determinado número de posições. Para realizar essa tarefa, são recebidos todos os blocos codificados para que seja possível montar a matriz fxb. Uma vez montada a matriz, inicia-se o processo de deslocamento dos dados, que é feito em função da sua linha. Na primeira, nenhum deslocamento é realizado. Na segunda, cada *byte* é deslocado uma posição à direita. De forma similar, os *bytes* da terceira e quarta linhas são deslocados duas e três posições, respectivamente, como mostra a Figura 4.7. Após o deslocamento, as colunas são novamente separadas e são criadas as unidades, a partir de cada coluna da matriz.

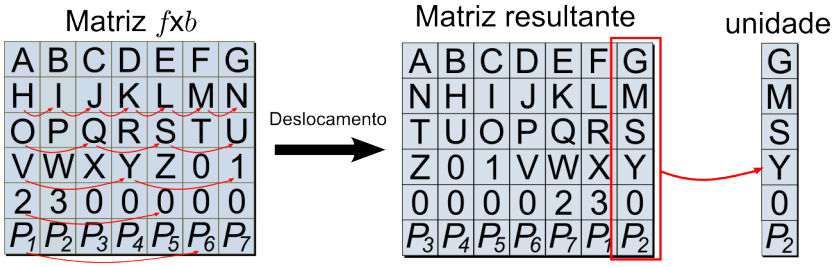


Figura 4.7: Processo de Intercalação e criação da unidadeCCD

No receptor o Intercalador realiza o processo inverso. As unidades são reagrupadas sequencialmente, formando uma matriz. Para reorganizá-las na ordem correta é preciso considerar o identificador da mensagem e o total de mensagens criadas. A partir dessas informações é possível saber quais unidades foram recebidas. Por exemplo, sabendo que existem 100 mensagens que agrupam 8 unidades cada, é possível aplicar a Equação 4.1 para saber quais unidades estão agrupadas em cada mensagem. Desta forma, sabe-se que a primeira mensagem agrupa as unidades cujos índices são: 1, 101, 201, 301, 401, 501, 601 e 701. Após a ordenação das unidades, os *bytes* são deslocados para a esquerda, em função da linha em que se encontram. Após esse processo é obtida a matriz fxb.

4.5 Funcionamento da Subcamada CCD com Reed-Solomon

Para que seja possível recuperar mensagens corrompidas é preciso utilizar um Código de Correção Direta de Erros. Com o intuito de analisar o funcionamento da subcamada CCD,

foi selecionado o FEC Reed-Solomon, embora qualquer outro código possa ser utilizado. Nesta subcamada, o RS é utilizado para codificação de blocos e recuperação de dados corrompidos.

Os dados recebidos pelo codificador são divididos em uma quantidade pré-definida de fragmentos ($\mid F \mid$), de tamanho ΔF, conforme a Equação 4.2. Um fragmento F_i é formado por $F_i = \{\omega_j, \omega_{j+1}, \omega_{j+2}, \ldots, \omega_{j+\Delta F} \mid j = i * \Delta F\}$, onde i é o índice do fragmento e ω representa o *byte* na mensagem original. Os n-ésimos *bytes* de cada fragmento formam um bloco, ou seja, um bloco B_i é formado por $B_i = \{F_{0,j}, F_{1,j}, \ldots, F_{|F|,j} \mid j = i\}$, onde i é o índice do bloco e j o índice do *byte* em cada fragmento. Sendo assim, o bloco B_1 é formado pelos *bytes* de índice 1 ($j = 1$) de cada fragmento.

$$\Delta F = \left\lceil \frac{k}{\mid F \mid} \right\rceil \tag{4.2}$$

4.6 Recepção das Mensagens e Reconstrução dos Dados Originais

No destinatário, as mensagens são recebidas pelo módulo interface para reconstrução dos blocos codificados. As mensagens recebidas são desagrupadas obtendo-se, assim, as unidades. Como as mensagens recebidas possuem um identificador de sequência e a quantidade total de mensagens criadas é conhecida, é possível ordenar as unidades na sequência correta. As unidades que não foram recebidas têm seus espaços preenchidos por zeros. As unidades são enviadas para o Intercalador, que realiza o processo inverso à intercalação. O resultado desse processo é uma matriz que contém os dados dos blocos codificados. A Figura 4.8 ilustra o processo de reconstrução dos blocos.

O módulo codificador recebe os blocos codificados, verifica a existência de erros e corrige os blocos, caso seja necessário. Os blocos que não podem ser corrigidos são solicitados novamente, através do envio de um NACK. O resultado do processo de codificação são os blocos, que são agrupados para formar os fragmentos. Caso o último fragmento possua zeros adicionais, estes são retirados e os dados originais são reconstruídos.

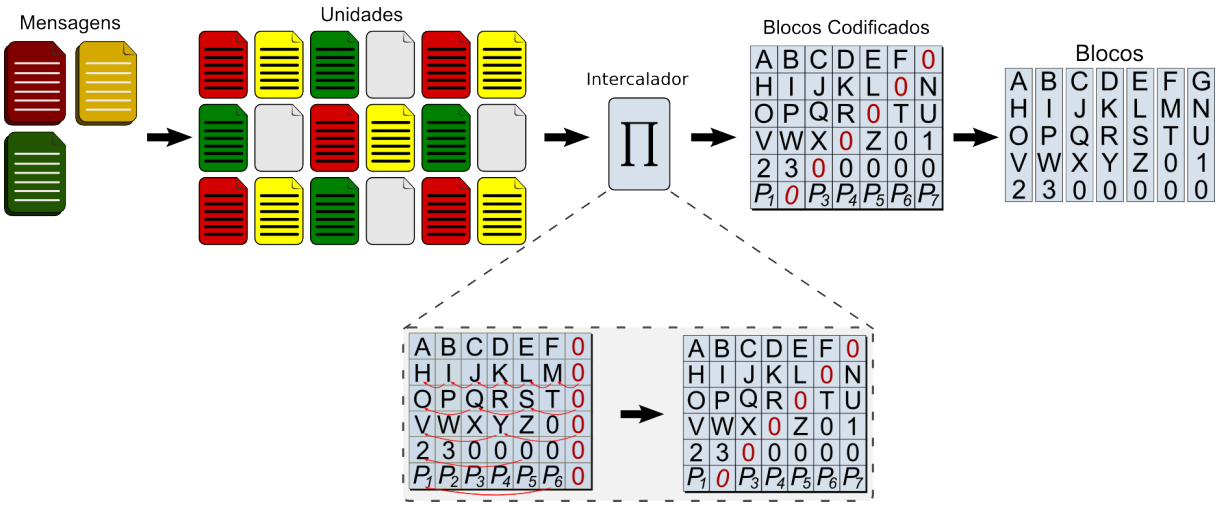


Figura 4.8: Reconstrução dos blocos

Para exemplificar, considere a utilização do código RS(255,223) e 1MB de dados recebidos da camada de Aplicação. Os dados são divididos em 223 fragmentos, conforme a Equação 4.3, e são obtidos fragmentos compostos por 4703 *bytes*. Os primeiros 4702 fragmentos possuem, juntos, 1.048.546 *bytes*[6] , do total de 1.048.576 *bytes*[7] que compõem os dados recebidos da Aplicação. Como sobram apenas 30 *bytes* para compor o último fragmento, os 193 *bytes* faltantes são completados com zeros.

$$\Delta F = \left\lceil \frac{1024 * 1024}{223} \right\rceil$$
$$= 4703 \tag{4.3}$$

Uma vez realizada a fragmentação, são criados 4703 blocos de tamanho 223. Após o processo de codificação, cada bloco possui 255 *bytes*, o que resulta em uma sobrecarga total de 150.689 *bytes*[8], ou 14,37%. Dados os tamanhos das unidades (255 *bytes*) e considerando um tamanho máximo para a mensagem de 2048 *bytes*, é necessário agrupar 8 unidades para formar uma mensagem. Sendo assim, serão formadas 588 mensagens. Aplicando a equação de distribuição (4.4), percebe-se que a primeira mensagem (M_1) é formada pelas

[6] 4702*223=1048546
[7] 1024*1024=1048576
[8] Resultado do processo de codificação

unidades com índice $M_1 = \{1, 589, 1177, 1765, 2353, 2941, 3529, 4117\}$. Por fim adicionam-se as informações sobre o índice da mensagem (1) e o número total de mensagens criadas (588) ao cabeçalho do agregado e encaminham-se as mensagens para a camada inferior.

$$\begin{aligned} id = k \mod & \mid M \mid \\ = k \mod & \ 588 \end{aligned} \tag{4.4}$$

Ao receber as mensagens, a subcamada CCD do destinatário irá desagrupá-las para obter as unidades. Uma vez que o identificador de cada mensagem é conhecido, pode-se aplicar a Equação 4.4 para reconstruir a sequência das unidades. A partir da recepção de um determinado número de mensagens já é possível dar continuidade ao processo de decodificação, não sendo necessário aguardar a recepção de todas. As unidades não recebidas tem seu espaço preenchido com zeros.

Uma vez conhecidos os valores correspondentes a cada unidade, é realizado o processo inverso à intercalação. Então, as unidades são agrupadas em uma matriz, em que cada unidade forma uma coluna. Os dados da coluna são deslocados à esquerda, em função da linha, a fim de obter os blocos codificados. Estes são decodificados e os dados originais são reconstruídos. Caso não seja possível decodificar algum bloco, a subcamada CCD aguarda a recepção de novas mensagens para realizar o processo novamente. O ideal é que a subcamada CCD saiba qual é o número mínimo de mensagens a serem recebidas para realizar todo o processo uma única vez.

CAPÍTULO 5

LIMITES ANALÍTICOS

Este capítulo apresenta os limites analíticos do EMCOD. Esta avaliação ignora problemas ou erros ocorridos durante o processo de roteamento das mensagens, custódia dos agregados e transmissão de dados, por considerar que tais erros são tratados pelos protocolos específicos de cada camada e são independentes do esquema proposto. São considerados problemas o recebimento de mensagens corrompidas e o não recebimento de algumas mensagens. Os limites analíticos consideram a utilização de um FEC qualquer, permitindo a utilização do código considerado mais adequado para o cenário no qual será utilizado.

Dada uma sequência de *Dados* originais D definida como um conjunto finito de dados, tal que $D = \{d_1, d_2, ..., d_n | d \in Dados\}$, a qual é recebida da camada de Aplicação do nó origem, D é processada pela subcamada CCD. O processamento dos dados envolve a sua divisão em fragmentos e blocos que são codificados por um FEC para gerar um bloco codificado (C)[1]. Os *bytes* dos blocos codificados são intercalados para gerar unidades (U), que são agrupadas em mensagens (M). As mensagens são enviadas para a subcamada de Encaminhamento e Custódia para serem encaminhadas pela rede.

A sequência de dados D de tamanho ΔD recebida da camada de Aplicação é dividida em uma quantidade determinada de fragmentos $(\mid F \mid)$, cujo tamanho (ΔF) é dado por:

$$\Delta F = \left\lceil \frac{\Delta D}{\mid F \mid} \right\rceil \tag{5.1}$$

Caso o último fragmento possua tamanho menor que os demais, então é preenchido com zeros, o que gera uma sobrecarga inicial de:

$$\lambda F = (\mid F \mid * \Delta F) - \Delta D \tag{5.2}$$

A criação de fragmentos gera uma sobrecarga diretamente proporcional ao tamanho

[1]Lembrando que um bloco codificado (C) é formado pelo bloco (B) e sua paridade

da mensagem original e à quantidade de fragmentos em que ela é dividida, sendo que a sobrecarga máxima de um fragmento é dada por $\Delta F - 1$.

Os blocos são formados pelos n-ésimos *bytes* de cada fragmento, portanto há $\mid B \mid$ blocos, cada um com tamanho:

$$\Delta B = \mid F \mid \tag{5.3}$$

Cada bloco é codificado por um FEC, que adiciona ρ *bytes* de paridade, o que resulta em blocos codificados de tamanho:

$$\Delta C = \Delta B + \rho \tag{5.4}$$

A sobrecarga gerada pela criação de um bloco codificado é diretamente proporcional à paridade utilizada pelo FEC[2]. A paridade permite que o EMCOD recupere a sequência original de dados, mesmo que nem todas as mensagens tenham sido recebidas. O processo de criação das unidades e mensagens fornece ao EMCOD uma capacidade de recuperação superior à capacidade de correção do FEC, visto que os *bytes* de um mesmo bloco são distribuídos por diferentes mensagens.

A quantidade de blocos codificados ($\mid C \mid$) criada é proporcional ao tamanho da sequência original de dados. O FEC utilizado para codificação possui uma capacidade de correção δ que permite a recuperação dos blocos, mesmo que alguns *bytes* tenham sido corrompidos[3]. Os blocos codificados são intercalados e geram unidades, cujos tamanhos e quantidades são diretamente proporcionais aos tamanhos e quantidades dos blocos codificados. A intercalação distribui os *bytes* de um bloco codificado entre as unidades.

