INTEGRAÇÃO MATLAB E NS-2 PARA UM SIMULADOR DE REDES PARA O PADRÃO DVB-S2

Ana Fernanda Quaresma Batista Santos

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Marcello Luiz Rodrigues de Campos

Rio de Janeiro

Dezembro de 2013

INTEGRAÇÃO MATLAB E NS-2 PARA UM SIMULADOR DE REDES PARA O PADRÃO DVB-S2

Ana Fernanda Quaresma Batista Santos

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Marcello Luiz Rodrigues de Campos, Ph.D.

Prof. Eduardo Antônio Barros da Silva, Ph.D.

Prof. Lisandro Lovisolo, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2013

Santos, Ana Fernanda Quaresma Batista

Integração Matlab e NS-2 para um Simulador de Redes para o Padrão DVB-S2/Ana Fernanda Quaresma Batista Santos. – Rio de Janeiro: UFRJ/COPPE, 2013.

XV, 110 p.: il.; 29,7cm.

Orientador: Marcello Luiz Rodrigues de Campos

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2013.

Referências Bibliográficas: p. 85 – 87.

1. DVB-S2. 2. Redes de Computadores. 3. Integração NS-2 e Matlab. I. Campos, Marcello Luiz Rodrigues de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

À minha família.

Ao meu querido amigo

Alberto W. Collavizza.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

INTEGRAÇÃO MATLAB E NS-2 PARA UM SIMULADOR DE REDES PARA O PADRÃO DVB-S2

Ana Fernanda Quaresma Batista Santos

Dezembro/2013

Orientador: Marcello Luiz Rodrigues de Campos

Programa: Engenharia Elétrica

O padrão DVB-S2, desenvolvido para transmissão de sinais de televisão digital via satélite, tem como uma das suas funcionalidades a distribuição de serviços de interatividade através da transmissão de pacotes de dados. Por oferecer seus serviços via satélite, este sistema de comunicação alcança clientes que estão em áreas isoladas e não conseguem acessar as redes de comunicação por meio de cabos ou fibras óticas. Um simulador do padrão DVB-S2 foi desenvolvido para testar a viabilidade de transmissão de pacotes de Internet através desse sistema e assim oferecer essa opção para clientes que estão em áreas onde não há outros meios de acesso a rede de dados. Para construir o simulador com transmissão de pacotes IP, foram utilizados dois *softwares*, o Matlab, uma ferramenta matemática com a qual foi implementada toda a parte de processamento de sinais relacionada ao DVB-S2, e o *Network Simulator 2* (NS-2), um simulador de redes de computadores que foi utilizado para implementar a parte lógica do projeto. A integração desses dois programas foi feita para que funcionem de forma transparente para o usuário. Além disso, o simulador foi construído como uma biblioteca dinâmica, dessa forma os usuários podem utilizar o simulador de DVB-S2 sem alterar a distribuição oficial do NS-2. Ao final do projeto, temos como resultado não apenas um simulador completo para transmissão de pacotes IP de acordo com padrão DVB-S2, mas temos também como produto uma ferramenta que permite criar simulações utilizando o Matlab e o NS-2, possibilitando implementar simuladores para testar o acesso à rede usando outros sistemas de comunicação.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

CROSS-LAYER SIMULATION OF A COMMUNICATION SYSTEM USING THE DVB-S2 STANDARD

Ana Fernanda Quaresma Batista Santos

December/2013

Advisor: Marcello Luiz Rodrigues de Campos

Department: Electrical Engineering

The standard DVB-S2, developed for digital television signal transmission via satellite, has as one of its features distribution of interactive services through packet data transmission. For offering its services via satellite, this communication system reaches clients that are in isolated areas and cannot access communication networks through cables or optical fiber. A DVB-S2 simulator was developed to test the viability of Internet packet transmission through this system. To construct the DVB-S2 simulator with IP packet transmission, we used two softwares: Matlab, a mathematical simulator tool in which was implemented all signal processing part related to DVB-S2, and Network Simulator 2 (NS-2), a computer network simulator that was used to implement the logical part of the project. The integration of these two programs was made such that they can function transparently to the user. Besides that, the simulator was constructed as a dynamic library, in that way the users can use the DVB-S2 simulator without making changes in NS-2 oficial distribution. The result achieved is not just a complete simulator for IP packet transmission using the DVB-S2 standard, but we also have procuced a tool that allows simulations using Matlab and NS-2, therefore testing physical, access and network layers in a single integrated tools.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

16APSK	*16 Amplitude and Phase Shift Keying*, p. 5
32APSK	*32 Amplitude and Phase Shift Keying*, p. 5
8PSK	*Eight Phase Shift Keying*, p. 5
ACM	*Adaptive Coding and Modulation*, p. 11
AWGN	*Additive White Gaussian Noise*, p. 55
BCH	*Bose-Chaudhury-Hocquenghem*, p. 5
BPSK	*Binary Phase Shift Keying*, p. 5
CBR	*Constant-Bit-Rate*, p. 12
CCM	*Constant Coding and Modulation*, p. 11
DFL	*Data Field Length*, p. 14
DTV	*Digital Television*, p. 3
DVB-ASI	*DVB Asyncrhonous Serial Interface*, p. 11
DVB-C	*Digital Video Broadcasting - Cable*, p. 4
DVB-H	*Digital Video Broadcasting - Handheld*, p. 5
DVB-RCS	*Digital Video Broadcasting - Return Channel via Satellite*, p. 46
DVB-S2	*Digital Video Broadcasting - Satellite - Second Generation*, p. 1
DVB-S	*Digital Video Broadcasting - Satellite*, p. 4
DVB-T	*Digital Video Broadcasting - Terrestrial*, p. 4

DVB	*Digital Video Broadcasting*, p. 1
FCC	*Federal Communications Commission*, p. 2
FEC	*Foward Error Correction*, p. 10
GS	*Generic Stream*, p. 11
HDTV	*High-Definition Television*, p. 3
IPTV	*Internet Protocol Television* ou Televisão via IP, p. 7
IP	*Internet Protocol*, p. 1
LDPC	*Low Density Parity Check code*, p. 5
LLR	*Log Likelihood Ratio*, p. 53
LL	*Link Layer*, p. 45
MAC	*Medium Access Control*, p. 45
MHP	*Multimedia Home Platform*, p. 5
MPEG	*Moving Pictures Experts Group*, p. 4
MSB	*Most Significant Bit*, p. 11
NHK	*Nippon Hoso Kyokai* - Corporação Japonesa de Radiodifusão, p. 3
NPD	*Null-Packet Deletion*, p. 12
NTSC	*National Television System Committee*, p. 2
OSI	*Open Systems Interconnection*, p. 42
PAL	*Phase Alternation Line*, p. 3
PLS	*Physical Layer Signalling*, p. 33
PRBS	*Pseudo Random Binary Sequence*, p. 18
QPSK	*Quadrature Phase Shift Keying*, p. 5
SECAM	*Sequential Couler Avec Memoire*, p. 3

TCP Transmission Control Protocol, p. 48

TS *Transport Stream*, p. 11

UDP User Datagram Protocol, p. 48

UPL *User Packet Length*, p. 11

UP *User Packets*, p. 11

USSR *Union of Soviet Socialist Republics*, p. 3

Capítulo 1

Introdução

A televisão é um aparelho eletrônico muito popular e um meio de comunicação importante e com grande penetração na sociedade. Desde que foi criada nos anos 20, a televisão mudou bastante e está deixando de ser apenas um aparelho no qual podemos ver imagens, há idéias como interatividade entre transmissoras e telespectador e transmissão de internet pelos canais de televisão.

Esta dissertação descreve um simulador do sistema de transmissão de televisão digital via satélite, o DVB-S2 , sobre o qual testamos a troca de pacotes IP .

Este primeiro capítulo apresenta um breve relato sobre a história do desenvolvimento da televisão, o surgimento da imagem digital, o projeto DVB e também novas pesquisas que estão sendo desenvolvidas sobre a transmissão de sinais de televisão. O segundo capítulo é dedicado ao sistema DVB-S2, o foco do projeto, e nele estão definidos os detalhes técnicos de cada módulo deste sistema. O terceiro capítulo traz informações sobre como foi feito o projeto, a integração das ferramentas utilizadas na implementação e detalhes sobre os softwares utilizados e a arquitetura do simulador descrito aqui. As simulações realizadas e os resultados obtidos estão expostos no quarto capítulo. No capítulo seguinte temos as conclusões e propostas de trabalhos futuros. Ainda neste texto, há anexos que possuem detalhes sobre como funciona e como foi construido o simulador.

1.1 A história da televisão

1.1.1 Desde a primeira TV até a imagem digital

A palavra "televisão" foi apresentada à sociedade pela primeira vez em 1900 na Exposição Mundial (*World's Fair*) em Paris, onde o primeiro Congresso Internacional de Eletricidade foi realizado. A palavra televisão surgiu de duas palavras de origens distintas, a palavra grega *tele* que quer dizer distante e a palavra latina *visio* que significa visão. O primeiro aparelho de televisão só foi apresentado ao público nos anos de 1920[2].

As primeiras televisões construídas eram monocromáticas e tinham uma tela pequena. Em 1932, o Reino Unido começou a transmissão de sinais de TV utilizando como base os experimentos de John Logie Baird que fez um aparelho que usava um sistema de 30 linhas. Outras definições de imagem foram testadas, mas em 1950 a maior parte dos países europeus adotou como padrão a imagem com varredura de 625 linhas e 50 campos (ou 25 quadros) por segundo[2].

Nos Estados Unidos, o primeiro sistema completo de transmissão começou a funcionar em 1934. Na década de 30, muitos sistemas mecânicos e eletrônicos foram criados e testados. A Comissão Federal de Comunicações (FCC) não estava satisfeita com a qualidade e desempenho do padrão de imagem adotado até então e as pesquisas na área não pararam. Em 1942, a FCC adotou como padrão o trabalho realizado pelo Comite Nacional de Sistema de Televisão (NTSC) que estabeleceu como padrão a imagem com varredura de 525 linhas com 60 campos (ou 30 quadros) por segundo [2].

O próximo acontecimento nessa evolução é o surgimento da TV colorida, que aconteceu nos anos 40 nos Estados Unidos. Nesta década vários protótipos de sistemas de televisão a cores foram apresentados e demonstrados, mas apenas no início dos anos 50 foi desenvolvido o primeiro sistema totalmente eletrônico e compatível com o sistema de televisão monocromática. Em 1953, esse sistema foi aprovado pela FCC e foi adotado como padrão de sistema de televisão colorida e no ano seguinte foi feita na California a primeira transmissão ao vivo e a cores[2]. Mas essas novas televisões não fizeram muito sucesso até a primeira década após seu lançamento, os primeiros poucos aparelhos vendidos eram muito caros e não funcionavam tão bem, além disso os programas a cores eram raros.