As unidades são distribuídas entre as mensagens criadas, cuja quantidade ($\mid M \mid$) é proporcional ao tamanho dos dados originais. Como diferentes mensagens possuem *bytes* de diferentes blocos codificados, a taxa de recuperação (λR) é proporcional à quantidade de mensagens criadas e com a sequência de mensagens recebidas. Quando o tamanho

[2]Por exemplo, o RS(255,223) apresenta sobrecarga de 32 *bytes*. O Capítulo 6 apresenta informações sobre a sobrecarga de codificação gerada pela utilização do RS.

[3]Os *bytes* que ainda não foram entregues ao destinatário são considerados corrompidos pela subcamada CCD

dos dados originais é muito pequeno, a taxa de recuperação pode ser inferior à taxa de correção do FEC, pois cada mensagem possui mais que um *byte* de um mesmo bloco codificado.

Quando $\mid M \mid \leq \mid C \mid$, a taxa de recuperação é proporcional à taxa de correção do FEC e à quantidade de blocos e de mensagens, sendo equivalente a:

$$\lambda R \propto \Delta M - \left(\frac{\delta}{\mid C \mid / \mid M \mid} \right) \tag{5.5}$$

Por exemplo, em uma mensagem composta por 200 *bytes* e codificada pelo RS(255,223), que é capaz de corrigir até 16 *bytes* corrompidos por bloco e cujo bloco codificado é composto por 255 *bytes*, a subcamada CCD é capaz de recuperar a mensagem original a partir da recepção de 187 *bytes* íntegros, conforme demonstra a Equação 5.6.

$$\begin{aligned} \lambda R &\propto 200 - \left(\frac{16}{255/200} \right) \\ \lambda R &\propto 186,45 \end{aligned} \tag{5.6}$$

Por outro lado, quando $\mid M \mid > \mid C \mid$, a taxa de recuperação varia proporcionalmente ao tamanho da mensagem, podendo ser calculada através da Equação 5.7, em que ΔM representa o tamanho das mensagens criadas.

$$\lambda R \propto \left(\mid M \mid - \left(\frac{\mid M \mid}{\Delta C} * \delta \right) \right) * \Delta M \tag{5.7}$$

Por exemplo, uma mensagem composta por 50MB de dados, codificada pelo RS(255,223) cria 29 mensagens de tamanho 2040 e precisa receber 55.448 *bytes* íntegros, ou seja, 27 mensagens íntegras, como mostra a Equação 5.8.

$$\begin{aligned} \lambda R &\propto \left(29 - \left(\frac{29}{255} * 16 \right) \right) * 2040 \\ \lambda R &\propto 27 * 2040 \end{aligned} \tag{5.8}$$

$$\lambda R \propto 55.448$$

Portanto, a quantidade de mensagens que podem ser perdidas sem prejudicar a recuperação dos dados é proporcional ao tamanho da mensagem.

O processamento de uma sequência de dados ocorre em duas fases distintas: codificação e decodificação. A primeira é realizada pelo nó origem e envolve a divisão dos dados em blocos, que serão codificados por um FEC, e intercalação dos *bytes* de cada bloco para criação das unidades. Na segunda, realizada no destinatário, os *bytes* das unidades são desintercalados e os blocos são decodificados a fim de reconstruir a sequência original de dados. Esse processamento gera uma sobrecarga que varia de acordo com o FEC utilizado e, por serem realizados em nós distintos, podem ser separados em sobrecarga de codificação (λCod) e de decodificação (λDec). Uma vez que não há processamento nos nós intermediários, estes não sofrem sobrecarga relacionada ao processo de codificação e decodificação.

A transmissão de uma mensagem entre dois nós envolve o processamento necessário para enviar os dados de uma interface de rede para a outra. O processamento realizado por eles é aquele necessário para a transmissão de qualquer mensagem. Então a sobrecarga de processamento (λP) gerada pelo processo de codificação e decodificação é dada por:

$$\lambda P = \lambda Cod + \lambda Dec \tag{5.9}$$

Para as mensagens alcançarem o destino, elas precisam ser roteadas através dos nós intermediários (saltos). A cada salto, é preciso enviar o preâmbulo e a mensagem. Portanto envio da mensagem sofre um atraso equivalente à quantidade saltos e à tecnologia utilizada.

CAPÍTULO 6

ANÁLISE DE DESEMPENHO

Este capítulo apresenta a análise de desempenho do EMCOD utilizando o Reed-Solomon como Código de Correção Direta de Erros. O cenário escolhido considera a existência de um nó origem, um nó destino e dez nós intermediários que transmitem a mensagem entre origem e destino. Como o EMCOD não altera o processo de encaminhamento das mensagens, as tecnologias e protocolos utilizados pelas camadas de Rede e Enlace não interferem no seu desempenho. Por esse motivo, considera-se a existência de uma rede homogênea, em que todos os nós possuem as mesmas tecnologias e utilizam os mesmos protocolos. O roteamento utilizado pelos nós é o Epidêmico e a transmissão dos dados é feita pela tecnologia 802.11b, que possui uma taxa de transmissão de dados de 11Mbps. Os resultados foram obtidos através de simulações realizadas no GNU Octave[1]. O código-fonte utilizado nas simulações são apresentados no Apêndice B.

Esta análise utiliza quatro códigos com diferentes capacidades de correção: RS(255,223), RS(511,479), RS(512,496) e RS(127,117). Os códigos selecionados são alguns dos mais utilizados, apresentam boa capacidade de correção e baixa probabilidade de erros [**?**]. A escolha de diferentes capacidades de correção permite avaliar sua relação com a taxa de recuperação de mensagens. Além disso, foram escolhidos dois códigos com mesma capacidade de correção a fim de avaliar a sobrecarga gerada e o impacto dessa sobrecarga para o EMCOD. A Tabela 6.1 apresenta os códigos RS utilizados para análise, suas respectivas capacidades de correção e sobrecarga de dados, representadas pelos *bytes* de paridade adicionados pelo código.

Além da análise dos códigos, é realizada uma comparação com os resultados alcançados pelo NER-DRP. Este protocolo de roteamento apresenta objetivos semelhantes aos do EMCOD e é uma variação do protocolo Epidêmico, que realiza a replicação das mensagens

[1]O GNU Octave é uma linguagem computacional de alto nível, desenvolvida cálculos numéricos. Mais informações podem ser obtidas em http://www.gnu.org/software/octave/

Tabela 6.1: Códigos RS utilizados para análise de desempenho

Código utilizado	*Bytes* de Paridade	Capacidade de Correção
RS(255,223)	32	16
RS(511,479)	32	16
RS(512,496)	16	8
RS(127,117)	10	5

para todos os vizinhos. Além do roteamento das mensagens, o NER-DRP realiza verificação e correção de erros utilizando o código RS(127,117). Devido a essas características, é pertinente realizar uma comparação do seu desempenho com o do EMCOD.

6.1 Sobrecarga de Mensagens

Para viabilizar a entrega de dados em redes cujos nós podem se desconectar definitivamente ou sofrem com longos atrasos para entrega da mensagem, a subcamada CCD divide os dados em mensagens menores. Buscando definir o tamanho ideal para as mensagens, tomou-se como base os quadros criados pela camada de enlace, que no caso das redes 802.11 é de 2304 *bytes* [**?**].

Considerando que dentro do quadro devem estar encapsulados os cabeçalhos das camadas superiores, foram reservados 128 *bytes* para cabeçalhos e blocos de extensão dos agregados, pois a soma dos tamanhos de todos os possíveis cabeçalhos e blocos não ultrapassam esse valor. Os demais 2176 *bytes* passam a ser utilizados pelo campo de dados da subcamada CCD. A escolha deste tamanho se justifica por reduzir a sobrecarga na camada de enlace, uma vez que esta não precisa quebrar os pacotes em quadros menores, ao mesmo tempo em que é possível utilizar o máximo da carga útil suportada pelo padrão.

Para analisar o número de mensagens criadas pela subcamada, foram considerados dados de diferentes tamanhos. Em cada código RS as mensagens possuem tamanhos que variam de acordo com os parâmetros do código. A variação do tamanho se deve ao número de *bytes* que compõem um bloco codificado e, consequentemente, formam uma unidade. A Tabela 6.1.1 mostra o número de unidades agrupadas pelos códigos e o tamanho das mensagens criadas.

Para possibilitar a análise da sobrecarga de mensagens, considerou-se a criação de

Tabela 6.1.1: Tamanho das mensagens criadas pelos códigos Reed-Solomon

Código utilizado	Quantidade de unidades	Tamanho da mensagem	Sobrecarga
RS(255,223)	8	2040	256
RS(511,479)	4	2044	128
RS(512,496)	4	2048	64
RS(127,117)	17	2159	170

uma subcamada CCD sem codificação. A subcamada sem codificação realiza a divisão da mensagem em fragmentos e blocos, porém não codifica os dados e, por isso, não adiciona *bytes* de paridade. Desta forma, os blocos criados formam unidades que serão agrupadas em mensagens com tamanho 2176, que é o tamanho máximo suportado pela subcamada CCD[2]. A Figura 6.1 mostra o número de mensagens criadas pelos códigos, permitindo sua comparação o número de mensagens criadas pela subcamada sem codificação. Analisando os dados apresentados no gráfico, pode-se perceber claramente a sobrecarga gerada pelo FEC.

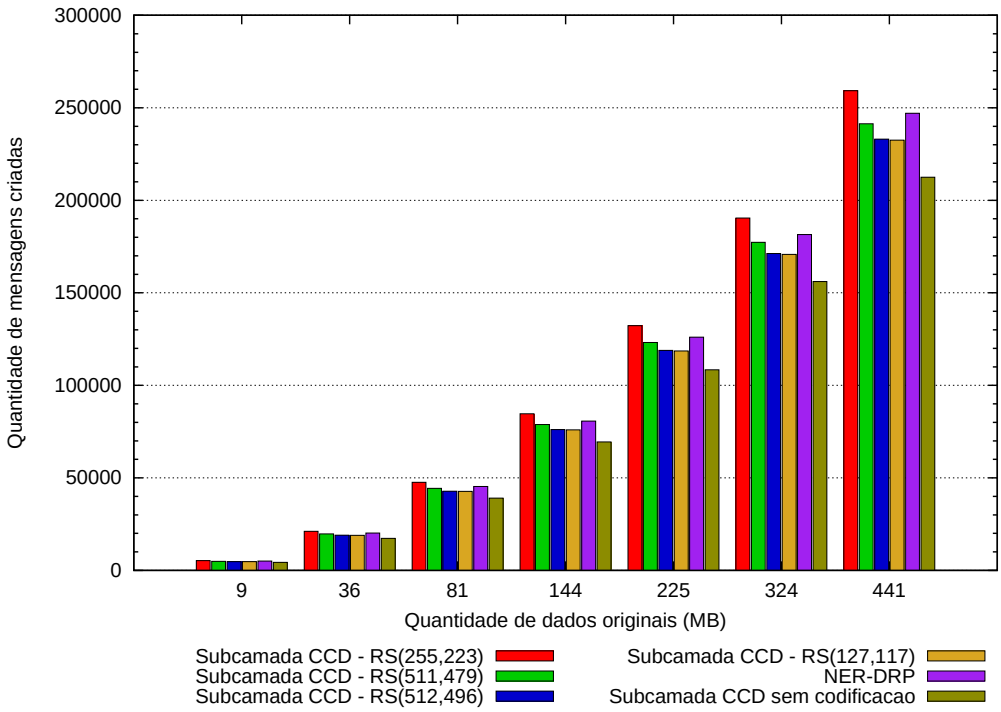


Figura 6.1: Quantidade de mensagens criadas

[2]Tamanho válido para esta análise.

Para realizar a comparação com o NER-DRP e, visando analisar apenas a sobrecarga em quantidade de dados, considerou-se a divisão da mensagem em blocos de tamanho 2032 *bytes*. Assim, os dados originais são divididos em blocos, que são codificados utilizando RS(127,117). A escolha deste tamanho se deve ao fato de que a especificação do NER-DRP define que cada bloco é independente dos demais, então entende-se que cada um possui seu próprio cabeçalho[3]. Desta forma, é necessário reservar espaço para cada um dos cabeçalhos na mensagem, o que reduz o tamanho do campo de dados. Sendo assim, ao invés de agrupar 17 blocos, como é feito na subcamada CCD que utiliza RS(127,117), é possível agrupar apenas 16 blocos em cada mensagem.

Ao comparar a subcamada CCD com o NER-DRP (Figura 6.1), percebe-se que o NER-DRP apresenta uma sobrecarga de mensagens de 16,3% quando comparado com a subcamada CCD sem codificação. Essa sobrecarga é 6,3% superior à da subcamada CCD que utiliza o RS(127,117) e a diferença se deve ao fato do NER-DRP enviar os cabeçalhos referentes a cada bloco criado.

A Figura 6.2 apresenta a sobrecarga de mensagens da subcamada CCD. A adição de *bytes* de paridade pelo Reed-Solomon resulta na criação de mais mensagens do que seriam necessárias se não fosse realizada a codificação dos dados. As mensagens adicionais representam a quantidade de mensagens resultantes do processo de codificação. A quantidade de mensagens criadas pela subcamada CCD, excluindo as mensagens adicionais, pode ser observada na coluna da *subcamada CCD sem codificação* da Figura 6.1.

As Figuras 6.1 e 6.2 mostram que a sobrecarga varia de acordo com o RS utilizado. O número de mensagens aumenta quando o Reed-Solomon adiciona mais *bytes* de paridade e pode ser menor de acordo com a relação entre a sobrecarga e a quantidade de dados originais[4]. A quantidade de mensagens criadas também varia de acordo com a quantidade de *bytes* que compõe cada mensagem, o que é definido considerando as características do RS.

Para calcular o número de mensagens adicionais, tomou-se como base a quantidade

[3]É importante lembrar que o NER-DRP encaminha cada bloco separadamente. Nesta análise os blocos são agrupados para facilitar a análise com a subcamada CCD.

[4]Os dados originais são aqueles que fazem parte da sequência de dados. Em um código RS(255,223) os blocos possuem 223 *bytes* originais.

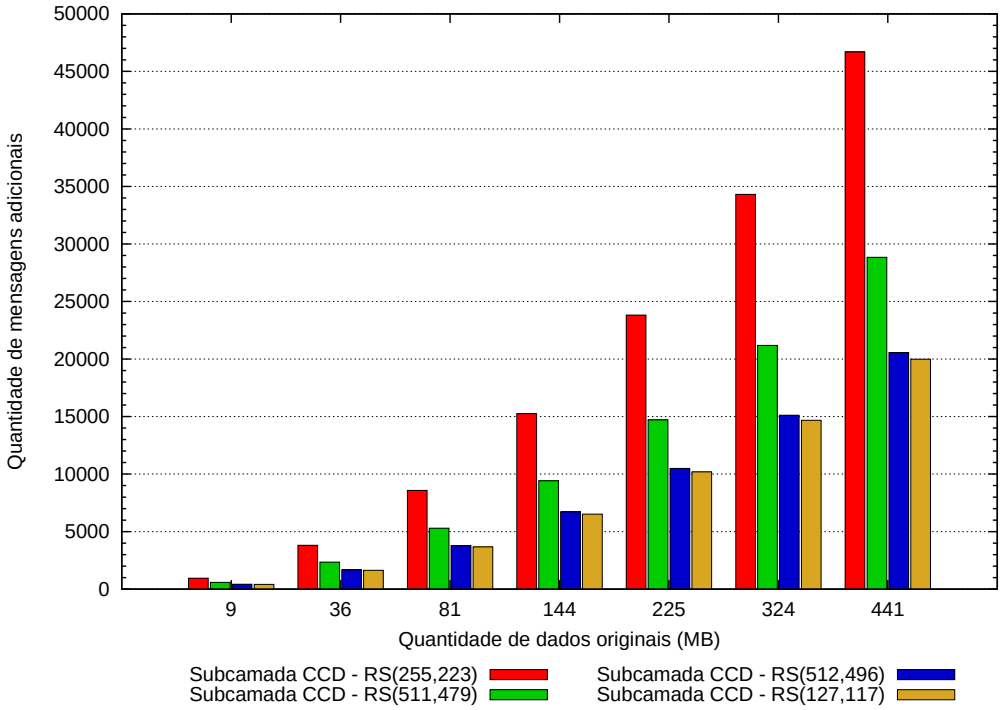


Figura 6.2: Número de mensagens adicionais

de mensagens criadas sem codificação. As menores sobrecargas são geradas pelos códigos RS(127,117) e RS(512, 496), que criam, respectivamente, 9,4% e 9,7% mensagens a mais que a subcamada CCD sem codificação. A baixa sobrecarga gerada por esses códigos se deve à sua menor capacidade de correção. Já o RS(255,223) e o RS(511,479), que possuem as melhores taxas de correção, apresentam as maiores sobrecargas, sendo que o primeiro cria 22% mais mensagens que a subcamada CCD sem codificação e 7,4% mais mensagens que o RS(511,479). A diferença entre esses dois últimos códigos está relacionada à sobrecarga proporcional por unidade, que no RS(511,479) é sensivelmente menor que no RS(255,223). Sendo assim, os códigos que geram menor sobrecarga são aqueles que acrescentam menos *bytes* de paridade em cada bloco.

6.2 Capacidade de Recuperação

Apesar da sobrecarga gerada, o processo de codificação permite a recuperação dos dados a partir da recepção de um determinado número de mensagens. Essa recuperação pode melhorar significativamente o desempenho de redes que possuem como requisito a garantia de entrega, principalmente se a rede for esparsa, pois nessas redes é inviável solicitar o reenvio de mensagens corrompidas ou não recebidas[5]. A Figura 6.3 mostra a quantidade de mensagens que precisam ser recebidas para que a subcamada CCD seja capaz de recuperar os dados originais. A área hachurada representa as mensagens que foram enviadas e podem ser perdidas sem prejudicar a recuperação de dados. A subcamada CCD sem codificação não é capaz de corrigir erros e, por isso, precisa receber todas as mensagens e seus dados foram adicionados ao gráfico para simples comparação. O NER-DRP precisa receber todos os blocos enviados, pois a recuperação de dados é realizada a cada salto. Como ele não pode recuperar mensagens não recebidas no destinatário, seus dados foram omitidos do gráfico.

Na Figura 6.3, a quantidade de mensagens necessárias para a recuperação varia de acordo com a quantidade de dados, capacidade de correção do RS e tamanho do bloco codificado. O código que possui melhor taxa de recuperação é aquele que possui tamanho de bloco 255, podendo perder 6,3% das mensagens enviadas. Em comparação à subcamada CCD sem codificação, o RS(255,223) precisa receber 14,3% mensagens a mais.

Apesar do RS(511,479) possuir a mesma capacidade de correção que o RS(255,223), a capacidade de recuperação sofre uma queda significativa devido à quantidade de blocos agrupados. Isso se deve ao fato de que o bloco criado com este código possui maior quantidade de dados originais para a mesma capacidade de correção, o que reduz a relação entre dados e paridade. Assim, este código pode perder apenas 3,1% das mensagens enviadas sem prejudicar a recuperação da sequência de dados. Porém, como este gera uma sobrecarga de dados menor, seu desempenho é 10% inferior ao da subcamada CCD sem codificação e 4,3% superior ao da subcamada CCD com RS(255,223).

Os códigos cujo desempenho se aproximam mais da subcamada CCD sem codificação

[5]Em cenários esparsos, o envio de uma mensagem pode demorar desde algumas horas até dias.

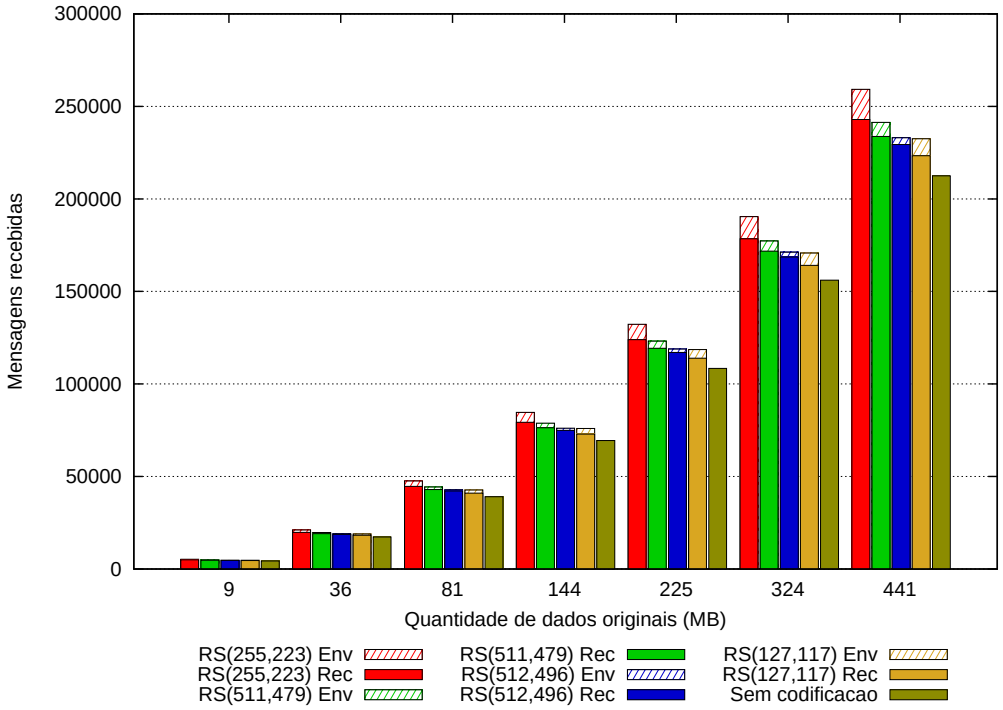


Figura 6.3: Capacidade de recuperação de mensagens utilizando Reed-Solomon

são os que apresentam menores taxas de correção. A quantidade de blocos agrupados por unidade gera um impacto positivo e faz com que o RS(127,117) alcance um bom desempenho. Utilizando este código é possível perder aproximadamente 3,9% das mensagens enviadas, um desempenho 0,8% superior ao RS(511,479). Além disso, o RS(127,117) precisa receber apenas 5,1% mais mensagens que a subcamada CCD sem codificação, o que o torna o código com a melhor relação entre sobrecarga e capacidade de recuperação.

Por outro lado, o código com pior taxa de recuperação é o RS(512,496). Além de apresentar sobrecarga superior ao RS(127,117) alcança uma taxa de recuperação de apenas 1,6%, sendo necessário entregar cerca de 8% mais mensagens que a subcamada CCD sem codificação.

A partir destes dados, percebe-se que os códigos que alcançam melhores taxas de recuperação de mensagens são aqueles que possuem a melhor relação entre tamanho de bloco e capacidade de correção. Apesar de gerar uma sobrecarga superior aos demais, o código RS(255,223) alcança resultados melhores quanto à recuperação de mensagens. No

entanto, a melhor relação entre sobrecarga e capacidade de recuperação é obtida com o RS(127,117). O fator comum a ambos os códigos é o agrupamento de mais unidades por mensagem, o que melhora a taxa de recuperação da sequência de dados.

6.3 Sobrecarga de Energia

O processamento de uma mensagem envolve seu tratamento pelas camadas de rede e sua transmissão até que alcance o destino. O tratamento das mensagens varia de acordo com a tecnologia e protocolos utilizados pelos nós da rede. No entanto, a utilização de um FEC para codificar os dados, gera uma sobrecarga relacionada ao processo de codificação e decodificação. Além disso, por gerar sobrecarga de mensagens, ocorre um processamento extra para a transmissão dos dados adicionais.

A estratégia utilizada pelo EMCOD gera esta sobrecarga de codificação apenas na origem e no destino, preservando os nós intermediários, que realizarão apenas o processamento necessário para transmissão das mensagens. Conhecendo a complexidade do FEC utilizado, é possível calcular a energia utilizada pelos processos de codificação e decodificação. A partir dos estudos de Biard e Noguet [**?**] ´e possível calcular a energia consumida pelos códigos RS utilizados para análise.

A Seção 3.2.3 apresenta um resumo dos estudos de Biard e Noguet, bem como a explicação sobre o cálculo da complexidade computacional. O cálculo da sobrecarga de processamento do Reed-Solomon toma como base a Equação 3.26. No entanto, a referida equação agrupa os dados relativos tanto à codificação quanto à decodificação. Como a subcamada CCD realiza ambos processos separadamente, é necessário desmembrar a equação de forma a obter os valores separadamente. O desmembramento considera as informações contidas nas Tabelas 3.2.2 e 3.2.3. A Equação 3.26 está dividida em duas: a Equação 6.1a que calcula a energia consumida pelo processo de codificação e a Equação 6.1b que calcula a energia consumida para decodificação. Os valores apresentados pelas duas equações são referentes ao consumo de energia por *bit* e, como a subcamada CCD trabalha com *bytes*,

os valores foram multiplicados por 8.

$$CCIB\_CODE = \frac{1}{m}\left[\frac{2t}{r} * (GFadd + GFmul\alpha i + GFreg)\right] \tag{6.1a}$$

$$\begin{aligned} CCIB\_DECODE = & \frac{1}{m}\left[\frac{1-r}{2r}(4t+1)GFmul + \frac{1}{r}(4t-1)GFmul\alpha i\right] + \\ & \frac{1}{m}\left[\left(\frac{1}{r}(4t-1) + \frac{1-r}{2r}(4t+1)\right)GFadd + \frac{1-r}{r}GFinv\right] + \\ & \frac{1}{m}\left[\left(\frac{1}{r}(4t-1) + \frac{1-r}{2r}(6t-1)\right)GFreg + 1GFmem\right] \end{aligned} \tag{6.1b}$$

Para obter toda energia extra consumida a partir da utilização da subcamada CCD, é preciso considerar ainda a energia de transmissão das mensagens, que varia de acordo com a tecnologia de rede utilizada. Uma vez que esta análise considera a utilização do 802.11b como tecnologia de rede, utilizou-se os dados obtidos por Bannack e Albini [**?**] para transmissão de dados. A Figura 6.4 apresenta uma comparação entre as sobrecargas geradas pela subcamada CCD que utilizam codificação. Uma comparação entre todos os códigos é apresentada na Figura 6.5.