Na Europa, a adoção da televisão a cores demorou um pouco mais. Entre os anos de 1953 e de 1967 foram apresentadas várias alternativas de sistemas a cores compatíveis com o

sistema de televisão monocromática existente naquele continente. Na França e na USSR foi adotado como padrão o sistema SECAM (cuja tradução da abreviatura significa cor sequencial usando memória) que foi desenvolvido por Henri de France. O sistema desenvolvido por este cientista sugeria transmitir a informação sobre cor em duas subportadoras, responsáveis por transmitir os dados relativos à crominância, sequencialmente em linhas alternadas. A implementação da técnica do sistema SECAM incentivou o desenvolvimento do sistema de Linha de Fase Alternante, abreviado como sistema PAL. Este sistema, a cada linha, alterna a fase de uma das médias dos componentes do sinal de cor e as distorções levam ao valor correto da cor. O sistema PAL foi adotado por um grande número de países europeus e as primeiras trasmissões com ele aconteceram em 1967 na Alemanha e Grã-Bretanha[2].

O desenvolvimento da tecnologia não parou com o aparelho de televisão a cores, os pesquisadores ainda tinham como melhorar a tecnologia existente até o momento, por isso os estudos continuaram em diversas partes do planeta. O desafio seguinte era criar um sistema de televisão totalmente digital. Podemos citar alguns importantes grupos que se destacaram nos estudos para este novo tipo de sistema e forneceram boas experiências e resultados que ajudaram o desenvolvimento dos sistemas de televisão digital, ou apenas DTV como se costuma mencionar. Os principais projetos foram o HDTV no Japão, os projetos Eureka EU 95 e PALplus na Europa e o *Advanced Compatible Television* nos Estados Unidos[2].

A Corporação Japonesa de Radiodifusão, cuja sigla é NHK, começou um projeto de pesquisa sobre o futuro da televisão logo após o sucesso da transmissão ao vivo dos Jogos Olímpicos de Verão em Tokyo em outubro de 1964[3]. Essa transmissão foi feita utilizando técnicas de transmissão de vídeo via satélite desenvolvidas pela própria NHK. Cinco anos mais tarde, a corporação estabelecia o conceito de televisão de alta definição, o HDTV.[2]

Uma das principais idéias que os japoneses tinham em vista quando lançaram o HDTV era que este sistema de televisão com alta qualidade de imagem se tornasse um padrão único mundial. Além de uma imagem com definição suficiente para ver a uma distância menor dos que as existentes até então, o padrão teria que ser compatível com os padrões de televisão já existentes. Este fato teve importância na escolha do número de linhas do padrão, até o presente momento tínhamos o padrão americano com 525 linhas e o europeu com 625 linhas, o maior divisor comum destes dois números é 25. O número de linhas escolhido para o HDTV foi 1125, pois é múltiplo de 25[2].

Na Europa, os países estavam divididos entre dois sistemas analógicos de televisão

a cores, o PAL e o SECAM. O lançamento do padrão HDTV se mostrou como uma ótima oportunidade para definir um sistema padrão comum a toda a Europa e substituir os sistemas analógicos utilizados. Tendo o padrão japonês como base, dois grupos de pesquisas foram criados na Europa, o projeto Eureka EU 95 e o PALplus. Muitos estudos foram feitos e algumas tecnologias desenvolvidas chegaram a ser implementadas. O projeto Eureka EU 95 terminou em 1995, já o PALplus de certa forma deu origem ao projeto de Transmissão Digital de Vídeo, o projeto DVB. Alguns de seus principais membros fundaram o DVB e levaram com eles a experiência adquirida no PALplus para a criação deste novo projeto[2].

1.1.2 O projeto DVB

O projeto DVB é uma referência para o desenvolvimento de televisão digital em muitos países. Nos primeiros 12 anos do projeto, contados a partir de sua fundação em 1993[1], a organização já contava com 270 membros de 32 países [2].

No ínicio das atividades do projeto DVB na Europa, o *Moving Pictures Experts Group* (MPEG) estava trabalhando em um conjunto de especificações para codificação de sinais de vídeo e áudio, sendo que a parte de áudio já estava na parte final do processo de padronização. Como o MPEG estava em fase final de sua padronização, o projeto DVB decidiu usar os padrões produzidos por eles e assim conseguir uma ampla base internacional para o desenvolvimento da DTV. Este fato fez com que muitas organizações européias e o MPEG estabalecessem uma cooperação intensiva[2].

O primeiro resultado importante do projeto DVB aconteceu em novembro de 1992 com a divulgação do relatório "Perspectivas da Televisão Digital Terrestre". Neste relatório o grupo mostrava seus objetivos e como poderiam desenvolver o sistema DVB para a Europa. Além disso, o documento também mostrava uma forte tendência pela escolha da transmissão terrestre e apresentava a transmissão de televisão com alta definição de imagem como objetivo[2].

Apesar da preferência pela transmissão terrestre, a primeira especificação a ficar completa foi a do sistema por satelite (DVB-S[4]) em novembro de 1993[2]. Dois meses depois foi divulgada a especificação para DVB via cabo coxial (DVB-C[5]), seguida de diversas outras especificações, como a dos sistemas terrestre (DVB-T[6]) e para aparelhos móveis (DVB-H[7]

[1]No final de 1991, transmissoras, fabricantes de equipamentos eletrônicos e orgãos regulatórios formaram um grupo chamado *European Launching Group* (ELG). Em 1993, o grupo mudou seu nome para DVB.

de *handheld devices*).

No ínicio, o projeto DVB fazia pesquisas voltadas apenas para tecnologias de redes de transmissão com intuito de transportar sinais de áudio e vídeo digitais para receptores de TV, *set top-boxes* e receptores de áudio de alta fidelidade. Depois de alguns anos produzindo especificações sobre estes assuntos, o grupo começou a produzir especificações também sobre canais de interatividade e mais tarde para a transmissão de dados através do sistema DVB. Outra evolução deste sistema de transmissão de TV foi a criação da plataforma MHP (que significa *Multimedia Home Platform*), que integra a interatividade na televisão digital e cujo *software* pode rodar em diversos tipos de terminais[8].

1.1.3 Evolução do DVB-S para o DVB-S2

É natural que em uma especificação desenvolvida, depois de algum tempo, sejam acrescentadas modificações e uma nova configuração de forma a melhorar o desempenho do sistema. Com o tempo, a tecnologia evolui e muitas coisas que até então não se pensava ser possíveis podem ser conseguidas, como por exemplo uma taxa melhor de compressão ou um codificador de canal mais robusto.

Assim aconteceu com o sistema DVB-S, tendo sua especificação sido lançada em novembro de 1993, muitas tecnologias conhecidas atualmente não tinham sido criadas ainda. Conforme a evolução de pesquisas na área, decidiu-se melhorar o sistema DVB via satélite implementando algumas inovações e gerando assim uma nova especificação, o DVB-S2.

Comparando as duas especificações, podemos citar como exemplo de evolução a modulação empregada. No DVB-S são utilizadas as constelações QPSK e BPSK. Com a evolução dos transmissores, o grupo resolveu adotar uma modulação diferente para transmissão. O DVB-S2 utiliza, além do QPSK, as constelações de 8PSK, 16APSK e 32APSK[8]. Também notamos uma melhora na codificação de canal utilizado, o sistema antigo utilizava como corretor de erro o código convoluncional FEC(*Forward Error Correction*) concatenado com um código Reed-Solomon[9]. Em contra partida, o novo sistema usa dois codificadores, o LDPC, como código interno, concatenado com o código externo BCH[8]. Na época da criação do DVB-S, mesmo conhecendo o LDPC e sabendo que este gerava uma codificação mais robusta, sua decodificação ainda era muito complexa e por essa razão não o colocaram na especificação do DVB-S.

Sem dúvida, as modificações na modulação e nos codificadores de canal melhoraram

muito o desempenho na transmissão permitindo um aumento no tráfego de dados e reduzindo a perda dos mesmos. Mas outras mudanças foram feitas no sistema visando a melhora de seu desempenho como, por exemplo a utilização de símbolos piloto no DVB-S2. Outra diferença importante entre os dois sistemas que podemos citar é o formato dos sinais de entrada. O DVB-S aceita como entrada apenas pacotes MPEG-2. A versão mais nova do sistema, além de receber pacotes MPEG-2, recebe também outros tipos de dados, como por exemplo pacotes IP. O sistema passou a aceitar novos tipos de dados porque viu-se que se podia conectar o sistema de TV digital com outras redes de comunicação e daí veio a necessidade de aceitar entradas diferentes além dos pacotes MPEG. Na Seção 1.2, a seguir, podemos entender melhor por que o sistema foi modificado para aceitar outros tipos de dados além dos pacotes MPEG com conteúdos de vídeo e áudio[8].

1.2 Além da imagem: outros aspectos da transmissão de TV

Quando falamos em *Digital Video Broadcasting*, a primeira idéia que nos vem à cabeça é a de transmissão de vídeo e áudio de alta qualidade, e por muito tempo só se pensou nisso, as pesquisas na área estavam voltadas apenas para este intuito. Depois de muitos anos, viu-se que a TV digital poderia contribuir com muito mais do que apenas a transmissão de vídeo e áudio com baixa interferência e distorção, a DTV podia também fazer interface com outros sistemas de comunicação, como redes de computadores, permitir transmissão de dados diversos e oferecer serviços interativos de multimidia, como já citado na Seção 1.1.2.

É comum chamar o software de "programa" quando o contexto é computadores, porém essa palavra já é usada na área de televisão para referênciar o conteúdo disponibilizado para os telespectadores. Então, a solução foi usar uma nova palavra para o equivalente neste mundo e foi definida a palavra aplicação. O grupo DVB decidiu ter como meta uma solução técnica para que o terminal do usuário pudesse receber e apresentar aplicações em um ambiente que fosse independente de um equipamento específico de uma empresa, do criador da aplicação, ou mesmo do provedor de serviços de radio-transmissão. Assim sendo, nasceu a plataforma MHP que implementa essa idéia.

Existem alguns lugares onde as pessoas não conseguem receber o sinal do sistema DVB pelos meios clássicos de radio difusão e para resolver este problema a equipe do projeto

DVB começou a estudar a possibilidade de transmitir o conteúdo de TV via internet. O grupo desenvolveu uma arquitetura aberta IP, chamando-a de IPTV. Nisso houve alguns problemas, como por exemplo entregar o MPEG-2 (que é o formato utilizado pelo padrão DVB para vídeo e áudio) pela internet. Então tiveram que fazer algumas adaptações ao formato dos sinais para poder entregá-los sem problemas para o consumidor via as redes IP[8].

O foco principal de desenvolvimento do projeto DVB está concentrado em pesquisas voltadas ao MHP, porém existem alguns grupos que desenvolvem pesquisas sobre outras aplicações. "Formatos de Conteúdos Avançados de Áudio/Vídeo (para o transporte de conteúdo DVB em redes IP)", "Convergência de Transmissão e Serviços Móveis (definindo tecnologias necessárias para a construção de redes híbridas onde encontramos transmissão de TV e ramos cooperativos da rede de telefonia móvel)" e "Infraestrutura do Protocolo de Internet (para a transmissão eficiente de streams de dados DVB em redes baseadas em IP)" [2] são apenas alguns dos projetos desenvolvidos por grupos sob orientação do Módulo Técnico do projeto DVB.

Nesta dissertação não vamos utilizar tais tecnologias, pois o objetivo deste trabalho não é transmitir sinais de televisão pela internet, e sim transmitir internet através do sistema de televisão via satélite.

1.3 Objetivo da dissertação

Alguns lugares de difícil acesso precisam de enlaces gerados por satélite para se conectar a redes de comunicação, pois nesses lugares não há como instalar enlaces por cabos ou fibras óticas. Como vimos ao longo deste capítulo, o projeto DVB é de grande importância quando falamos de TV digital e é largamente difundido no planeta, este sistema é utilizado inclusive no Brasil.

Nesta dissertação pretende-se desenvolver um simulador que considere todos os parâmetros do padrão DVB-S2, para que possamos usá-lo para avaliar o seu desempenho na transmissão de pacotes IP. Para desenvolver este simulador, utilizamos dois *softwares* a fim de estudar e avaliar a transmissão de pacotes IP através do sistema DVB-S2 de forma completa. O Matlab foi utilizado para implementar a camada física da simulação e o Network Simulator 2 é utilizado para simular a camada lógica. Na construção do simulador foi feita a integração

desses dois *softwares* para que funcionem de forma transparente para o usuário, dessa maneira se pode fazer a simulação completa do sistema utilizando apenas um único programa, executando o simulador a partir do NS-2.

Além da utilização desses dois *softwares*, o simulador foi construído de forma que possamos adicionar o sistema DVB-S2 no NS-2 sem alterar sua distribuição oficial. Dessa maneira, os usuários do NS-2 podem usufruir do simulador desenvolvido neste projeto apenas instalando uma biblioteca em sua máquina e carregando ela no momento da simulação. Não é necessário fazer qualquer alteração na distribuição oficial do NS-2, nem compilar novamente o programa caso este já esteja funcionando no computador do usuário. Essa característica traz também a vantagem da utilização do simulador em redes de computadores. Se um usuário do NS-2, que utiliza este programa instalado em uma rede, quiser utilizar também o simulador do DVB-S2, basta instalar a biblioteca na sua máquina. Assim sendo, o usuário não depende do administrador da rede para complementar o NS-2 com este simulador.

No texto, encontra-se detalhes não apenas do funcionamento do simulador, mas também de como este projeto foi construído. Nos anexos, estão descritos os passos da construção da biblioteca que implementa o simulador. Seguindo as instruções que estão contidas neste documento, o leitor será capaz de reproduzir o simulador proposto aqui ou de criar seu próprio simulador baseado em um outro sistema de comunicação a sua escolha.

Retornando à questão da transmissão de pacotes IP utilizando o padrão DVB-S2, o simulador foi desenvolvido de forma a testar esse sistema de forma completa. Através dos resultados obtidos, verificaremos a viabilidade ou não do uso deste sistema para a transmissão de pacotes IP. Comprovando a eficácia deste sistema, ele poderá vir a ser mais uma opção futura de acesso a Internet para pessoas que vivem em áreas isoladas.

Capítulo 2

Arquitetura DVB-S2

2.1 Introdução

No capítulo anterior fomos apresentados à uma breve história da televisão e como surgiu a televisão digital e também o projeto DVB. Agora é o momento de conhecer melhor o funcionamento do DVB-S2 e este capítulo é totalmente dedicado a este sistema. Aqui temos a explicação dos detalhes técnicos de cada parte deste sistema e a importância de cada função que desempenha.

O sistema DVB-S2 pode ser dividido em seis subsistemas segundo as funções que executam e as informações que geram na saída. Na Seção 2.2 teremos uma visão geral do sistema DVB e quais são esses subsistemas. Nas seções subsequentes vamos descobrir como funciona cada um destes subsistemas, quais são seus módulos, a função de cada um deles e detalhes relevantes dos mesmos.

Antes, devemos ressaltar alguns detalhes importantes. As informações contidas neste capítulo foram retiradas em grande parte da especificação do sistema DVB-S2[1]. Assim sendo, para manter a coerência, foi utilizada a mesma nomeclatura usada nesse documento. As figuras que representam de forma gráfica os dados, possuem à sua esquerda os bits mais significativos que são os primeiros bits a serem enviados durante a transmissão.

2.2 Visão Geral

O sistema DVB-S2 pode ser dividido em subsistemas segundo as funções que executam, como podemos ver na Figura 2.1. O primeiro subsistema do DVB-S2 é chamado

de Modo de Adaptação. Ele recebe esse nome pois é responsável por fazer as adaptações necessárias no fluxo que entra no sistema para que este fique com um dos formatos exigidos pelo padrão. Em seguida temos a Adaptação de *Stream* (ou Fluxo). Como o DVB-S2 pode receber outros tipos de dados além dos fluxos de transporte MPEG-2, algumas vezes os dados recebidos não possuem o tamanho suficiente pedido pelo sistema. Nesses casos é necessário fazer mais adaptações no fluxo para que ele passe a ter um tamanho em bits que esteja de acordo com as especificações. Como o canal utilizado pelo satélite para transmissão possui muita interferência é necessária a utilização de uma boa codificação de canal e o subsistema de Codificação FEC (*Foward Error Correction*) desempenha esta tarefa. O próximo subsistema, chamado de Mapeamento, é responsável por mapear os bits em símbolos de uma das quatro constelações permitidas pelo sistema. Na sequência, temos o subsistema de Formação de Quadros da Camada Física, que, como o próprio nome já diz, constrói os quadros da camada física e tem algumas funções importantes como a inserção de símbolos piloto. O último subsistema é o de Modulação que modula a informação preparando-a para ser enviado pelo canal.

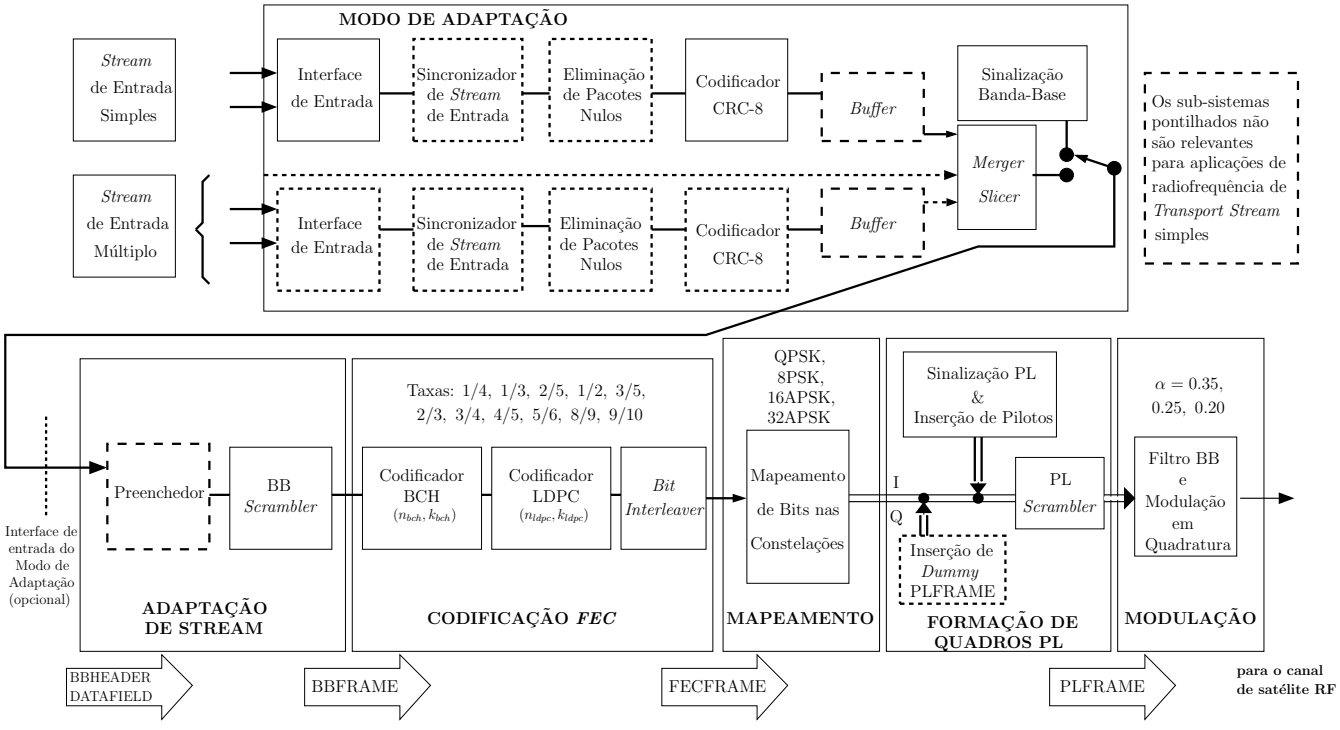


Figura 2.1: Arquitetura do Sistema DVB-S2.

2.3 Modo de Adaptação

O DVB-S2 recebe como entrada diversos tipos de fluxos e estes precisam sofrer algumas adaptações para que o sistema possa trabalhar com esses dados. Os módulos que fazem as adaptações compõe o Modo de Adaptação. Suas funções são fazer a interface do fluxo de entrada, a sincronização do fluxo de entrada (opcional), detecção de pacotes nulos (somente para Codificação e Modulação Adaptativa - ACM e para o formato de entrada *Transport Stream*), a união de fluxos de entrada (*Merger* - somente para o caso de múltiplos streams de entrada) e a divisão (*Slicer*) em DATA FIELDS, que é parte do "pacote" que este subsistema entrega. Para uma codificação e modulação constante (CCM), o Modo de Adaptação funciona como uma conversão transparente de DVB-ASI (DVB Asyncrhonous Serial Interface) para bits, adicionado de uma codificação CRC-8. Mas quando a codificação e a modulação são feitas de forma adaptativas (ACM), o Modo de Adaptação precisa ter as características descritas no apêndice D de [1].

O cabeçalho de banda-base (BBHEADER) que especifica o formato do fluxo de entrada e o tipo do Modo de Adaptação precisa ser anexado à frente do DATA FIELD, que é o quadro formado por este subsistema e que pode ser visto na Figura 2.3.

De acordo com a Figura 2.1, a sequência de entrada pode ser:

- ***Transport Stream* (TS)** simples ou múltiplo;
- ***Generic Stream* (GS - em pacotes ou contínuo)** simples ou múltiplo.

2.3.1 Interface de Entrada

A interface de entrada deve mapear a entrada que está num formato analógico para um formato lógico, ou seja, receber um sinal elétrico e transformá-lo em bits. O primeiro bit recebido é o mais significativo (MSB - *Most Significant Bit*).

O *Transport Stream* é formado por unidades chamadas de *User Packets* (UP) que possuem um comprimento constante UPL = 188×8 bits, que é o comprimento de um pacote MPEG, sendo que o primeiro byte é de sincronização (sync-byte = 47H). Este comprimento é representado por UPL pois recebe o nome de *User Packet Length*.

O *Generic Stream* pode ser caracterizado por um fluxo contínuo de bits ou fluxo com pacotes UP de comprimento constante igual a UPL, mas se o comprimento de pacote constante excede 64K bits, que é o valor máximo que UPL pode atingir, ele é tratado como

um fluxo contínuo e então teremos que UPL = 0_D. Outro caso em que o *Generic Stream* pode ser tratado como um fluxo contínuo é quando ele é composto por um fluxo de comprimento variável.

Para o *Generic Stream* em pacotes, se o primeiro byte da UP for de sincronização, o pacote deve permanecer inalterado, caso contrário, devemos inserir um sync-byte cujo valor é 0_D antes do pacote e somar 8 ao valor do UPL, pois iremos acrescentar 1 byte, ou seja, 8 bits, ao comprimento do pacote.