Os códigos que geram menos processamento extra são os RS(512,496) e RS(511,479), cuja sobrecarga é de 32,1% e 36,6%, respectivamente. Esse percentual se deve ao fato dos códigos possuírem melhor relação entre dados originais e *bytes* de paridade. Assim, apesar do código RS(255,223) ter a mesma capacidade de correção que o RS(511,479), o primeiro codifica menos dados originais por bloco e gasta 9,9% mais energia que o segundo para recuperar a mesma quantidade de dados. O RS(127,117) possui um tamanho de bloco bastante reduzido e, por esse motivo, apresenta alta sobrecarga de processamento, despendendo 38,9% mais energia que a subcamada CCD sem codificação para entregar e recuperar os dados originais.

A energia consumida pelos processos de codificação e decodificação é significativa. Por isso, realizar esse processamento extra em cada nó intermediário, como faz o NER-DRP, torna-se muito custoso e, em muitos casos, pode inviabilizar a sua utilização. A Figura 6.5 mostra um comparativo entre a energia consumida pelo NER-DRP e a subcamada CCD. Neste comparativo, considera-se que os nós intermediários apenas decodificam os blo-

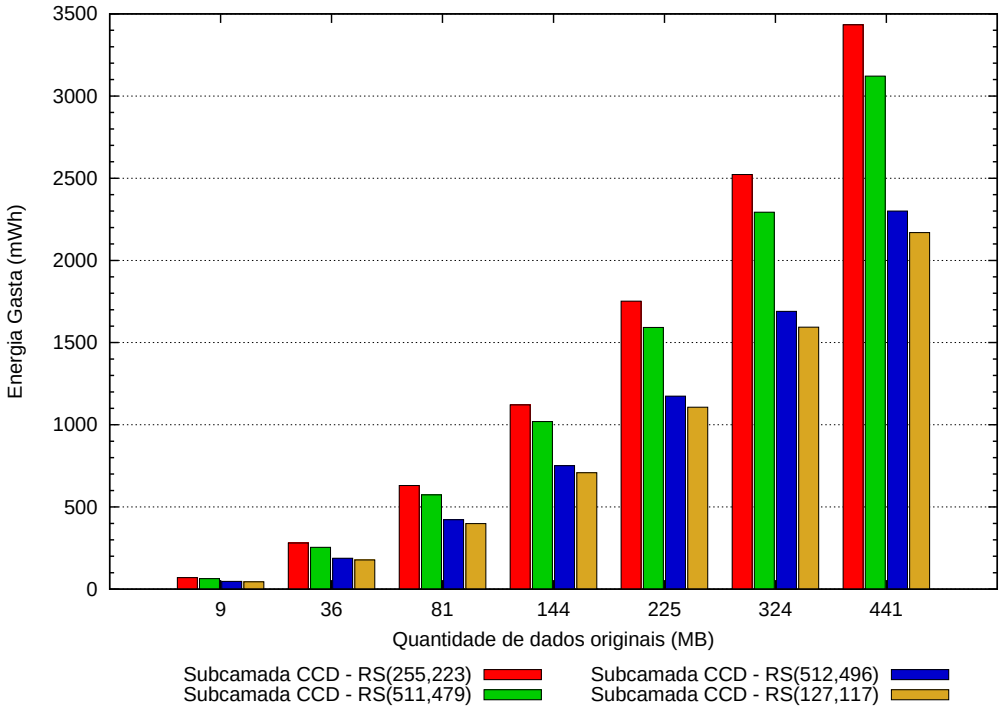


Figura 6.4: Sobrecarga de Energia

cos, não sendo necessário codificá-los novamente, pois esta análise não considera perda de dados. Adicionalmente, são apresentados os da subcamada CCD sem codificação, que apresenta apenas a energia consumida pelo processo de transmissão das mensagens. De acordo com os dados, o NER-DRP gasta 74,1% mais energia que a subcamada sem codificação para enviar a mesma quantidade de dados.

Apesar da utilização de um FEC ser capaz de maximizar a taxa de entrega de mensagens ao destinatário, por detectar e corrigir erros durante a transmissão, realizar este processo a cada salto não é vantajoso. Assim, o esquema utilizado pela subcamada CCD é eficiente, pois permite a perda de algumas mensagens com uma sobrecarga aceitável.

6.4 Atraso para Entrega das Mensagens

Um dos pontos mais críticos quando se trata de transmissão de mensagens em redes DTNs são os atrasos para sua entrega ao destinatário. Nessas redes, evitar retransmissões pode

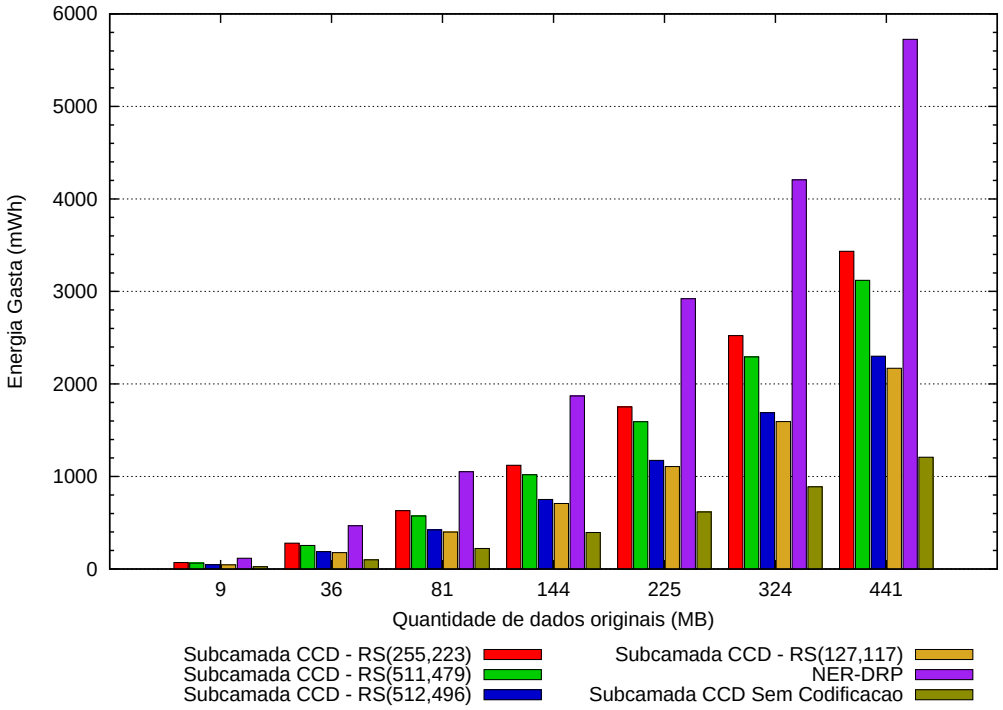


Figura 6.5: Comparação das sobrecargas de energia

causar um impacto significativo no desempenho. Visando minimizar os impactos causados por essas restrições, a subcamada CCD utiliza correção de erros no destino. Se um, ou alguns, nós entregarem ao destinatário um número suficiente de mensagens, então já será possível recuperar a sequência de dados.

O NER-DRP utiliza correção de erros a cada salto para evitar corrupção das mensagens, causada por problemas na sua transmissão ou armazenamento. Caso um determinado bloco não possa ser recuperado através do código de correção de erros, o NER-DRP pode solicitar o seu reenvio para outro nó. No entanto, para que o novo bloco seja recebido é necessário estabelecer contato com um nó que possua a custódia deste bloco, o que nem sempre ocorre.

Para comparar o desempenho das duas estratégias, foi calculada a quantidade de mensagens que podem ser perdidas sem prejudicar a recuperação dos dados. A Figura 6.6, mostra o atraso para entrega, considerando a quantidade mínima de mensagens que devem chegar ao destino e possibilitar a recuperação da sequência de dados. Como o NER-DRP

não permite a perda de nenhum bloco, considerou-se que todos os blocos foram recebidos pelo destinatário. As estimativas de atraso consideram um intervalo de 60 segundos entre contatos, além do tempo para envio e recebimento da mensagem. Nota-se que os resultados variam de acordo com o tamanho da sequência de dados. Quanto maior a mensagem, maior a diferença para a subcamada CCD sem codificação.

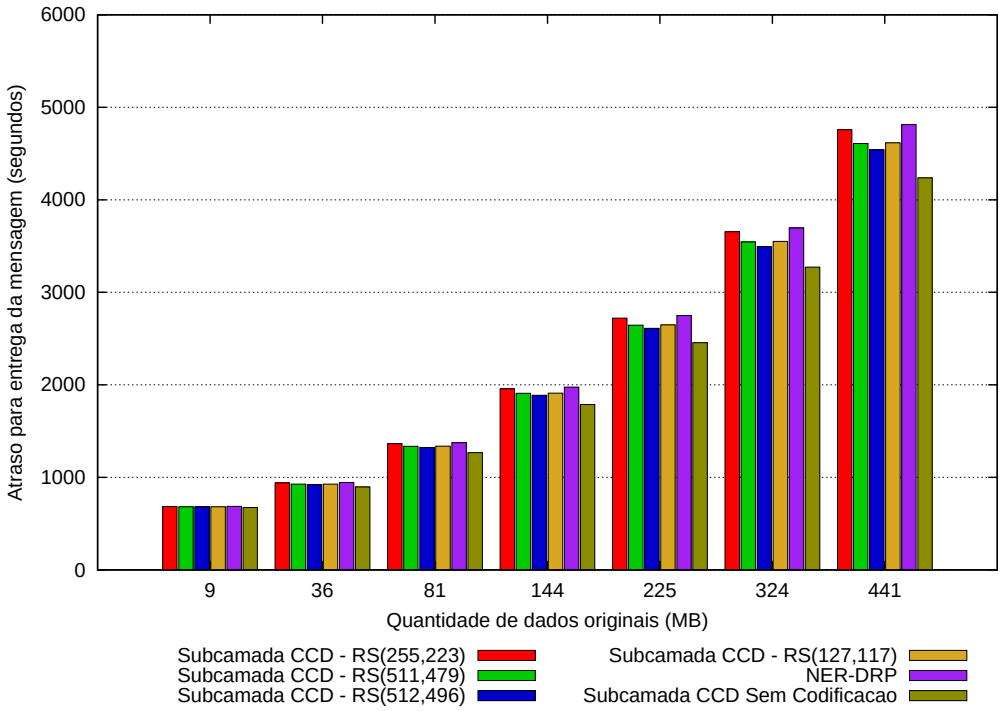


Figura 6.6: Atraso para entrega de mensagens

O código que apresenta melhor desempenho é o RS(512,496), que em média demora 4,8% mais tempo que a subcamada CCD para entregar as mensagens. O segundo melhor é o RS(511,479), que demora 1,1% mais tempo que o anterior. O RS(127,117) apresentou desempenho muito próximo ao RS(511,479), sendo 6% inferior ao da subcamada CCD sem codificação e apenas 0,1% inferior ao primeiro código. O pior resultado foi alcançado pelo RS(255,223), que é 8,3% mais lento que a subcamada CCD sem codificação. Essa lentidão se deve à quantidade de mensagens que precisam ser entregues ao destinatário. No entanto, quando comparado ao NER-DRP, o RS(255,223) é em média 0,9% mais rápido, chegando a ser 1,3% mais rápido para entrega de mensagens maiores.

A análise de melhor caso considera um atraso mínimo entre contatos, pois verifica apenas as diferenças de desempenho entre subcamada CCD, o NER-DRP e a subcamada sem Codificação. Esta última foi utilizada com o objetivo de verificar o impacto causado pelo reenvio de mensagens através da rede. No entanto, é importante ressaltar que o atraso pode ser muito superior, quando são considerados os atrasos entre contatos. Para o NER-DRP, é necessário contabilizar ainda o tempo gasto pelo processo de decodificação, que é realizado a cada recepção de bloco e foi desconsiderado nesta análise.

Com relação à subcamada CCD sem codificação, os dados apresentam um acréscimo no atraso para entrega de suas mensagens de 38%. Quando comparado com o RS(127,117), este atraso apresenta uma diferença média de 38,3%, chegando a alcançar picos de 63,6%. O RS(255,223) é o que possui menor diferença, sendo aproximadamente 36,8% mais rápido que a subcamada CCD sem codificação[6]. Já o NER-DRP é cerca de 40% mais lento que a subcamada CCD com RS(255,223), o que o torna 8,6% mais demorado que a subcamada CCD sem codificação e 43,7% mais lento que o RS(127,117). Sendo assim, sua utilização em ambientes que sofrem atrasos para entrega pode ser inviável.

6.5 Análise dos Resultados

Diante dos resultados apresentados, conclui-se que é vantajoso utilizar a subcamada CCD em cenários que sofrem com longos atrasos para entrega de mensagens. A subcamada CCD aliada a um FEC que apresente baixa sobrecarga de dados pode reduzir o tempo para recuperação dos dados originais. Essa possibilidade se deve ao fato de combinar os dados das mensagens e utilizar um FEC para recuperar os dados não recebidos.