2.3.2 Sincronizador de Stream de Entrada

O processamento executado pelo módulo anterior às vezes pode sofrer algumas variações provocando atrasos na transmissão da informação do usuário, principalmente quando estamos trabalhando com modulação e codificação adaptativa (ACM). O Sincronizador de Stream de Entrada trabalha de modo a garantir uma taxa constante de bit (CBR - *Constant-Bit-Rate*) e um atraso de transmissão fim-a-fim constante para fluxos de entrada em pacotes. A especificação para este processo está no anexo D de [1]. O uso deste módulo é opcional, em alguns casos o uso é normativo, o anexo mencionado mostra para quais casos é possível e necessário utilizar o Sincronizador.

2.3.3 Eliminação de Pacotes Nulos

Quando o sistema está recebendo como entrada *Transport Stream*, que são pacotes MPEG, alguns destes pacotes podem ser nulos, ou seja, eles não conter informação útil. Para que estes pacotes não sobrecarreguem o sistema, que terá que fazer processamentos com pacotes cujos dados não têm nenhum valor, estes devem ser excluídos.

A exclusão de pacotes nulos também permite a redução da taxa de informação e aumenta a proteção contra erros no modulador. Os pacotes nulos removidos precisam ser reinseridos quando a informação passar pelo receptor. Para que isso ocorra de maneira correta, o cabeçalho que precede o quadro formado pelo Modo de Adaptação, como veremos mais adiante na seção 2.3.6, possui um campo que indica se um pacote nulo foi retirado. O nome deste campo é NPD que significa *Null-Packet Deletion*.

2.3.4 Codificador CRC-8

Como o canal utilizado pelo DVB-S2 é sujeito a muitas interferências, é necessário que os dados transmitidos sejam muito bem protegidos e para isso usamos alguns codificadores de canal. O primeiro codificador que pode ser usado é o CRC, para o caso em que os fluxos recebidos pelo sistema estão em pacotes, ou seja, quando UPL $\neq 0_D$, aplicamos o código corretor de erros, caso contrário não o fazemos.

O codificador sistemático CRC de 8 bits somente processa a parte útil da UP, o que significa que não processa o sync-byte, e para isso utiliza o polinômio gerador

$$g(x) = (x^5 + x^4 + x^3 + x^2 + 1)(x^2 + x + 1)(x + 1) = x^8 + x^7 + x^6 + x^4 + x^2 + 1 \qquad (2.1)$$

Os bits de paridade calculados com o código CRC devem substituir o próximo sync-byte, como está sendo mostrado na primeira gravura da Figura 2.2, e este sync-byte retirado será copiado num campo específico do BBHEADER, que é o cabeçalho do quadro formado pelo Modo Adaptativo como já mencionamos anteriormente.

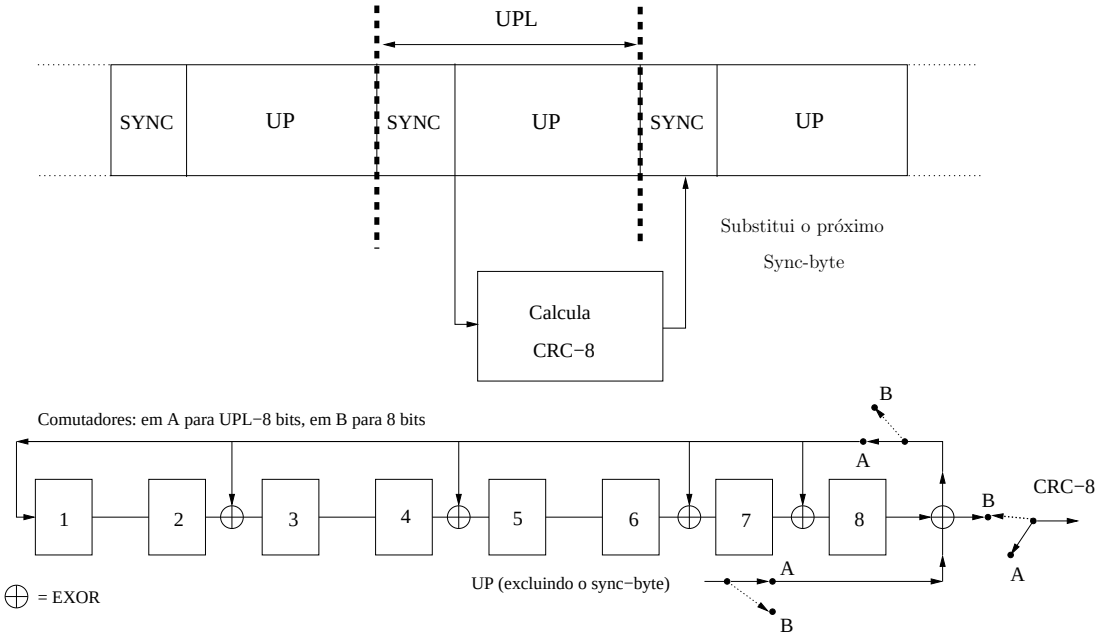


Figura 2.2: Implementação do codificador CRC 8 bits.

Por questão de simplicidade, na implementação do projeto, os bits de paridade são inseridos no mesmo pacote para o qual eles foram calculados. Essa mudança não causará nenhum impacto relevante para as simulações propostas. Também não há substituição de sync-byte, pois os pacotes que serão transmitidos na simulação não possuem esse byte de

sincronização, pois que consideramos a transmissão de pacotes UDP e estes não possuem sync-byte.

2.3.5 *Merger / Slicer*

O sinal na entrada do sistema pode admitir uma grande gama de tamanhos. Vimos que pode tanto ser um *Transport Stream* com unidades de tamanhos fixos ou um *Generic Stream* em pacotes de tamanhos fixos e variáveis ou mesmo contínuo. Como lidar com todos os diferentes tipos de comprimento? O sistema não leva muito em consideração o tamanho destes sinais que recebe, ele segue tamanhos específicos segundo as características de transmissão e assim gera pacotes com esses tamanhos, sobre os quais irá fazer outros processamentos.

O módulo que cria os pacotes com tamanhos específicos recebe como entrada *Generic Stream* contínuo ou em pacotes. Enquanto o módulo está processando as informações recebidas, as próximas sequências devem ser armazenadas num *buffer* e, depois que o módulo acabar de ler a sequência, ele poderá receber aquela que está no *buffer*.

O *Slicer* lê a entrada, ou seja, "fatia" ela em segmentos chamados de DATA FIELDS compostos com DFL(*Data Field Length*) bits, como mostra a Figura 2.3. Mesmo que receba múltiplas entradas, o *Slicer* as lê separadamente. O valor DFL pode variar dentro do seguinte intervalo:

$$K_{bch} - (10 \times 8) \geqslant DFL \geqslant 0 \tag{2.2}$$

no qual os possíveis valores de K_{bch} podem ser encontrados nas Tabelas 2.2 e 2.3 e os (10×8) bits são referentes ao tamanho do BBHEADER, como veremos na próxima seção.

O *Merger* deve concatenar, em uma saída, diferentes DATA FIELDS lidos e fatiados de uma entrada. Se for aplicado apenas um stream na entrada, então só é preciso usar o *Slicer*.

2.3.6 Inserção do Cabeçalho de Banda-Base

Os módulos apresentados até este momento fizeram modificações no fluxo de entrada levando a um pacote que nomeamos de DATA FIELD. Para indicar essas modificações e o conteúdo deste novo pacote precisa-se inserir um cabeçalho, como já mencionamos anterior-

mente, e nesse coloca-se informações importantes sobre o DATA FIELD que será entregue pelo Modo de Adaptação. O cabeçalho, conhecido como BBHEADER, possui um tamanho fixo de 10 bytes. Na Figura 2.3, podemos ver os campos do BBHEADER que possuem as informações sobre o DATA FIELD e que serão detalhadas a seguir.

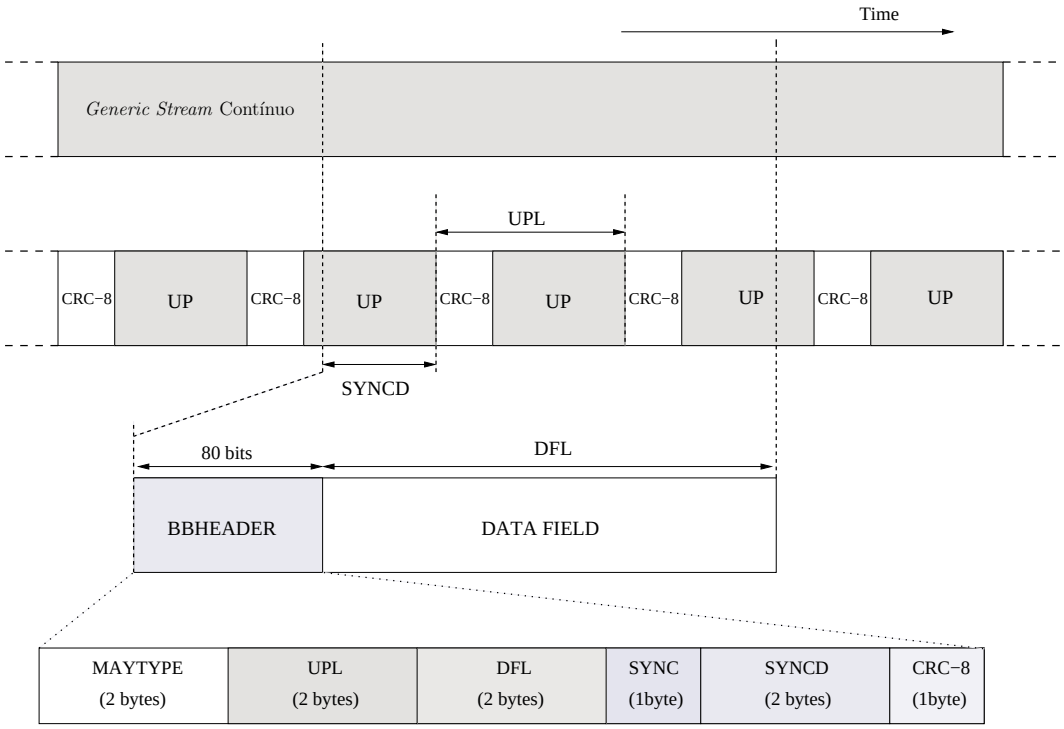


Figura 2.3: Formato do stream de saída do Modo de Adaptação e BBHEADER.

O campo MATYPE (2 bytes) descreve o formato do fluxo, o tipo de Modo de Adaptação e outras características. O primeiro byte é separado da seguinte maneira:

- **TS/GS** (2 bits): indica se a entrada é um *Transport Stream* ou um *Generic Stream* (contínuo ou em pacotes);
- **SIS/MIS** (1 bit): indica se a entrada é um stream simples ou múltiplos streams;
- **CCM/ACM** (1bit): indica se a codificação e a modulação são constantes (CCM) ou adaptativas (ACM). É preciso ressaltar que se a codificação e modulação forem variáveis (VCM), essa será tratada como ACM;
- **ISSYI** (Input Stream Synchronization Indicator) (1 bit): quando este campo está com o valor igual a 1, significa que está ativadoindicando que existe um sync-byte no pacote;

- **NPD** (Null-Packet Deletion) (1 bit): indica se a exclusão de pacotes nulos está ativa ou não;
- **RO** (2 bits): Fator Roll-Off de transmissão (α).