O EMCOD permite recuperar os dados originais mesmo que ocorra corrupção de mensagens. O esquema necessita de pouco processamento adicional, o que gera sobrecarga de energia reduzida. Estratégias similares, como a utilizada pelo NER-DRP, apresentam menor desempenho com maior consumo de recursos. Sendo assim, utilizar o EMCOD em redes que possuem altos índices de atraso melhora o desempenho da rede.

[6]Considerando um cenário com atraso mínimo.

CAPÍTULO 7

ANÁLISE DE DESEMPENHO CONSIDERANDO PERDAS

Neste capítulo são apresentados dados sobre o desempenho da subcamada CCD em cenários que apresentam perdas de pacotes. Para obter os resultados, considerou-se o envio de dados com tamanho de 144 MB. Em relação às perdas, foram tomados valores que variam entre a menor e a maior perda suportadas pelos códigos analisados. Para esta análise é considerada a perda de uma quantidade fixa de mensagens. O valor máximo utilizado como referência é o do RS(255,223), o FEC que apresenta maior capacidade de correção, que é capaz de perder 79.328 mensagens das 84.639 criadas. O código-fonte utilizado nas simulações são apresentados na Seção B.2 do Apêndice B.

Para o tamanho de mensagens selecionado, cada código é capaz de recuperar uma quantidade de mensagens. A Tabela 7.1 apresenta a quantidade de mensagens criadas e a quantidade de mensagens que podem ser perdidas pelos códigos analisados, sem que a perda prejudique a recuperação da sequência de dados.

Tabela 7.1: Capacidade de recuperação de mensagens

Código utilizado	Mensagens Criadas	Capacidade de Recuperação
RS(255,223)	84.639	79.328 (6,3%)
RS(511,479)	78.808	76.340 (3,1%)
RS(512,496)	76.107	74.498 (1,6%)
RS(127,117)	75.916	72.927 (3,9%)
CCD sem codificação	69.392	-
NER-DRP	80.660	-

Conforme discutido nos capítulos anteriores, a subcamada CCD aguarda a recepção das mensagens para efetuar a recuperação dos dados originais. Caso não seja possível recuperar esses dados, devido à perda de mais mensagens do que o FEC suporta, faz-se necessário solicitar o reenvio de algumas mensagens. Sendo assim, esta análise considera a solicitação do reenvio do menor número de mensagens necessárias para a recuperação dos dados. A Figura 7.1 apresenta a quantidade de mensagens enviadas, incluindo avisos de

não recebimento e o reenvio das mensagens não recebidas. A área hachurada representa a sobrecarga gerada, devido ao envio de pacotes NACK e reenvio das mensagens.

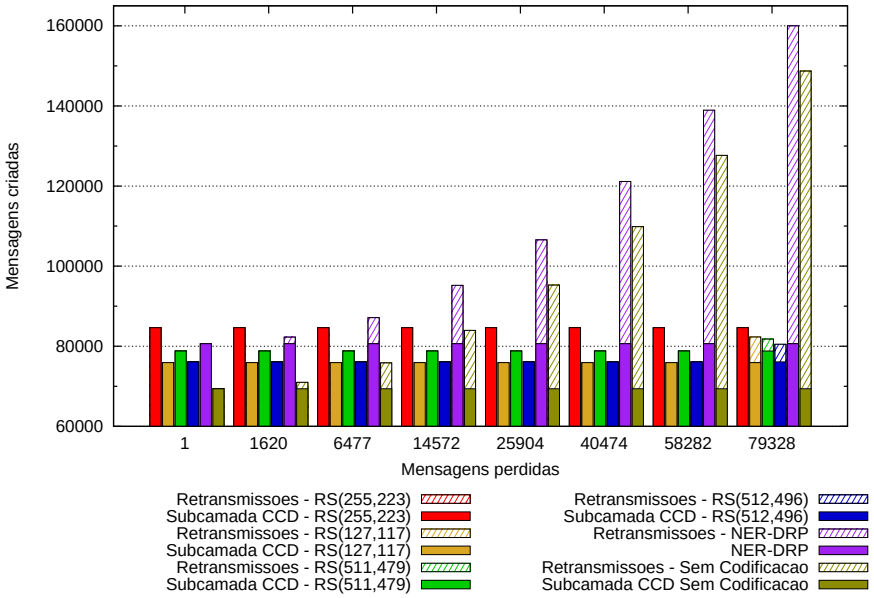


Figura 7.1: Quantidade de mensagens enviadas

A partir dos dados apresentados, pode-se notar que a ocorrência de perda de mensagens pode causar impacto significativo sobre o desempenho da rede. A subcamada CCD sem codificação sofre sobrecarga relevante, que chega a 114%[1]. Já o NER-DRP apresenta um resultado um pouco melhor, alcançando índice de sobrecarga de 98%. A diferença entre os percentuais se deve ao fato de o NER-DRP criar mais mensagens e o número de mensagens perdidas ser fixo. O desempenho da subcamada CCD, utilizando RS para correção de erros, pode ser considerado bom pois, ainda que o ambiente cause perda de quase oitenta mil mensagens, o RS(127,117) gera uma sobrecarga de pouco mais de 8%, contra 5% do RS(511,479) e 3% do RS(512,496). Sendo assim, apesar da subcamada CCD gerar uma sobrecarga inicial relacionada ao processo de codificação, existe a compensação pela recuperação das mensagens.

Além da sobrecarga de mensagens, é preciso considerar o incremento no atraso para entrega ao destinatário gerado pela ocorrência de perdas. O atraso é calculado consi-

[1]Considerando a parte hachurada.

derando o tempo para envio do preâmbulo, dos dados da mensagem e um atraso de 60 segundos entre contatos. Esses tempos são considerados para cada salto. Adicionalmente, considera-se um atraso de 60 segundos antes do envio de avisos de não recebimento e o atraso decorrente do processo de decodificação da mensagem. A Figura 7.2 apresenta os dados referentes ao atraso entre o envio dos dados pelo nó origem e recepção de todas as mensagens pelo destinatário. A área hachurada representa o atraso referente ao reenvio de dados, considerando avisos de não recebimento e reenvio das mensagens.

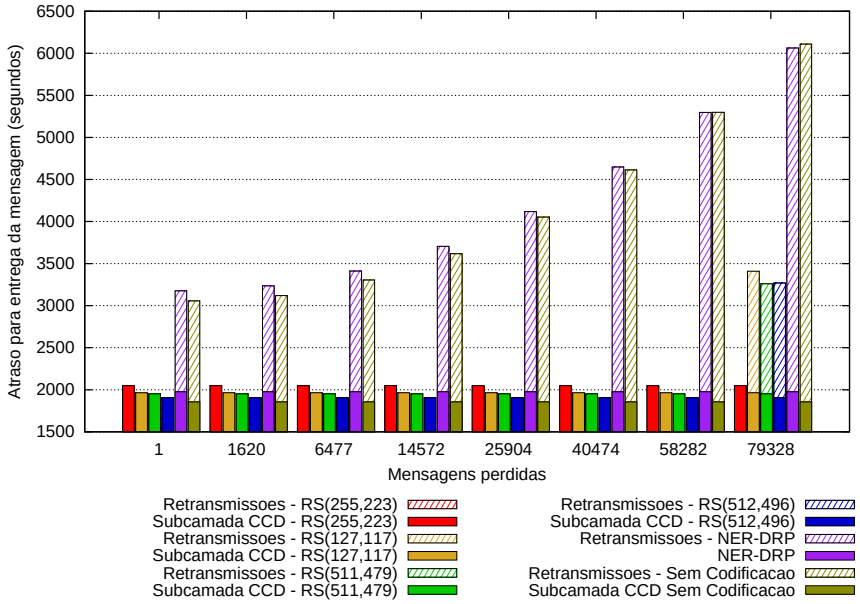


Figura 7.2: Atraso para entrega de mensagens

Quando não ocorre perda de mensagens, a subcamada CCD sem codificação apresenta desempenho superior à subcamada CCD com Reed-Solomon. No entanto, a perda de uma única mensagem é suficiente para justificar a utilização de codificação. Quando ocorre a perda de uma mensagem, a subcamada CCD sem codificação apresenta um atraso 50% superior ao do pior desempenho alcançado com codificação e o NER-DRP alcança índices superiores a 55%. À medida que a quantidade de mensagens perdidas aumenta, o tempo gasto para o NER-DRP e a subcamada CCD sem codificação entregar os dados pode até triplicar, em relação a um cenário sem perdas.

A subcamada CCD com RS sofre ampliação do atraso quando o número de mensagens

perdidas é superior à sua capacidade de correção. Para os valores analisados, o pior resultado é aquele alcançado pelo RS(127,117), que sofre degradação do seu desempenho de mais de 65%, o que está relacionado à sua baixa capacidade de correção de erros. Em contrapartida, os códigos RS(511,479) e RS(512,496) obtêm resultados semelhantes, sendo que seus tempos aumentam em aproximadamente 59%. Diante desses resultados, fica evidente a viabilidade de se utilizar a subcamada CCD em ambientes que sofrem longos atrasos para envio de mensagens.

A utilização de Código de Correção Direta de Erros dentro do sistema de transmissão de mensagens aumenta a complexidade computacional desse sistema. Em cenários que sofrem com desconexões e consequente perda de mensagens, sua utilização pode melhorar significativamente o desempenho da rede, podendo gerar, inclusive, melhor utilização dos seus recursos. Ao analisar o desempenho da subcamada CCD em cenários que apresentam perdas de mensagens, a economia de recursos e melhoria do desempenho torna-se evidente. No entanto, é preciso avaliar cuidadosamente o cenário para decidir sobre a melhor estratégia ou FEC a ser utilizado.

Em cenários que apresentam baixos índices de perda de mensagens e o atraso não é um quesito importante, não é aconselhável implementar a subcamada CCD, pois sua utilização gera sobrecarga desnecessária ao sistema. Por outro lado, se o tempo para entrega de mensagens for um requisito relevante, sua utilização passa a ser recomendada. A decisão sobre qual FEC associar à subcamada CCD deve considerar as características do cenário em questão.

Para redes que possuem baixos percentuais de perdas de pacotes, pode-se utilizar códigos com baixa capacidade de correção e melhor relação entre sobrecarga de processamento e de dados. Em contrapartida, se a rede sofre muitas perdas e longos atrasos, torna-se viável utilizar um código que apresente melhor capacidade de correção, ainda que sua sobrecarga seja superior.

Com relação aos códigos analisados, o RS(512,496) é o mais indicado para ambientes que apresentam baixas taxas de perda ou nos quais o consumo de energia é um quesito importante, pois apresenta baixa sobrecarga de mensagens. O RS(127,117) mostrou-se

adequado para cenários intermediários, pois possui uma boa relação entre sobrecarga e atraso. Em cenários com altas taxas de perda de pacotes, o RS(255,223) é o código com melhor desempenho devido à sua capacidade de recuperação de mensagens, apesar de gerar maior sobrecarga.

Sendo assim, é possível dizer que a escolha do código e o desempenho final da subcamada CCD estão fortemente relacionados às características da rede. De qualquer forma, a subcamada CCD apresenta melhor desempenho que o NER-DRP, devido à sua alta sobrecarga na rede. Em todos os cenários analisados, o NER-DRP alcança resultados inferiores aos alcançados pelo EMCOD. A escolha varia entre utilizar o EMCOD ou manter a estratégia tradicional com uma camada de Agregação sem subdivisões.

CAPÍTULO 8

CONSIDERAÇÕES FINAIS

Este capítulo apresenta as considerações finais sobre o esquema proposto e propõe alguns trabalhos futuros. A seção 8.1 mostra os resultados alcançados pelo EMCOD e a motivação para utilizá-lo em uma rede DTN. A seção seguinte lista os trabalhos a serem realizados após a defesa desta dissertação.

8.1 Entrega de Mensagens Codificadas em Redes DTN

As redes DTN possuem características que inviabilizam o serviço de entrega confiável de protocolos como o TCP, tornando a entrega de mensagens ao destinatário um grande desafio. Redes especialmente desafiadoras são aquelas que sofrem com longos atrasos e constantes desconexões, características que podem atrasar ou impedir a entrega da mensagem ao destinatário. Cientes do problema, pesquisadores tem desenvolvido protocolos que aumentam a taxa de entrega de mensagens íntegras. No entanto, uma análise criteriosa sobre os protocolos encontrados demonstra que eles geram sobrecarga de processamento desnecessária, tornando oportuna a criação de uma estratégia que maximize a taxa de entrega e reduza o tempo para recuperação dos dados, ao mesmo tempo em que gere baixa sobrecarga.

Com essa perspectiva é proposto o EMCOD, um esquema para entrega de mensagens que utiliza codificação de rede e divisão da sequência de dados em mensagens menores para transmissão. O EMCOD altera a estrutura da camada de agregação dividindo-a em duas subcamadas, sendo que uma agrupa todas as funcionalidades já atribuídas à essa camada e a outra, chamada Controle de Codificação de Dados, implementa o esquema proposto. A subcamada adicionada trabalha de forma independente dos protocolos utilizados pelas DTNs permitindo que a mensagem seja roteada por nós que não a implementem, pois o processamento dos dados é restrito à origem e ao destino. A inexistência de processamento

adicional nos nós intermediários reduz a sobrecarga e economiza recursos de rede.