Na Tabela 2.1 estão os valores que cada um dos campos que compõe o primeiro byte de MATYPE pode assumir.

TS/GS	**SIS/MIS**	**CCM/ACM**	**ISSYI**	**NPD**	**RO**
11 = Transport	1 = simples	1 = CCM	1 = ativo	1 = ativo	00 = 0,35
00 = Generic em pacotes	0 = múltiplos	0 = ACM	0 = não ativo	0 = não ativo	01 = 0,25
01 = Generic contínuo					10 = 0,20
10 = reservado					11 = reservado

Tabela 2.1: Valores dos campos de primeiro byte de MATYPE [1].

O segundo byte do MATYPE só é utilizado quando o campo **SIS/MIS** indica se está trabalhando com múltiplos streams de entrada, então esse byte é usado como Identificador de Stream de Entrada (**ISI** - *Input Stream Identifier*). Caso contrário, esse byte é reservado.

O terceiro e quarto bytes do cabeçalho correspondem ao **UPL** (*User Packet Length*) que indica o tamanho da unidade UP que está sendo usada. Quando a UPL tem valor igual a zero, isso significa que o stream é contínuo, e quando assume valores entre 1 a 65535, a UP tem o valor indicado por este campo. Devemos lembrar que quando o tamanho é igual a 188×8 bits trata-se de um fluxode transporte MPEG-2.

Os próximos dois bytes da sequência formam o campo **DFL** (*Data Field Length*) que indica o comprimento do DATA FIELD e pode variar seu valor de 0 a 58112. Relembrando, DATA FIELD é a saída do bloco *Merger / Slicer* e pode ser composto por mais de uma UP.

O próximo campo do BBHEADER é o **SYNC** e ocupa um byte. Quando trabalhamos com *Transport* ou *Generic Stream* em pacotes, este campo contém uma cópia do byte de sincronização da UP (*User Packet Sync byte*). Mas quando trabalhamos com *Generic Stream* contínuo, não utilizamos sync byte, ou melhor, o valor deste é zero, então esse byte do campo SYNC é reservado para sinalização dos protocolos da camada de transporte ou para uso privado.

Em seguida temos o campo **SYNCD** ocupando dois bytes e serve para indicar o começo da primeira UP dentro do DATA FIELD. Quando utilizamos *Transport* ou *Generic Stream* em pacotes, a distância em bits entre o primeiro bit do DATA FIELD e o primeiro

bit da UP, que na verdade é o primeiro bit do CRC-8, é colocado no campo SYNCD. Se encontrarmos um valor nulo, o primeiro bit do DATA FIELD também é o primeiro bit da UP, porém se este campo possuir o valor 65535, significa que dentro do DATA FIELD não há nenhum começo de UP. A Figura 2.3 mostra com um exemplo esta distância SYNCD em um DATA FIELD. Se trabalharmos com *Generic Stream* contínuo, o campo SYNCD é reservado para uso futuro.

Por fim, o último byte pertence ao campo **CRC-8** que é o resultado do código corretor de erro que é aplicado aos 9 bytes do cabeçalho BBHEADER. O circuito de codificação usado é o que está apresentado pela Figura 2.2, cuja representação matemática é dada pela Equação 2.1. O mesmo código também é utilizado para o cálculo do CRC para a UP.

2.4 Adaptação de stream

Apesar das adaptações feitas pelo subsistema anterior, algumas vezes o pacote não fica com um dos tamanhos padrões exigidos pelo sistema, quando isso acontece é necessário fazer mais uma modificação, que é preencher o quadro. Então, a idéia principal deste subsistema é preencher o stream recebido com zeros para formar um quadro de banda-base, chamado de BBFRAME, que deve possuir um comprimento constante predefinido em [1] segundo alguns parâmetros. Esse comprimento constante é representado por K_{bch} e possui um certo valor de acordo com a taxa utilizada para correção de erro (taxa FEC). Esse preenchimento do quadro pode ocorrer, por exemplo, quando a informação produzida pelo usuário não for suficiente para completar um BBFRAME. Além do preenchimento do quadro de banda-base, esse subsistema é responsável também por embaralhar os bits do quadro. Esse embaralhamento tem como finalidade ajudar a correção de erros, como veremos na seção 2.4.2.

A Figura 2.4 apresenta um BBFRAME com preenchimento, mostrando como fica o quadro na saída do subsistema de Adaptação de Stream se o módulo de preenchimento for usado.

2.4.1 Preenchimento

O subsistema de codificação precisa receber um BBFRAME com comprimento fixo de K_{bch} bits segundo a taxa FEC utilizada pelo sistema, então o subsistema de adaptação de stream preenche o fluxo recebido, completando-o com $(K_{bch} - DFL - 80)$ bits, onde

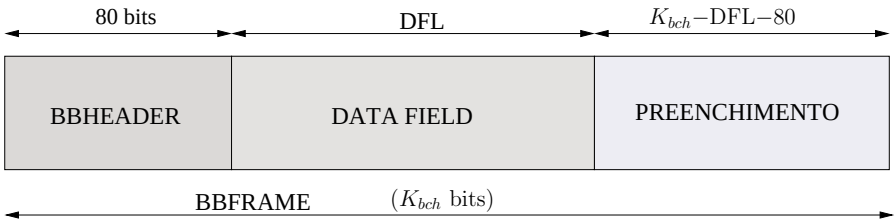


Figura 2.4: A saída do módulo de adaptação de stream: BBFRAME.

DFL é o comprimento do DATA FIELD e 80 bits são oriundos do BBHEADER. Na Figura 2.4 podemos ver estes comprimentos indicados no stream. Para aplicações de serviços de radiodifusão, temos $DFL = K_{bch} - 80$, então não é necessário preenchimento.

2.4.2 Scrambling

Nesta parte do sistema é feito um embaralhamento (em inglês chamado de *scrambling*) dos bits com o objetivo de fazer o espalhamento espectral do sinal. Esse embaralhamento deve ser feito de forma sincronizada, começando do bit mais significativo e terminando depois de K_{bch} bits.

Para embaralhar os bits, utiliza-se um polinômio conhecido por gerador de Sequência Binária Pseudo Randômica (PRBS - *Pseudo Random Binary Sequence*) que é descrito matematicamente por

$$1 + X^{14} + X^{15} \tag{2.3}$$

A Figura 2.5 representa o polinômio gerador PRBS indicado pela equação 2.3 com a sequência [1 0 0 1 0 1 0 1 0 0 0 0 0 0 0] que deve ser utilizada para iniciar o circuito. Cada vez que se começa a aplicar o *scrambling* em um novo BBFRAME, é necessário inicializar o circuito novamente com esta sequência. Para embaralhar os bits do BBFRAME, faz-se uma soma lógica de cada bit do quadro com os gerados pelo polinômio PRSB a partir da sequência de inicialização.

2.5 Codificação de Correção Antecipada de Erro

Ao transmitir o sinal através do ar, o canal usado para transmissão via satélite, o sinal pode sofrer muitas modificações causadas por interferências existentes nesse meio. Nesta

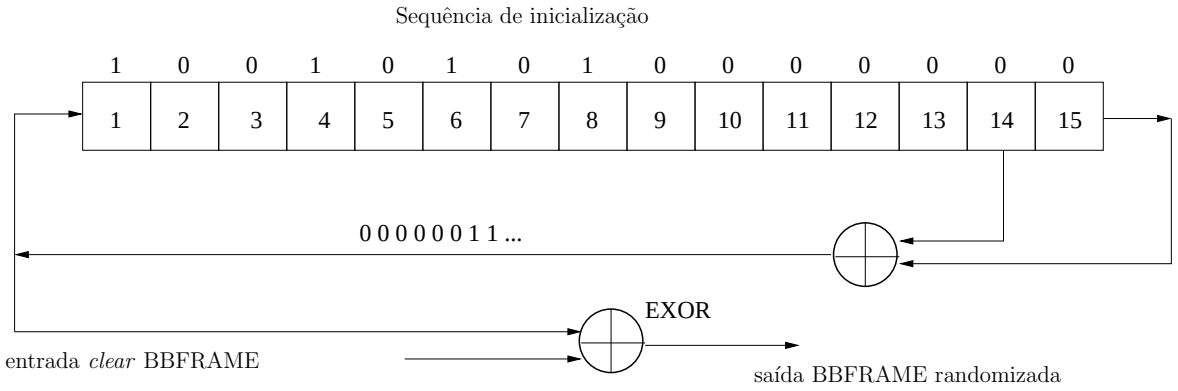


Figura 2.5: Possível implementação do polinômio PRBS.

mesma seção já apresentamos o codificador CRC-8 utilizado pelo Modo de Adaptação que ajuda a proteger a informação transmitida, agora vamos apresentar mais dois codificadores que compõem o subsistema de Codificação de Correção Antecipada de Erro, ou codificação FEC (por causa do nome em inglês *Forward Error Correction Encoding*). Os dois códigos utilizados por este subsistema são o código externo BCH (que leva o nome de seus inventores Bose, Ray-Chaudhuri e Hocquenghem) e o código interno LDPC (*Low Density Parity Check*). Além da codificação, o subsistema também realiza um *interleaving* que ajuda a separar erros que o sinal possa sofrer ao ser transmitido pelo canal.

Este subsistema recebe como entrada quadros compostos por BBFRAMEs e na saída entrega quadros formados por FECFRAMEs. Como podemos observar na Figura 2.6, o FECFRAME é composto por um BBFRAME seguido por bits de paridade do código externo BCH e do código interno LDPC, necessariamente nesta ordem.

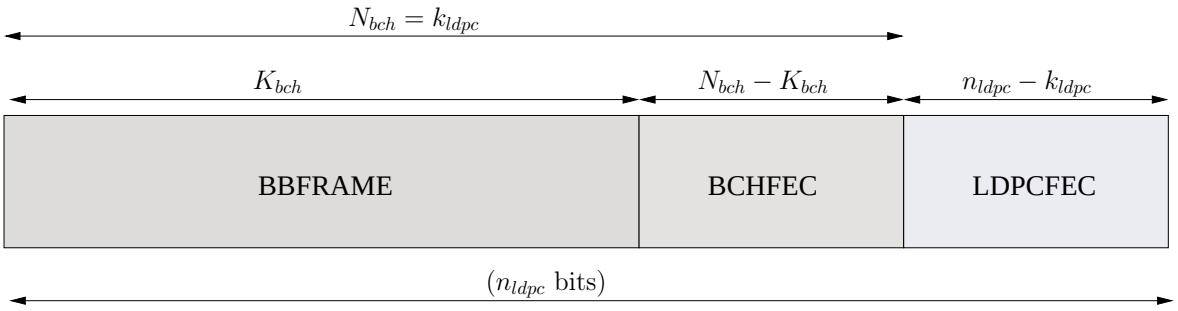


Figura 2.6: Saída do módulo de codificação com LDPC.

2.5.1 Código Externo (BCH)

Um código BCH(N_{bch}, K_{bch}) que pode corrigir até t erros deve ser aplicado sobre cada BBFRAME(K_{bch}) para gerar bits de cheque de paridade. Os paramêtros utilizados pelo código BCH, como por exemplo a quatidade t de erros que o código é capaz de corrigir e o tamanho do bloco de cheque de paridade, para $n_{ldpc} = 64800$, que é o tamanho de um quadro normal, estão na Tabela 2.2 e para $n_{ldpc} = 16200$, ou seja, para quadros curtos, estão na Tabela 2.3.

Código LDPC	**Bloco Não-codificado BCH (K_{bch})**	**Blc. Cod. BCH (N_{bch}) Blc. Não-Cod. LDPC (k_{ldpc})**	**t erros BCH**	**Blc. Cod. LDPC n_{ldpc}**
1/4	16 008	16 200	12	64 800
1/3	21 408	21 600	12	64 800
2/5	25 728	25 920	12	64 800
1/2	32 208	32 400	12	64 800
3/5	38 688	38 880	12	64 800
2/3	43 040	43 200	10	64 800
3/4	48 408	48 600	12	64 800
4/5	51 648	51 840	12	64 800
5/6	53 840	54 000	10	64 800
8/9	57 472	57 600	8	64 800
9/10	58 192	58 320	8	64 800

Tabela 2.2: Parâmetros de codificação para um FECFRAME normal contendo $n_{ldpc} = 64800$ bits.

Vamos considerar que no BBFRAME temos a mensagem

$$\mathbf{m} = (m_{k_{bch}-1}, m_{k_{bch}-2}, \ldots, m_1, m_0)$$

e queremos codificá-la na palavra código

$$\mathbf{c} = (m_{k_{bch}-1}, m_{k_{bch}-2}, \ldots, m_1, m_0, d_{n_{bch}-k_{bch}-1}, d_{n_{bch}-k_{bch}-2}, \ldots, d_1, d_0)$$

Para conseguir isto, segundo [1], basta seguir os seguintes passos:

- Multiplicar a mensagem na forma polinomial
  $m(x) = m_{k_{bch}-1}x^{k_{bch}-1} + m_{k_{bch}-2}x^{k_{bch}-2} + \ldots + m_1x + m_0 \quad \text{por} \quad x^{n_{bch}-k_{bch}}$
- Dividir $x^{n_{bch}-k_{bch}}m(x)$ pelo polinômio gerador $g(x)$. Tem-se como denominador

Código LDPC	Bloco Não-cod. BCH (K_{bch})	Blc. Cod. BCH (N_{bch}) Blc. Não-Cod. LDPC (k_{ldpc})	Correção t erros BCH	Taxa Efetiva LDPC k_{ldpc} 16200	Blc. Cod. LDPC n_{ldpc}
1/4	3 072	3 240	12	1/5	16 200
1/3	5 232	5 400	12	1/3	16 200
2/5	6 312	6 480	12	2/5	16 200
1/2	7 032	7 200	12	4/9	16 200
3/5	9 552	9 720	12	3/5	16 200
2/3	10 632	10 800	12	2/3	16 200
3/4	11 712	11 880	12	11/15	16 200
4/5	12 432	12 600	12	7/9	16 200
5/6	13 152	13 320	12	37/45	16 200
8/9	14 232	14 400	12	8/9	16 200
9/10	NA	NA	NA	NA	NA

Tabela 2.3: Parâmetros de codificação para um FECFRAME curto contendo $n_{\text{ldpc}} = 16200$ bits.

$$d(x) = d_{n_{bch}-kbch-1}x^{n_{bch}-k_{bch}-1} + d_{n_{bch}-kbch-2}x^{n_{bch}-k_{bch}-2} + \ldots + d_1x + d_0$$

- Constroi-se a palavra código na forma polinomial $c(x) = x^{n_{bch}-k_{bch}}m(x) + d(x)$.

Para obtermos o polinômio gerador $g(x)$ precisamos de informações das Tabelas 2.2 e segundo o tipo de quadro com o qual estamos trabalhando. Se estivermos trabalhando com um quadro normal de $n_{\text{ldpc}} = 64800$, da Tabela 2.2, 2.4, 2.3 e 2.5 conseguimos o número t de erros corrigidos pelo código BCH segundo a codificação que queremos utilizar e da Tabela 2.4 precisamos multiplicar os primeiros t polinômios para obter o polinomio gerador necessário para a codificação. Caso utilizemos um quadro curto com $n_{\text{ldpc}} = 16200$, precisaremos fazer exatamente o mesmo procedimento, mas iremos utilizar as Tabelas 2.3 e 2.5.

2.5.2 Código Interno (LDPC)

O codificador LDPC codifica sistematicamente uma informação com k_{ldpc} bits, que representamos por $\mathbf{i} = (i_0, i_1, \ldots, i_{k_{\text{ldpc}}-1})$, em uma palavra código com n_{ldpc} bits, representado por $\mathbf{c} = (i_0, i_1, \ldots, i_{k_{\text{ldpc}}-1}, p_0, p_1, \ldots, p_{n_{\text{ldpc}}-k_{\text{ldpc}}-1})$, sendo os primeiros bits originados da informação que foi codificada e os bits representados por p_n, para $n = 0, \ldots, n_{\text{ldpc}} - k_{\text{ldpc}} - 1$, os bits de cheque de paridade. Os paramêtros do codificador LDPC(n_{ldpc}, k_{ldpc}) podem ser encontrados nas Tabelas 2.2 e 2.3.

O processo de codificação é responsável por gerar os bits de paridade $(p_0, p_1, \ldots, p_{n_{\text{ldpc}}-k_{\text{ldpc}}-1})$ para cada bloco de k_{ldpc} bits de informação. A realização desta

$g_1(x)$	$1 + x^2 + x^3 + x^5 + x^{16}$
$g_2(x)$	$1 + x + x^4 + x^5 + x^6 + x^8 + x^{16}$
$g_3(x)$	$1 + x^2 + x^3 + x^4 + x^5 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{16}$
$g_4(x)$	$1 + x^2 + x^4 + x^6 + x^9 + x^{11} + x^{12} + x^{14} + x^{16}$
$g_5(x)$	$1 + x + x^2 + x^3 + x^5 + x^8 + x^9 + x^{10} + x^{11} + x^{12} + x^{16}$
$g_6(x)$	$1 + x^2 + x^4 + x^5 + x^7 + x^8 + x^9 + x^{10} + x^{12} + x^{13} + x^{14} + x^{15} + x^{16}$
$g_7(x)$	$1 + x^2 + x^5 + x^6 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{15} + x^{16}$
$g_8(x)$	$1 + x + x^2 + x^5 + x^6 + x^8 + x^9 + x^{12} + x^{13} + x^{14} + x^{16}$
$g_9(x)$	$1 + x^5 + x^7 + x^9 + x^{10} + x^{11} + x^{16}$
$g_{10}(x)$	$1 + x + x^2 + x^5 + x^7 + x^8 + x^{10} + x^{12} + x^{13} + x^{14} + x^{16}$
$g_{11}(x)$	$1 + x^2 + x^3 + x^5 + x^9 + x^{11} + x^{12} + x^{13} + x^{16}$
$g_{12}(x)$	$1 + x + x^5 + x^6 + x^7 + x^9 + x^{11} + x^{12} + x^{16}$

Tabela 2.4: Polinômios geradores do código BCH para um FECFRAME normal contendo $n_{\text{ldpc}} = 64800$ bits.

$g_1(x)$	$1 + x + x^3 + x^5 + x^{14}$
$g_2(x)$	$1 + x^6 + x^8 + x^{11} + x^{14}$
$g_3(x)$	$1 + x + x^2 + x^6 + x^9 + x^{10} + x^{14}$
$g_4(x)$	$1 + x^4 + x^7 + x^8 + x^{10} + x^{12} + x^{14}$
$g_5(x)$	$1 + x^2 + x^4 + x^6 + x^8 + x^9 + x^{11} + x^{13} + x^{14}$
$g_6(x)$	$1 + x^3 + x^7 + x^8 + x^9 + x^{13} + x^{14}$
$g_7(x)$	$1 + x^2 + x^5 + x^6 + x^7 + x^{10} + x^{11} + x^{13} + x^{14}$
$g_8(x)$	$1 + x^5 x^8 + x^9 + x^{10} + x^{11} + x^{14}$
$g_9(x)$	$1 + x + x^2 + x^3 x^9 + x^{10} x^{14}$
$g_{10}(x)$	$1 + x^3 + x^6 + x^9 + x^{11} + x^{12} + x^{14}$
$g_{11}(x)$	$1 + x^4 + x^{11} + x^{12} + x^{14}$
$g_{12}(x)$	$1 + x + x^2 + x^3 + x^5 + x^6 + x^7 + x^8 + x^{10} + x^{13} + x^{14}$

Tabela 2.5: Polinômios geradores do código BCH para um FECFRAME curto contendo $n_{\text{ldpc}} = 16200$ bits.

tarefa pelo codificador, quando a informação está contida num quadro normal, pode ser resumida em alguns passos[1]:

- Primeiramente iniciamos os bits de paridade com zeros,

  $p_0 = p_1 = p_2 = \ldots = p_{n_{\text{ldpc}} - k_{\text{ldpc}} - 1} = 0$

- Atribuimos o conteúdo do primeiro bit de informação, o bit i_0, aos bits de paridade cujos endereços estão especificados na primeira linha das tabelas existentes no anexo B da norma referênciada em [1]. Nessa referência encontramos onze tabelas, numeradas de B.1 a B.11, cada uma referente a um modo de codificação LDPC. Podemos usar como exemplo uma codificação com taxa 8/9. Neste caso devemos utilizar a Tabela B.10. Lembrando que as operações são feitas no Corpo de Galois, temos

$$p_0 = p_0 \oplus i_0 \qquad p_{2848} = p_{2848} \oplus i_0$$
$$p_{6235} = p_{6235} \oplus i_0 \qquad p_{3222} = p_{3222} \oplus i_0$$

- Os próximos 359 bits do bloco de informação, os bits i_m, onde $m = 1, 2, \ldots, 359$, serão atribuídos aos bits de paridade p_n cujos endereços serão calculados a partir da seguinte fórmula

$$n = \{x + m \mod 360 \times q\} \mod (n_{ldpc} - k_{ldpc}) \tag{2.4}$$

  onde x corresponde a posição dos bits de paridade aos quais o primeiro bit i_0 foi atribuído e q pode ser encontrado na Tabela 2.6 segundo a taxa do código utilizado. Dando continuidade ao nosso exemplo, para a taxa de código 8/9, $q = 20$, então fazendo as atribuições para o bit i_1 temos

$$p_{20} = p_{20} \oplus i_1 \qquad p_{2868} = p_{2868} \oplus i_1$$
$$p_{6255} = p_{6255} \oplus i_1 \qquad p_{3242} = p_{3242} \oplus i_1$$

- Quando chegarmos ao bit i_{360}, devemos voltar à tabela do Anexo B de [1] e utilizar os bits que estão na segunda linha da mesma. Como fizemos no segundo passo, vamos atribuir o valor de i_{360} aos bits de paridade cujo os endereços estão especificados na segunda linha da tabela correspondente à taxa de código. Os 359 bits seguintes, ou seja, os bit i_m sendo $m = 361, 362, \ldots, 719$ serão atribuidos a bits de paridade de forma similar ao que foi realizado no passo três, utilizando a Equação 2.4, mas agora x se refere aos endereços dos bits de paridade aos quais o bit i_{360} foi atribuido.

- De maneira similiar, cada grupo de 360 bits deve ser tratado como no passo anterior e, a cada grupo novo, devemos usar a linha seguinte na tabela do anexo B da norma em [1] para achar os novos endereços dos bits de paridade.