A subcamada CCD utiliza um FEC para codificação dos dados que permite ao destinatário recuperar a sequência original, mesmo que algumas mensagens não tenham sido entregues. A codificação utilizada é amplamente implementada por sistemas de comunicação e não está restrita a um código específico, permitindo que seja utilizado aquele que apresente melhores resultados para cada cenário. A codificação permite ao destinatário reconstruir a mensagem a partir da recepção de alguns blocos íntegros, não sendo necessário aguardar a entrega de todos. Assim, é possível reduzir o tempo para reconstrução da sequência de dados e minimizar a sobrecarga na rede, uma vez que o envio de avisos de não recebimento é reduzido sensivelmente.

A construção das mensagens a partir de dados intercalados provê ao EMCOD uma capacidade de recuperação da sequência original que é superior à capacidade de correção do FEC e reduz o tempo para recuperação da sequência original. A análise de desempenho mostra a redução no tempo para entrega de mensagens em aproximadamente 50%, o que viabiliza a utilização do esquema em redes que sofrem longos atrasos e perdas de mensagens. A redução no número de mensagens transferidas pode chegar a 60%, melhorando significativamente o desempenho global da rede.

Obviamente, o esquema adiciona sobrecarga de processamento nos nós origem e destino e, por isso, não é aplicável em todos os casos, sendo necessária uma avaliação criteriosa sobre a rede e suas características. Apesar da sobrecarga de processamento na origem e no destino, a sobrecarga gerada sobre os demais nós da rede é mínima, restringindo-se ao encaminhamento de algumas mensagens adicionais. Um fator importante e que gera impacto significativo no desempenho do EMCOD é o FEC utilizado para codificação. A escolha da codificação apropriada é crucial para alcançar bons resultados. A partir da utilização de um FEC que apresenta bons resultados para o cenário em que será utilizado, é possível reduzir sensivelmente o tempo para entrega das mensagens e manter a sobrecarga em níveis inferiores àqueles alcançados quando o esquema não é implementado.

8.2 Trabalhos Futuros

O desempenho do EMCOD é fortemente afetado pelo FEC utilizado para codificação de dados. Um trabalho futuro consiste na análise do seu desempenho utilizando outros códigos, como LDPC e Códigos Turbo. Estes são amplamente utilizados para codificação de rede em sistemas modernos de transmissão de dados e podem apresentar bons resultados quando incorporados à subcamada CCD.

Acredita-se que além da redução do tempo e da sobrecarga, o EMCOD pode aumentar a segurança da rede. Esse aumento de segurança se deve à estratégia utilizada para construção da mensagem, que inviabiliza a recuperação de uma sequência de dados a partir da recepção de poucas mensagens. Como a avaliação sobre a segurança foge ao escopo deste trabalho, esta poderá ser tratada em trabalhos futuros.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Salman Ali, Junaid Qadir, e Adeel Baig. Routing protocols in Delay Tolerant Networks - A Survey. *International Conference on Emerging Technologies*, páginas 70–75. IEEE Computer Society, outubro de 2010.

[2] Emil Artin. *Galois Theory*. Dover Publications, New York, USA, 1ª edição, 1998.

[3] Aruna Balasubramanian, Brian Levine, e Arun Venkataramani. DTN routing as a resource allocation problem. *Conference on Applications, technologies, architectures, and protocols for computer communications*, volume 37, páginas 373–384. ACM, outubro de 2007.

[4] Angelo Bannack e Luiz Carlos Pessoa Albini. Investigating the Load Balance of Multi-Path Routing to Increase the Lifetime of a MANET. *4th IEEE International Conference on Circuits and Systems for Communications*, páginas 109–113. IEEE Computer Society, maio de 2008.

[5] Claude Berrou, Alain Glavieux, e Punya Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1. *IEEE International Conference on Communications, 1993*, volume 2, páginas 1064–1070. IEEE Computer Society, maio de 1993.

[6] Jörg Bewersdorff. *Galois Theory for Beginners: A Historical Perspective*, volume 35 of *Student Mathematical Library*. American Mathematical Society, Rhode Island, USA, 2006.

[7] Lionel Biard e Dominique Noguet. Reed-Solomon Codes for Low Power Communications. *Journal of Communications*, 3(2):13–21, abril de 2008.

[8] Richard Burden e Douglas Faires. *Numerical Analisys*. Cengage Learning, Boston, USA, 8ª edição, 2010.

[9] John Burgess, Brian Gallagher, David Jensen, e Brian Neil Levine. MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks. *IEEE International Conference on Computer Communications*, páginas 1–11. IEEE Computer Society, abril de 2006.

[10] Scott Burleigh, Adrian Hooke, Leigh Torgerson, Kevin Fall, Vint Cerf, Bob Durst, Keith Scott, e Howard Weiss. Delay-tolerant networking: an approach to interplanetary Internet. *IEEE Communications Magazine*, 41(6):128–136, junho de 2003.

[11] Yue Cao e Zhili Sun. Routing in Delay/Disruption Tolerant Networks: A Taxonomy, Survey and Challenges. *IEEE Communications Surveys & Tutorials*, 15(2):654–677, maio de 2013.

[12] Ling-Jyh Chen, Chen-Hung Yu, Tony Sun, Yung-Chih Chen, e Hao-hua Chu. A hybrid routing approach for opportunistic networks. *SIGCOMM workshop on Challenged networks*, páginas 213–220. ACM, setembro de 2006.

[13] Kun-Cheng Chung, Yi-Chin Li, e Wanjiun Liao. Exploiting Network Coding for Data Forwarding in Delay Tolerant Networks. *IEEE 71st Vehicular Technology Conference*, páginas 1–5. IEEE Computer Society, maio de 2010.

[14] Barry A. Cipra. The Ubiquitous Reed Solomon Codes. *SIAM News*, 26(1):2–4, janeiro de 1993.

[15] Carina T. de Oliveira, Marcelo D. D. Moreira, Marcelo G. Rubinstein, Luís Henrique M. K. Costa, e Otto Carlos M. B. Duarte. Redes Tolerantes a Atrasos e Desconexões. *Simpósio Brasileiro de Redes de Computadores*. SBC, maio de 2007.

[16] Peter Elias. Error-Free Coding. Relatório Técnico 285, Massachusetts Institute of Technology, setembro de 1954.

[17] Peter Elias. Coding for Noisy Channels. *IRE Convention Record*, 4:37–46, março de 1955.

[18] Kevin Fall. A delay-tolerant network architecture for challenged internets. *Conference*

on Applications, technologies, architectures, and protocols for computer communications, páginas 27–34. ACM, agosto de 2003.

[19] Behrouz A. Forouzan. *Comunicação de Dados e Redes de Computadores*. Bookman, Porto Alegre, 3ª edição, 2006.

[20] László Fuchs. *Infinite Abelian Groups*, volume 36-I. Academic Press, New York, 1970.

[21] Matthew Gast. *802.11 Wireless Networks: The Definitive Guide*. O'Reilly Media, Sepastopol, CA, 2ª edição, abril de 2005.

[22] Zheng Guo, Bing Wang, e Jun-Hong Cui. Prediction Assisted Single-Copy Routing in Underwater Delay Tolerant Networks. *IEEE Global Telecommunications Conference*, páginas 1–6. IEEE Computer Society, dezembro de 2010.

[23] Venkatesan Guruswami e Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, setembro de 1999.

[24] Richard Wesley Hamming. Error Detecting and Error Correcting Codes. *The Bell System Technical Journal*, 26(2):147–160, abril de 1950.

[25] Simon Haykin. *Sistemas de Comunicação - Analógicos e Digitais*. Bookman, Porto Alegre, 4ª edição, 2004.

[26] Simon Haykin e Michael Moher. *Sistemas Modernos de Comunicação Wireless*. Bookman, Porto Alegre, 2008.

[27] Hwei P. Hsu. *Teoria e Problemas de Comunicação Analógica e Digital*. Schaum. Bookman, Porto Alegre, 2ª edição, 2006.

[28] Chung-Ming Huang, Kun-chan Lan, e Chang-Zhou Tsai. A Survey of Opportunistic Networks. *International Conference on Advanced Information Networking and Applications - Workshops*, páginas 1672–1677. IEEE Computer Society, março de 2008.

[29] Yin-Ki Ip, Wing-Cheong Lau, e On-Ching Yue. Forwarding and Replication Strategies for DTN with Resource Constraints. *IEEE 65th Vehicular Technology Conference*, páginas 1260–1264. IEEE Computer Society, abril de 2007.

[30] Sushant Jain, Kevin Fall, e Rabin Patra. Routing in a Delay Tolerant Network. *SIGCOMM Computer Communication Review*, 34(4):145–158, outubro de 2004.

[31] Sarah J. Johnson. *Iterative Error Correction: Turbo, Low-Density Parity-Check and Repeat-Accumulate Codes*. Cambridge University Press, New York, USA, 2010.

[32] E.P.C. Jones, L. Li, J.K. Schmidtke, e P.A.S. Ward. Practical Routing in Delay-Tolerant Networks. *IEEE Transactions on Mobile Computing*, 6(8):943–959, agosto de 2007.

[33] Maurice J. Khabbaz, Chadi M. Assi, e Wissam F. Fawaz. Disruption-Tolerant Networking: A Comprehensive Survey on Recent Developments and Persisting Challenges. *IEEE Communications Surveys & Tutorials*, 14(2):607–640, maio de 2012.

[34] MHR. Khouzani, Soheil Eshghi, Saswati Sarkar, Ness B. Shroff, e Santosh S. Venkatesh. Optimal energy-aware epidemic routing in DTNs. *ACM International Symposium on Mobile Ad Hoc Networking and Computing*, páginas 175–182. ACM, junho de 2012.

[35] Seungbeom Lee, Hanho Lee, Jongyoon Shin, e Je-Soo Ko. A High-Speed Pipelined Degree-Computationless Modified Euclidean Algorithm Architecture for Reed-Solomon Decoders. *IEEE International Symposium on Circuits and Systems, 2007*, páginas 901–904. IEEE Computer Society, maio de 2007.

[36] Yun Li, Zhun Wang, Xiaohu You, Qi-Lie Liu, e Weiyi Zhang. NER-DRP: Dissemination-based Routing Protocol with Network-layer Error Control for Intermittently Connected Mobile Networks. *Mobile Networks and Applications*, 17(5):618–628, outubro de 2012.

[37] Yong Liao, Kun Tan, Zhensheng Zhang, e Lixin Gao. Estimation based erasure-coding routing in delay tolerant networks. *International conference on Wireless communications and mobile computing*, páginas 557–562. ACM, setembro de 2006.

[38] Rudolf Lidl e Harald Niederreiter. *Finite Fields*. Encyclopedia of mathematics and its applications 20. Cambridge University Press, New York, USA, 2ª edição, 1997.

[39] Anders Lindgren, Avri Doria, e Olov Schelén. Probabilistic routing in intermittently connected networks. *SIGMOBILE Mobile Computing and Communications Review*, 7(3):19–20, julho de 2003.

[40] Cong Liu e Jie Wu. Scalable routing in delay tolerant networks. *ACM international symposium on Mobile ad hoc networking and computing*, páginas 51–60. ACM, setembro de 2007.

[41] Florence Jessie MacWilliams e Neil James Alexander Sloane. *The Theory of Error Correcting Codes*. North Holland Publishing Co., New York, NY, USA, 1ª edição, 1977.

[42] Preston Marshall. The disruption tolerant networking program, 2005. http://www.darpa.mil/sto/solicitations/DTN/briefs.htm.

[43] Muriel Médard e Alex Sprintson. *Network Coding: Fundamentals and Applications*. Academic Press, Oxford, 2012.

[44] Hugues Mercier, Vijay K. Bhargava, e Vahid Tarokh. A survey of error-correcting codes for channels with symbol synchronization errors. *IEEE Communications Surveys Tutorials*, 12(1):87–96, fevereiro de 2010.

[45] James S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software—Practice & Experience*, 27(9):995–1012, setembro de 1997.

[46] Irvin S. Reed e Gustave Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, julho de 1960.

[47] Fabrício Jorge Lopes Ribeiro, Aloysio de Castro Pinto Pedroza, e Luís Henrique Maciel Kosmalski Costa. Deepwater Monitoring System in Underwater Delay/Disruption Tolerant Network. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 10(1):1324–1331, janeiro de 2012.

[48] Gabriel Sandulescu e Simin Nadjm-Tehrani. Opportunistic DTN routing with window-aware adaptive replication. *Asian Conference on Internet Engineering*, páginas 103–112. ACM, novembro de 2008.

[49] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, julho de 1948.

[50] Bernard Sklar. *Digital Communications: Fundamentals and Applications*. Prentice Hall P T R, New Jersey, USA, 2ª edição, 2001.