- Após realizar essas operações com todos os bits do bloco de informação que queremos codificar, devemos realizar a seguinte operação de forma sequencial, começando com $i = 1$

$$p_i = p_i \oplus p_{i-1} \qquad i = 1, 2, \ldots, n_{\text{ldpc}} - k_{\text{ldpc}} - 1 \tag{2.5}$$

- Finalmente, após todas estas operações, temos em p_i sendo $i = 0, 1, 2, \ldots, n_{\text{ldpc}} - k_{\text{ldpc}} - 1$ os bits de paridade.

Se estivermos trabalhando com quadros curtos, ou seja, com $n_{\text{ldpc}} = 16200$ bits, precisamos realizar as mesmas operações, porém devemos substituir as tabelas encontradas no anexo B de [1] pelas tabelas do anexo C da mesma referência, e também devemos utilizar a Tabela 2.7 ao invés da Tabela 2.6.

Taxa de Código	**q**
1/4	135
1/3	120
2/5	108
1/2	90
3/5	72
2/3	60
3/4	45
4/5	36
5/6	30
8/9	20
9/10	18

Tabela 2.6: Valores de q para um FECFRAME normal contendo $n_{\text{ldpc}} = 64800$ bits.

2.5.3 Interleaving

É necessário embaralhar os bits para separá-los e assim evitar erros em rajadas comuns no meio utilizado por satélites para transmissão. Contudo, não se faz isso em todos os casos. Executa-se esta operação quando modula-se o sinal transmitido com 8PSK, 16APSK ou 32APSK. Como esses sinais precisam de uma demodulação mais precisa, já que símbolos ficam mais próximos nestas constelações, deve-se intercalar os bits do quadro que obtemos

Taxa de Código	q
1/4	36
1/3	30
2/5	27
1/2	25
3/5	18
2/3	15
3/4	12
4/5	10
5/6	8
8/9	5

Tabela 2.7: Valores de q para um FECFRAME curto contendo $n_{\text{ldpc}} = 16200$ bits.

na saída do módulo de codificação LDPC para ajudar o receptor na correção dos bits que foram corrompidos durante a transmissão. Este processo é feito da seguinte maneira:

- primeiramente os bits do stream devem ser escritos de forma serial na direção coluna;
- depois devem ser lidos, também serialmente, na direção linha, começando pelo bit mais significativo (exceto quando utilizamos a modulação 8PSK com taxa 3/5, neste caso o bit mais significativo será o terceiro a ser lido).

As Figuras 2.7 e 2.8 mostram de forma gráfica estas operações e na Tabela 2.8 encontramos o número de linhas e colunas que utilizamos para cada tipo de modulação e tamanho de quadro (n_{ldpc}).

Modulação	Linhas (para $n_{ldpc} = 64800$)	Linhas (para $n_{ldpc} = 16200$)	Colunas
8PSK	21600	5400	3
16APSK	16200	4050	4
32APSK	12960	3240	5

Tabela 2.8: Estrutura para intercalação de bits.

2.6 Mapeamento de Bits na Constelação

Uma das inovações mais importantes no DVB-S2 foi a inclusão de novas modulações digitais. Este tipo de modulação mapeia um certo número de bits em um símbolo de uma constelação correspondente à modulação digital desejada. Quanto mais bits por símbolo o sistema é capaz de modular, maior é sua taxa de transmissão. O DVB-S utilizava apenas

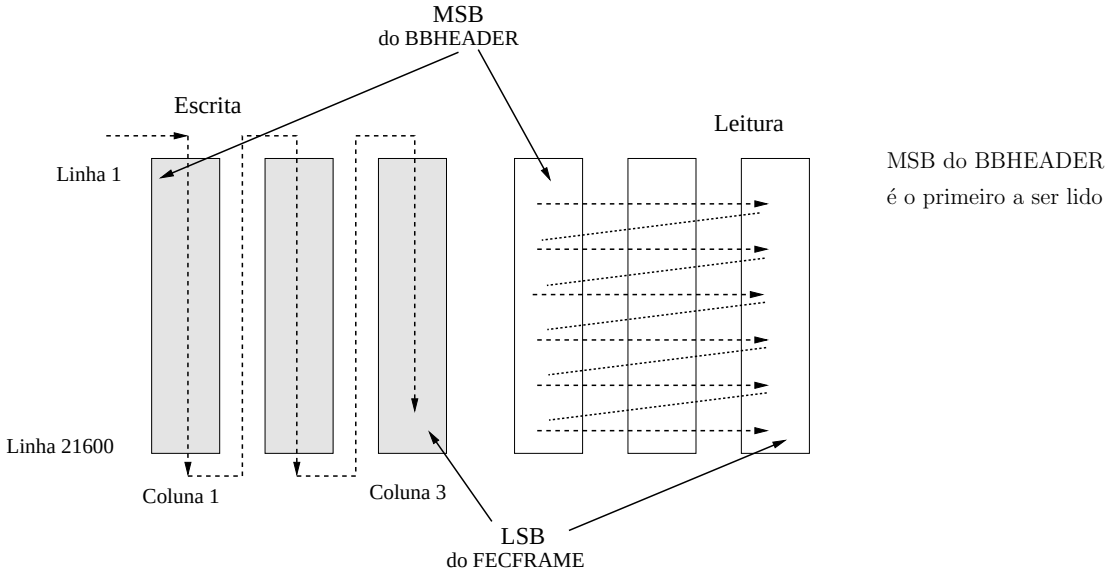


Figura 2.7: Interleaving para 8PSK e com FECFRAME normal (todas as taxas, exceto 3/5).

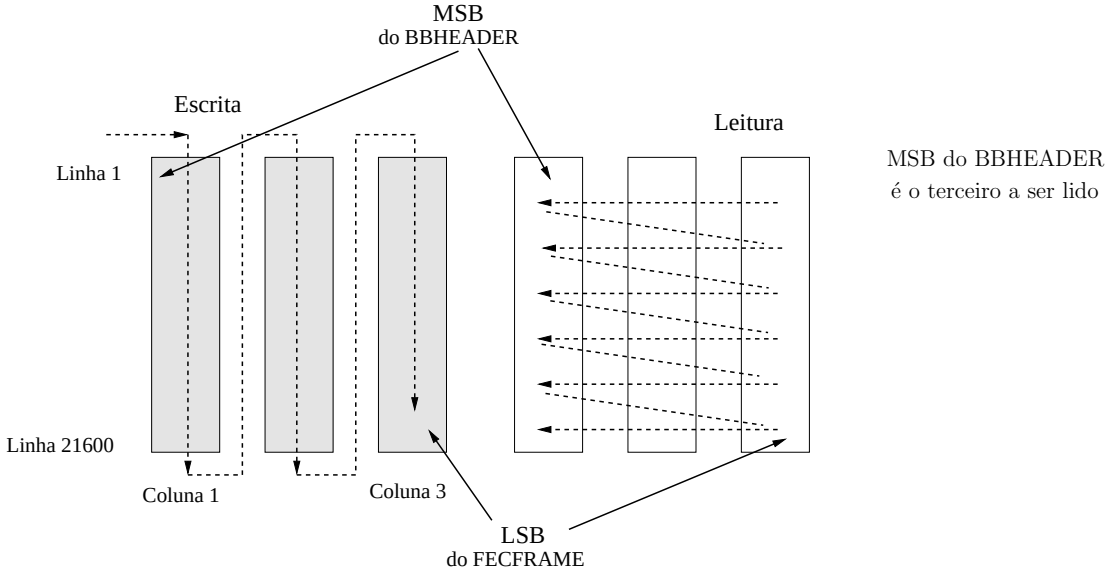


Figura 2.8: Interleaving para 8PSK e com FCEFRAME normal (taxa de 3/5).

dois tipos de modulação, o BPSK e o QPSK. O DVB-S2 usa quatro tipos de modulação, mantiveram o QPSK e adicionaram o 8PSK, o 16APSK e o 32APSK.

O sinal recebido na entrada do subsistema pode se apresentar de duas maneiras diferentes, ele pode ser um FECFRAME normal contendo 64800 bits ou um FECFRAME curto com 16200 bits. Cada FECFRAME precisa passar por uma conversão serial-paralelo de acordo com a modulação na qual será mapeada, mantendo sempre a ordem, sendo o bit mais significativo do FECFRAME sempre o primeiro bit na sequência paralela. Para cada

uma das modulações utilizadas, será formada uma sequência paralela segundo sua eficiência de modulação η_{MOD}, que vale 2 para QPSK, 3 para 8PSK, 4 para 16APSK e 5 para 32APSK.

As sequências paralelas geradas são mapeadas em uma das quatro constelações das modulações citadas acima, formando sequências de tamanhos variáveis (I,Q) segundo η_{MOD}. Estas sequências são vetores complexos (onde I representa a componente em fase e Q a componente em quadratura), também podem ser calculadas a partir do módulo ρ e da fase $\exp(j\phi)$. Então damos à sequência paralela o nome de XFECFRAME (este nome vem do inglês *compleX* FECFRAME).

Nos tópicos a seguir, temos mais detalhes sobre cada uma das modulações utilizadas pelo sistema DVB-S2. Aqui são explicados como são formados os símbolos e serão apresentadas as constelações de cada uma dessas modulações.

2.6.1 QPSK

Para a modulação QPSK (*Quadrature Phase Shift Keying*), o sistema deve utilizar o código de Gray[1] convencional com mapeamento absoluto. O mapeamento na constelação QPSK deve ser como o mostrado na Figura 2.9. A energia média normalizada por símbolo deve ser igual a $\rho^2 = 1$.

Dois bits do FECFRAME são mapeados para um símbolo QPSK, então o i-ésimo símbolo QPSK deve ser formado pelos bits $2i$ e $2i+1$ do quadro, onde $i = 0, 1, 2, \ldots, (\mathrm{N}/2)-1$ e N é o tamanho do quadro originado pelo codificador LDPC.

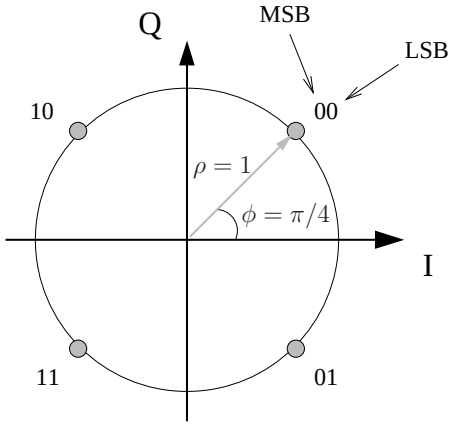


Figura 2.9: Constelação da modulação QPSK.

[1]Código de Gray é um sistema de código binário onde os números sequênciais variam apenas de um bit.

2.6.2 8PSK

Para a modulação 8PSK (*Phase Shift Keying*), o sistema deve utilizar o código de Gray convencional com mapeamento absoluto. O mapeamento na constelação 8PSK deve ser como mostra a Figura 2.10. A energia normalizada por símbolo deve ser igual a $\rho^2 = 1$.

Três bits do FECFRAME são mapeados para um símbolo 8PSK, então o i-ésimo símbolo 8PSK deve ser formado pelos bits $3i$, $3i + 1$ e $3i + 2$ do quadro onde $i = 0, 1, 2, \ldots, (\mathrm{N}/3) - 1$ e N é o tamanho do quadro originado pelo codificador LDPC.