[51] Thrasyvoulos Spyropoulos, K. Psounis, e C.S. Raghavendra. Single-copy routing in intermittently connected mobile networks. *IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, páginas 235–244. IEEE Computer Society, outubro de 2004.

[52] Thrasyvoulos Spyropoulos, Konstantinos Psounis, e Cauligi S. Raghavendra. Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks. *ACM SIGCOMM workshop on Delay-tolerant networking*, páginas 252–259. ACM, agosto de 2005.

[53] Amin Vahdat e David Becker. Epidemic Routing for Partially-Connected Ad Hoc Networks. Relatório Técnico CS-200006, abril de 2000.

[54] John Whitbeck, Vania Conan, e Marcelo Dias de Amorim. Performance of Opportunistic Epidemic Routing on Edge-Markovian Dynamic Graphs. *IEEE Transactions on Communication*, 59(5):1259–1263, maio de 2011.

[55] Jingfeng Xue, Jiansheng Li, Yuanda Cao, e Ji Fang. Advanced PROPHET Routing

in Delay Tolerant Network. *International Conference on Communication Software and Networks*, páginas 411–413. IEEE Computer Society, fevereiro de 2009.

Apêndices

APÊNDICE A

CAMPOS FINITOS

A teoria dos campos finitos ou Campos de Galois (GF, *Galois Fields)* [**?** **?**] postulada por Évariste Galois e Niels Henrik Abel no século XIX, permite calcular equações polinomiais complexas através de aritmética modular. GF é um campo formado por um número finito de elementos (α) e representado por $GF(\alpha)$. Operações aritméticas realizadas entre dois elementos do campo resultam em um terceiro elemento pertencente ao campo. As operações de soma e multiplicação, realizadas através de disjunção exclusiva, são associativas e cumulativas, possuindo a forma de um grupo abeliano [**?** **?** **?**], e as operações de subtração e divisão são equivalentes às operações de adição e multiplicação, respectivamente.

Os Campos de Galois podem ser estendidos para campos com α^n elementos, resultando em um campo estendido $GF(\alpha^n)$, onde α é um número primo e n é um número inteiro positivo. $GF(\alpha^n)$ é constituído pelos valores $(0, \alpha^0, \alpha^1, ..., \alpha^{m-1})$, sendo que $m = 2^n - 1$. Por conseguinte, pode-se dizer que um campo finito é formado por α^m elementos e a soma ou produto entre dois deles deve resultar em um outro elemento do campo, com grau igual ou menor que 2^{n-1}. Cada componente do campo pode ser representado pela forma polinomial da equação A.1.

$$\alpha_2 x^2 + \alpha_1 x^1 + \alpha_0 \tag{A.1}$$

A obtenção dos elementos pertencentes ao campo depende de uma classe especial de polinômios, denominada polinômios primitivos. Esses polinômios têm grau n, no qual são irredutíveis, ou seja, não é possível fatorar. As regras matemáticas que definem se um polinômio é primitivo [**?**] fogem do escopo deste trabalho, sendo satisfatório saber que um polinômio primitivo é aquele que gera a sequência completa do campo finito, motivo pelo qual também é chamado de polinômio gerador. É interessante saber ainda, que

existem polinômios pré-determinados de acordo com o valor de n. Uma tabela contendo alguns dos principais polinômios primitivos pode ser encontrada em [**?**].

O polinômio gerador por si só não fornece todos os elementos do campo, porém, é possível obtê-los através de aritmética efetuada sobre o polinômio primitivo. Por exemplo, considere o campo representado por $GF(2^3)$, onde $m = 3$ e cujo polinômio gerador é $\alpha^3 + \alpha + 1$. Sabe-se que este possui 8 elementos, sendo o primeiro nulo. Os elementos pertencentes ao campo são obtidos em função do polinômio gerador, ou seja, $f(\alpha) = \alpha^3 + \alpha + 1$. Para obtê-los, é necessário encontrar as raízes da função, que, para $GF(2^3)$, são três elementos do próprio campo. A equação A.2 demonstra o cálculo da raíz de α^1.

$$\begin{aligned} f(\alpha) &= 0 \\ \alpha^3 + \alpha + 1 &= 0 \\ \alpha^3 &= -1 - \alpha \end{aligned} \tag{A.2}$$

A obtenção dos elementos menores que α^3 ocorre de forma simples, conforme apresentado na equação A.3. O polinômio gerador define $\alpha^3 = -1 - \alpha$. Uma vez que, pela aritmética disjuntiva exclusiva (XOR), as operações de adição e subtração são equivalentes, pode-se dizer que $\alpha^3 = -1 - \alpha$ é o mesmo que $\alpha^3 = 1 + \alpha$. Doravante será utilizada a última representação. Os demais elementos do campo são obtidos através da multiplicação de α, substituindo α^3 por $1 + \alpha$ e adicionando os termos, como demonstrado na equação A.4.

$$\begin{aligned} \alpha^0 &= 0 \\ \alpha^1 &= \alpha \\ \alpha^2 &= \alpha^2 \end{aligned} \tag{A.3}$$

$$\alpha^4 = \alpha.\alpha^3 = \alpha.(1+\alpha) = \alpha + \alpha^2$$

$$\alpha^5 = \alpha.\alpha^4 = \alpha.(\alpha + \alpha 2) = \alpha^2 + \alpha^3 = \alpha 2 + \alpha + 1$$

$$\alpha^6 = \alpha + \alpha 5 = \alpha.(\alpha 2 + \alpha + 1) = \alpha 3 + \alpha 2 + \alpha = 1 + \alpha + \alpha^2 + \alpha = 1 + \alpha 2 \qquad \text{(A.4)}$$

A partir de A.3 e A.4, obtém-se a sequência completa dos elementos do campo, conforme a tabela A. Como uma das propriedades de um campo finito é que ele é fechado, os elementos maiores que α^6 podem ser obtidos voltando ao início da tabela, ou seja, o valor de α^7 é o mesmo que o valor de α^0.

Potência de α	Forma Polinomial	Forma binária	Forma Decimal
0	0	000	0
α^0	1	001	1
α^1	α	010	2
α^2	α^2	100	4
α^3	$\alpha + 1$	011	3
α^4	$\alpha^2 + \alpha$	110	6
α^5	$\alpha^2 + \alpha + 1$	111	7
α^6	$\alpha^2 + 1$	101	5

Tabela A.0.1: Elementos do campo denotado por $GF(2^3)$

APÊNDICE B

CÓDIGOS-FONTE

Este Apêndice apresenta os códigos-fonte utilizados para obtenção dos dados referentes ao desempenho do EMCOD. As simulações foram realizadas no GNU® Octave e são apresentadas em duas seções. A Seção B.1 apresenta o Código-fonte utilizado para simular o desempenho do EMCOD em um ambiente sem perda de dados. Na Seção B.2 é apresentado o código utilizado para simular o desempenho do EMCOD em um ambiente que sofre perdas de dados.

B.1 Desempenho em ambientes desconsiderando perdas

A complexidade dos códigos Reed-Solomon apresentada na Subseção 3.2.3 foi implementada afim de calcular o consumo de energia. Como esses processos são realizados em momentos e nós diferentes, foram criados dois códigos: o Código-fonte B.1 calcula a energia consumida na fase de codificação e o Código-fonte B.2 calcula a energia consumida na fase de decodificação.

As funções `complexidade_codificador_rs` e `complexidade_decodificador_rs` recebe mcomo parâmetros o tamanho do bloco codificado, a quantidade de dados a serem codificadas e o número de símbolos que compõe cada unidade de dados. O valores retornados são a energia consumida pelo processo de codificação e decodificação, respectivamente.

```
1 function CCIB = complexidade_codificador_rs(n, k,m)
2   t=(n-k)/2;
3   R=k/n;
4   GFadd=m*0.4;
5   GFmulaxi=0.4*m*(m-2)/2;
6   GFmul=0.4*(m^2+3*(m-1)^2/2);
7   GFinv=8*m;
8   GFreg=2*m;
```

```
    GFmem=10*m;
    CCIB = ((1/m) * (((2*t)/R)*(GFadd + GFmulaxi + GFreg)))*8;
endfunction
```

Código-fonte B.1: Complexidade do Codificador Reed-Solomon

```
function CCIB = complexidade_decodificador_rs(n, k,m)
    t=(n-k)/2;
    R=k/n;
    GFadd=m*0.4;
    GFmulaxi=0.4*m*(m-2)/2;
    GFmul=0.4*(m^2+3*(m-1)^2/2);
    GFinv=8*m;
    GFreg=2*m;
    GFmem=10*m;
    t2 = (1-R)/R;
    t1 = (1-R)/(2*R);
    n1=1/R;
    k1 = 1;
    m2 = 1/m;

    CCIB = ((1/m)*((((1-R)/R)*GFinv)+(((1-R)/(2*R))*(4*t+1)*GFmul)+(((1/R)
        *(4*t-1))* GFmulaxi)+(((1/R)*(4*t-1)+((1-R)/(2*R))*(4*t+1))*GFadd)
        +(((1/R)*(4*t-1)+((1-R)/(2*R))*(6*t+1))*GFreg)+(1*GFmem)))*8;
endfunction
```

Código-fonte B.2: Complexidade do Decodificador Reed-Solomon

Os Códigos-fonte B.3 a B.8 são partes de um único Código fonte e são apresentados separadamente a fim de facilitar a compreensão. O Código-fonte B.3 apresenta as variáveis utilizadas como valores de referência. Os valores das variáveis `qtd_fragmento` e `tamanho_unidadeCCD` foram ajustados para cada FEC Reed-Solomon utilizado nas simulações.

```
%Unidades de medida
unidade_medida = 1024 *1024;                    %tamanho de mensagens
medida_watt = 1000000000000;                    %watt-hour
```

```
4  medida_milliwatt = 1000000000;                %miliwatt-hour
5  medida_nanowatt = 1000;                       %nano-hour
6
7  variacao=5;
8  numero_saltos = 10;                           %numero de saltos
9  tamanho_mensagem = i*variacao*unidade_medida;  %tamanho da mensagem em MB
10 tamanho_msg = i*variacao;                     %tamanho da mensagem em B
11 qtd_fragmento = 223;                          %numero total de fragmentos
12
13 %tamanho de cada fragmento
14 tamanho_fragmento = ceil(tamanho_mensagem / qtd_fragmento);
15 qtd_bloco = tamanho_fragmento;                %numero total de blocos
16 tamanho_unidadeCCD = 255;                     %tamaho da unidadeCCD em B
17
18 % tamanho da mensagem codificada
19 tamanho_msg_codificada = qtd_bloco * tamanho_unidadeCCD;
20 cabecalho = 12 + 14;                          %tamanho do Cabecalho
```

Código-fonte B.3: Valores de referência utilizados na simulação

O Código-fonte B.4 apresenta a sequência de códigos utilizada para calcular a sobrecarga de dados referente à utilização do RS.

```
1  sobrecarga = tamanho_msg_codificada - tamanho_mensagem; %sobrecarga em B
2  sobrecarga_bruta = (tamanho_mensagem / qtd_fragmento * tamanho_unidadeCCD)
       - tamanho_mensagem;
3  sobrecarga_percent = sobrecarga_bruta * 100 / tamanho_mensagem;
4  sobrecarga_real_percent = sobrecarga * 100 / tamanho_mensagem;
5
6  %tamanho da mensagemCCD
7  mensagemCCD = tamanho_unidadeCCD * 7;
8  %quantidade de mensagens codificadas
9  qtde_mensagemCCD = ceil(tamanho_msg_codificada / mensagemCCD);
10 %quantidade de mensagens sem codificacacao
11 qtde_mensagem_semcodificacao = ceil(tamanho_mensagem / mensagemCCD);
12
```

```
%numero de mensagens de sobrecarga
sobrecarga_mensagens = qtde_mensagemCCD - qtde_mensagem_semcodificacao;
```

Código-fonte B.4: Cálculo da sobrecarga de dados

A capacidade de correção do EMCOD foi calculada pela sequência de códigos apresentada no Código-fonte B.5.

```
capacidade_correcao = qtde_mensagemCCD-(ceil(tamanho_mensagem/(223*8*16)));
mensagens_perdidas = ceil(tamanho_mensagem/(223*8*16));
```

Código-fonte B.5: Cálculo da capacidade de correção do EMCOD

Para calcular a energia consumida pelos processos de codificação e decodificação de dados foi utilizado o Código-fonte B.6.

```
consumo_codificacao_por_fragmento = complexidade_codificador_rs(
    tamanho_unidadeCCD, qtd_fragmento, 8);
consumo_total_codificacao = consumo_codificacao_por_fragmento * qtd_bloco;
consumo_decodificacao_por_fragmento = complexidade_decodificador_rs(
    tamanho_unidadeCCD, qtd_fragmento, 8);
consumo_total_decodificacao = consumo_decodificacao_por_fragmento *
    qtd_bloco;
consumo_total = consumo_total_codificacao+consumo_total_decodificacao; % pJ
consumo_pWh = consumo_total /3600; % pWh
```

Código-fonte B.6: Energia consumida pelos processos de codificação e decodificação

A energia consumida para transmitir as mensagens foram calculadas a partir do Código-fonte B.7.

```
energia_preparacao = 44.777;
energia_envio_byte = 161;
energia_recepcao_preparacao = 30.749;
energia_recepcao_byte = 111;

%energia para transmissao de uma unica mensagem (completa) - 1 salto
energia_transmissao_msg = (((mensagemCCD + cabecalho)* energia_envio_byte)
    + ((mensagemCCD  +cabecalho) * energia_recepcao_byte) +
    energia_preparacao + energia_recepcao_byte);
```

```
%transmissao da ultima mensagem (que pode ter menos bytes) - RS
ultima_msgCCD =  tamanho_msg_codificada   - ( mensagemCCD * (
    qtde_mensagemCCD-1));
energia_ultima_msgCCD = (((ultima_msgCCD + cabecalho) * energia_envio_byte)
     + ((ultima_msgCCD +cabecalho) * energia_recepcao_byte) +
    energia_preparacao + energia_recepcao_byte);

%transmissao da ultima mensagem - sem codificacao
ultima_msg =  tamanho_mensagem   - ( mensagemCCD * (
    qtde_mensagem_semcodificacao-1));
energia_ultima_msg = (((ultima_msg +cabecalho)* energia_envio_byte) + ((
    ultima_msg  +cabecalho)* energia_recepcao_byte) + energia_preparacao +
    energia_recepcao_byte);

%energia transmissao nak
energia_transmissao_nak = ((cabecalho * energia_envio_byte) + (cabecalho *
    energia_recepcao_byte) + energia_preparacao + energia_recepcao_byte);

%========== Energia gasta com RS ==========
energia_transmissao_comRS = (energia_transmissao_msg * (qtde_mensagemCCD-1)
    ) + energia_ultima_msgCCD;
energia_transmissao_comRS_Wh = energia_transmissao_comRS/medida_milliwatt;

%total de energia gasta no codificador (RS + transmissao) em pWh
energia_codificador_comRS = consumo_total_codificacao +
    energia_transmissao_comRS;
%total de energia gasta no codificador (RS + transmissao)
energia_codificador_comRS_Wh = energia_codificador_comRS/medida_milliwatt;

%total de energia gasta no codificador (RS + transmissao) em pWh
energia_decodificador_comRS = consumo_total_decodificacao +
    energia_transmissao_comRS;
%total de energia gasta no codificador (RS + transmissao)
energia_decodificador_comRS_Wh = energia_decodificador_comRS/
    medida_milliwatt;
```

```
32 %energia para enviar por 10 saltos
33 energia_gasta_comRS_10 = energia_transmissao_comRS * numero_saltos;
34 energia_gasta_comRS_Wh_10 = energia_gasta_comRS_10 / medida_milliwatt;
35
36 %========== Energia gasta sem codificacao ==================
37 energia_transmissao_semRS = (energia_transmissao_msg * (
       qtde_mensagem_semcodificacao-1)) + energia_ultima_msg;
38 energia_transmissao_semRS_Wh = energia_transmissao_semRS/medida_milliwatt;
39
40 energia_transmissao_semRS_10 = ((energia_transmissao_msg * (
       qtde_mensagem_semcodificacao-1)) + energia_ultima_msg) * numero_saltos;
        %energia gasta (transmissao de mensagens) em pWh
41 energia_transmissao_semRS_Wh_10 = energia_transmissao_semRS_10/
       medida_milliwatt;
42
43 %energia considerando mensagens perdidas - 1 salto
44 energia_perdidas =  energia_transmissao_semRS + (energia_transmissao_msg *
       round(mensagens_perdidas)) + (energia_transmissao_nak * round(
       mensagens_perdidas) );
45
46 %energia considerando mensagens perdidas - 10 saltos
47 energia_perdidas_10 =  (energia_transmissao_semRS + (
       energia_transmissao_msg * round(mensagens_perdidas)) + (
       energia_transmissao_nak * round(mensagens_perdidas))) * numero_saltos;
```

Código-fonte B.7: Energia consumida durante a transmissão das mensagens

A fim de calcular o tempo gasto para transmitir as mensagens, da origem até o destino, foi utilizado o Código-fonte B.8.

```
1 bytes_transmitidos_rs = ((mensagemCCD + cabecalho) * (qtde_mensagemCCD-1))
      + (ultima_msgCCD + cabecalho);
2 atraso = 1;
3
4 %Tempo para reenviar mensagens perdidas
5 perdidas  = (mensagemCCD + cabecalho) * round(mensagens_perdidas);
```

```
perdidas_nak = cabecalho * round(mensagens_perdidas) ;
bytes_transmitidos_semcod = (((mensagemCCD + cabecalho) * (
    qtde_mensagem_semcodificacao -1)) + (ultima_msg + cabecalho)) +
    perdidas + perdidas_nak;

%tempo para envio da mensagem considerando atraso de 60s entre contatos
tamanho_pacote = mensagemCCD + cabecalho; %tamanho do pacote em bytes
tempo_gasto_comRS = ((192+(0.727*tamanho_pacote))*(capacidade_correcao));
tempo_gasto_comRS_s = tempo_gasto_comRS / 1000000 ; %tempo em segundos
tempo_gasto_semRS = ((192+(0.727 * tamanho_pacote))*(
    qtde_mensagem_semcodificacao -1))+(192+(0.727*(ultima_msg+cabecalho)));
tempo_gasto_semRS_s = tempo_gasto_semRS / 1000000; %tempo em segundos

%Tempo para reenviar mensagens perdidas
tempo_reenviar =(192+(0.727*(mensagemCCD + cabecalho)))*round(
    mensagens_perdidas);
tempo_nak = (192+(0.727*cabecalho))* round(mensagens_perdidas);
tempo_reenviar_s = ((tempo_reenviar +tempo_nak)/1000000)+atraso;

%Tempo para enviar mensagem original + mensagens perdidas
tempo_reenviar_total = ((tempo_gasto_semRS + tempo_reenviar + tempo_nak) /
    1000000)+ atraso;

%Tempo considerando 10 saltos
tempo_gasto_comRS_10 = (((192 + (0.727 * tamanho_pacote))*(
    capacidade_correcao))+atraso)*numero_saltos;
tempo_gasto_comRS10_s = tempo_gasto_comRS_10/1000000 ;%tempo em segundos
tempo_reenviar_total_10 = ((tempo_gasto_semRS + tempo_reenviar + tempo_nak
    + (3*atraso)) / 1000000);

%Tempo recepcao do minimo de mensagens recebidas para recuperar mensagem
tempo_minimo_recepcao_rs = tempo_gasto_comRS - tempo_reenviar ;
tempo_minimo_recepcao_rs_s = tempo_minimo_recepcao_rs / 1000000;
```

Código-fonte B.8: Tempo gasto para transmissão das mensagens

B.2 Desempenho em ambientes com perda de dados

Para analisar o desempenho do EMCOD considerando perda de dados, tomou-se como base o envio de uma mensagem com 144MB e quantidades fixas de pacotes perdidos (não foram considerados valores percentuais). Os Códigos-fonte B.9 a B.13 são partes de um mesmo Código-fonte, porém estão segmentados para facilitar a compreensão.

O Código-fonte B.9 apresenta os valores utilizados como referência durante a simulação. Os valores das variáveis `qtd_fragmento` e `tamanho_unidadeCCD` foram ajustados para cada FEC utilizado na simulação.

```
1  tamanho_mensagem = 144*1024*1024;    %tamanho da mensagem em B
2  tamanho_msg =144;                    %tamanho da mensagem em MB
3  qtd_fragmento = 223;                 %numero total de fragmentos
4  tamanho_fragmento = ceil(tamanho_mensagem / qtd_fragmento);
5  qtd_bloco = tamanho_fragmento;       %numero total de blocos
6  tamanho_unidadeCCD = 255;            %tamaho em B
7  tamanho_msg_codificada = qtd_bloco * tamanho_unidadeCCD;
8  tamanho_quadro = 2176;               %tamanho do quadro 802.11
9  %quantidade de unidades agrupadas por mensagem
10 qtdunCCD = floor(tamanho_quadro / tamanho_unidadeCCD);
11 mensagemCCD = tamanho_unidadeCCD * qtdunCCD; %tamanho da mensagemCCD
12 tx_correcao_RS = (tamanho_unidadeCCD - qtd_fragmento) / 2;
13 cabecalho = 16+34;                   %16->Epidemic e 34->802.11
14 medida_watt = 1000000000000;         %watt-hour
15 medida_milliwatt = 1000000000;       %miliwatt-hour
16 medida_nanowatt = 1000;              %nanowatt-hour
17 medida_energia = medida_milliwatt;
18 saltos = 10;
19 atraso_contato = 1000000 * 60;       % 60s entre contatos
20 funcao_atraso = 1000000000;          %tempo em segundos * 1000
21 tx_perda = ceil((i*40.2357)^2)+1;
22 tx_perda2 = 72000 + i*1000;
```

Código-fonte B.9: Valores de referência utilizados na simulação

A capacidade de recuperação de mensagens é diretamente considerada como capa-

cidade de correção de erros na sequência de dados. Para calcular essa capacidade foi utilizado o Código-fonte B.10.

```
qtde_mensagemCCD = ceil(tamanho_msg_codificada/mensagemCCD);
capacidade_correcao = qtde_mensagemCCD-round((qtde_mensagemCCD/
    tamanho_unidadeCCD)*tx_correcao_RS);
```

Código-fonte B.10: Cálculo da capacidade de correção

Para calcular a sobrecarga de dados sofrida pelo EMCOD foi utilizado o Código-fonte B.11.

```
if tx_perda > capacidade_correcao
  sobrecarga_mensagens = qtde_mensagemCCD+(tx_perda-capacidade_correcao);
else
  sobrecarga_mensagens = qtde_mensagemCCD;
endif

total_msg = sobrecarga_mensagens;
tamanho_pacote = mensagemCCD + cabecalho;

if tx_perda2 > capacidade_correcao
  sobrecarga_mensagens2 = qtde_mensagemCCD+(tx_perda2-capacidade_correcao);
else
  sobrecarga_mensagens2 = qtde_mensagemCCD;
endif

total_msg = sobrecarga_mensagens;
total_msg2 = sobrecarga_mensagens2;
```

Código-fonte B.11: Sobrecarga de dados

O Código-fonte B.12 apresente a sequência de códigos utilizada para calcular o consumo de energia, considerando os processos de codificação, decodificação e transmissão de dados.

```
energia_preparacao = 44.777;
energia_envio_byte = 161;
```

```
energia_recepcao_preparacao = 30.749;
energia_recepcao_byte = 111;

energia_transmissao_msg = ((tamanho_pacote*energia_envio_byte)+(
    tamanho_pacote*energia_recepcao_byte)+energia_preparacao+
    energia_recepcao_byte);

consumo_codificacao_por_fragmento = complexidade_codificador_rs(
    tamanho_unidadeCCD, qtd_fragmento, 8);
consumo_total_codificacao = consumo_codificacao_por_fragmento * qtd_bloco*
    qtd_fragmento;
consumo_decodificacao_por_fragmento = complexidade_decodificador_rs(
    tamanho_unidadeCCD, qtd_fragmento, 8);
consumo_total_decodificacao = consumo_decodificacao_por_fragmento *
    qtd_bloco*qtd_fragmento;

energia_gasta = consumo_total_codificacao + consumo_total_decodificacao+ ((
    energia_transmissao_msg * total_msg) * saltos);
energia_gasta_Wh = energia_gasta / medida_energia;

energia_gasta2 = consumo_total_codificacao + consumo_total_decodificacao+
    ((energia_transmissao_msg * total_msg2) * saltos);
energia_gasta2_Wh = energia_gasta2 / medida_energia;
```

Código-fonte B.12: Energia consumida pelo EMCOD

Para calcular o tempo gasto para transmissão das mensagens foi utilizado o Código-fonte B.13.

```
tamanho_pacote = mensagemCCD + cabecalho; %tamanho do pacote em bytes
tempo = ((((192 + (0.727 * tamanho_pacote))*(total_msg))+atraso_contato)*
    saltos);

if tx_perda > capacidade_correcao
  tempo_nak = (((192+(0.727*cabecalho))*round(tx_perda-capacidade_correcao)
     +atraso_contato)*saltos);
```

```
6    tempo_retorno = ((((192+(0.727*tamanho_pacote))*round(tx_perda-
         capacidade_correcao))+atraso_contato)*saltos);
7  else
8    tempo_nak = 0;
9    tempo_retorno = 0;
10 endif
11 tempo_s = (tempo+tempo_nak+tempo_retorno);              %tempo em segundos
12
13 tempo2 = ((((192+(0.727*tamanho_pacote))*(total_msg2))+atraso_contato)*
       saltos);
14 if tx_perda2 > capacidade_correcao
15   tempo_nak2 = (((192+(0.727*cabecalho))*round(tx_perda2-
         capacidade_correcao)+atraso_contato)*saltos);
16 else
17   tempo_nak2 = 0;
18 endif
19 tempo2_s = (tempo2+tempo_nak2) ;              %tempo em segundos
```

Código-fonte B.13: Tempo para transmissão das mensagens