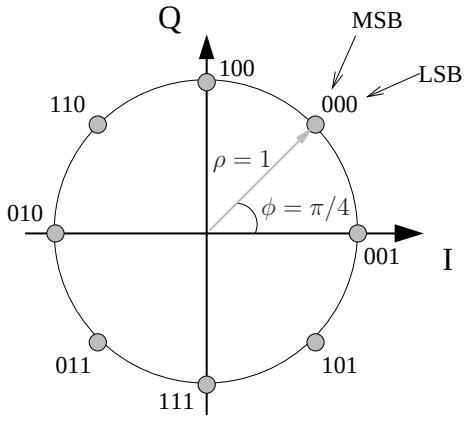


Figura 2.10: Constelação da modulação 8PSK.

2.6.3 16APSK

A constelação da modulação 16APSK (*Amplitude and Phase Shift Keying*), que pode ser vista na Figura 2.11, deve ser composta por duas circunferências concêntricas com 4 e 12 pontos modulados com PSK, com os raios valendo R_1 e R_2 respectivamente. A razão entre o raio da circunferência externa e o raio da interna ($\gamma = \mathrm{R}_2/\mathrm{R}_1$) deve ser como descrito na Tabela 2.9.

São admitidos dois valores para as amplitudes da constelação, permitindo uma otimização na performance de acordo com as características do canal:

- E = 1 (sendo E a unidade de energia média do símbolo) correspondendo a $[\mathrm{R}_1]^2 + 3[\mathrm{R}_2]^2 = 4$;
- $\mathrm{R}_2 = 1$.

Quatro bits do FECFRAME são mapeados para um símbolo 16APSK, então o i-ésimo símbolo 16APSK deve ser formado pelos bits $4i$, $4i+1$, $4i+2$ e $4i+3$ do quadro onde $i = 0, 1, 2, \ldots, (\mathrm{N}/4) - 1$ e N é o tamanho do quadro originado pelo codificador LDPC.

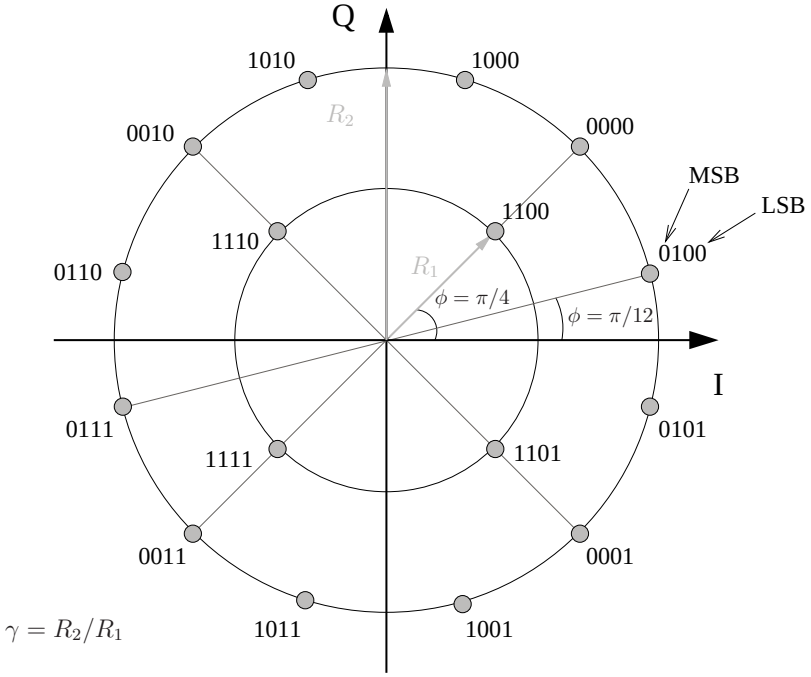


Figura 2.11: Constelação da modulação 16APSK.

Taxa de Código	**Eficiência Espectral de Codificação/Modulação**	γ
2/3	2,66	3,15
3/4	2,99	2,85
4/5	3,19	2,75
5/6	3,32	2,70
8/9	3,55	2,60
9/10	3,59	2,57

Tabela 2.9: Razão ótima γ para constelação da modulação 16APSK, usando um canal linear [1].

2.6.4 32APSK

A constelação da modulação 32APSK, que pode ser vista na Figura 2.12, deve ser composta por três circunferências concêntricas com 4, 12 e 16 pontos modulados com PSK, com os raios valendo R_1, R_2 e R_3 respectivamente. A razão entre o raio da circunferência intermediária e o raio da interna é representado por $\gamma_1 = \mathrm{R}_2/\mathrm{R}_1$ e a razão entre o raio da

circunferência externa e o raio da interna é representado por $\gamma_2 = \mathrm{R}_3/\mathrm{R}_1$. Ambas razões devem ter os valores descritos na Tabela 2.10.

São admitidos dois valores para as amplitudes da constelação, permitindo uma otimização na performance de acordo com as características do canal:

- E = 1 (sendo E a unidade de energia média do símbolo) correspondendo a $[\mathrm{R}_1]^2 + 3[\mathrm{R}_2]^2 + 4[\mathrm{R}_3]^2 = 8$;
- $\mathrm{R}_3 = 1$.

Cinco bits do FECFRAME são mapeados para um símbolo 32APSK, então o i-ésimo símbolo 32APSK deve ser formado pelos bits $5i$, $5i+1$, $5i+2$, $5i+3$ e $5i+4$ do quadro onde $i = 0, 1, 2, \ldots, (\mathrm{N}/5) - 1$ e N é o tamanho do quadro originado pelo codificador LDPC.

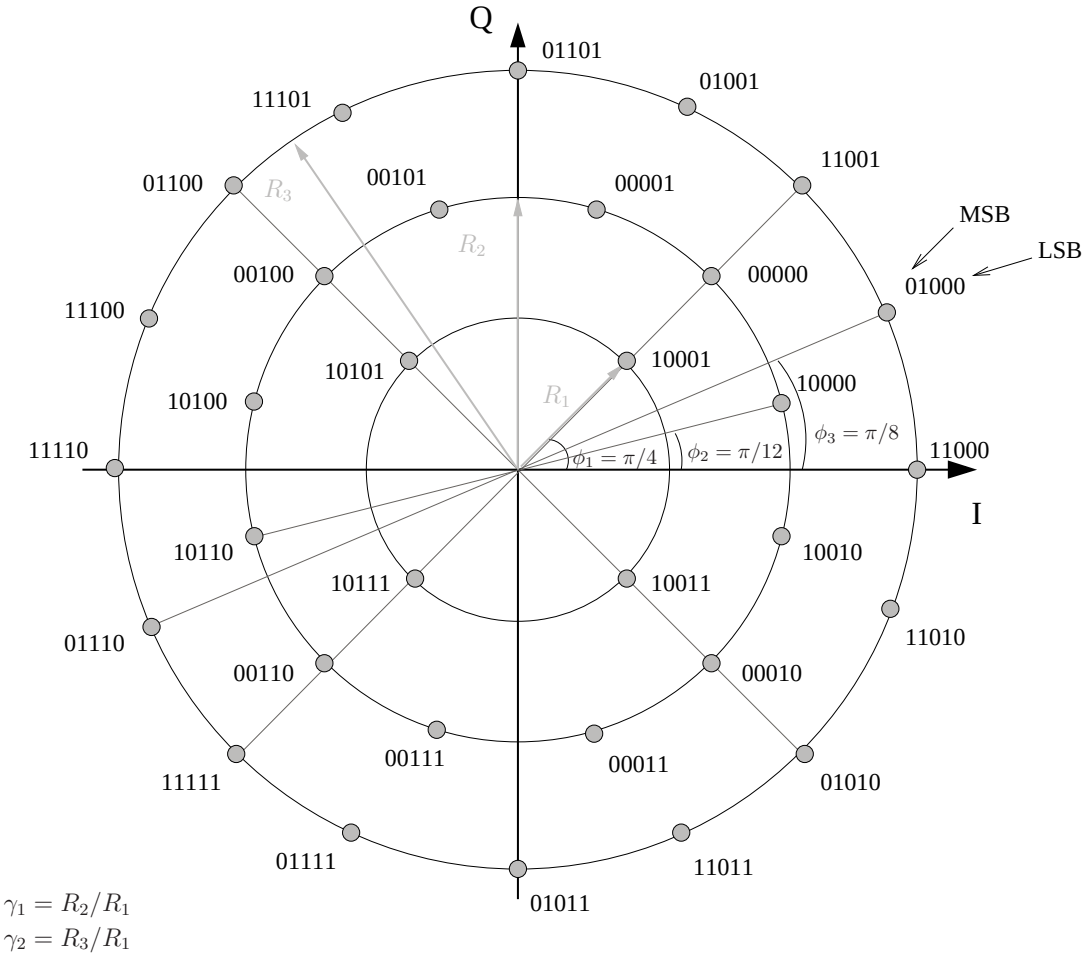


Figura 2.12: Constelação da modulação 32APSK.

Taxa de Código	Eficiência Espectral de Codificação/Modulação	γ_1	γ_2
3/4	3,74	2,84	5,27
4/5	3,99	2,72	4,87
5/6	4,15	2,64	4,64
8/9	4,43	2,54	4,33
9/10	4,49	2,53	4,30

Tabela 2.10: Razões ótimas γ_1 e γ_2 para constelação da modulação 32APSK, usando um canal linear [1].

2.7 Formação de Quadros da Camada Física

Após mapear os bits do sinal em símbolos segundo as modulações utilizadas para a transmissão, o sistema precisa preparar o sinal e colocá-lo num formato adequado para ser enviado pelo canal. O subsistema abordado nesta seção faz essa preparação do sinal formando quadros que serão entregues na camada física e por essa razão damos o nome de subsistema de Formação de Quadros da Camada Física a esta parte do sistema.

Os quadros gerados pelo subsistema recebem o nome de PLFRAME e, para gerar os mesmos, o subsistema precisa realizar algumas tarefas, além de se manter sempre sincronizado com o subsistema de Codificação FEC. Essas tarefas podem ser divididas da seguinte maneira:

- Geração de *dummy* PLFRAMEs quando não há um XFECFRAME pronto na entrada deste subsistema para ser executado e transmitido. Dessa maneira mantém-se a taxa de transmissão;

- Divisão do XFECFRAME em um número inteiro S constante de SLOTS, cada um contendo 90 símbolos;

- Todo quadro precisa ser acompanhado de um cabeçalho que indica o conteúdo dele e assim ajudar o receptor a entender o sinal contido no quadro. Esse cabeçalho do quadro da camada física recebe o nome de PLHEADER e sua formação está sob a responsabilidade deste subsistema. O PLHEADER deve ocupar exatamente um SLOT, cujo o comprimento é de 90 símbolos como dito anteriormente;

- Nem sempre é necessário enviar símbolos pilotos, mas para os modos que requerem estes símbolos, devem ser inseridos blocos de pilotos a cada 16 SLOTS para ajudar a sincronização. Este bloco de pilotos deve ser composto por $P = 36$ símbolos pilotos;

- A última tarefa a ser realizada pelo subsistema de Formação de Quadros da Camada Física é embaralhar os símbolos modulados (I,Q) para dispersão de energia. Chamamos isto de *scrambling* em nível de camada física.

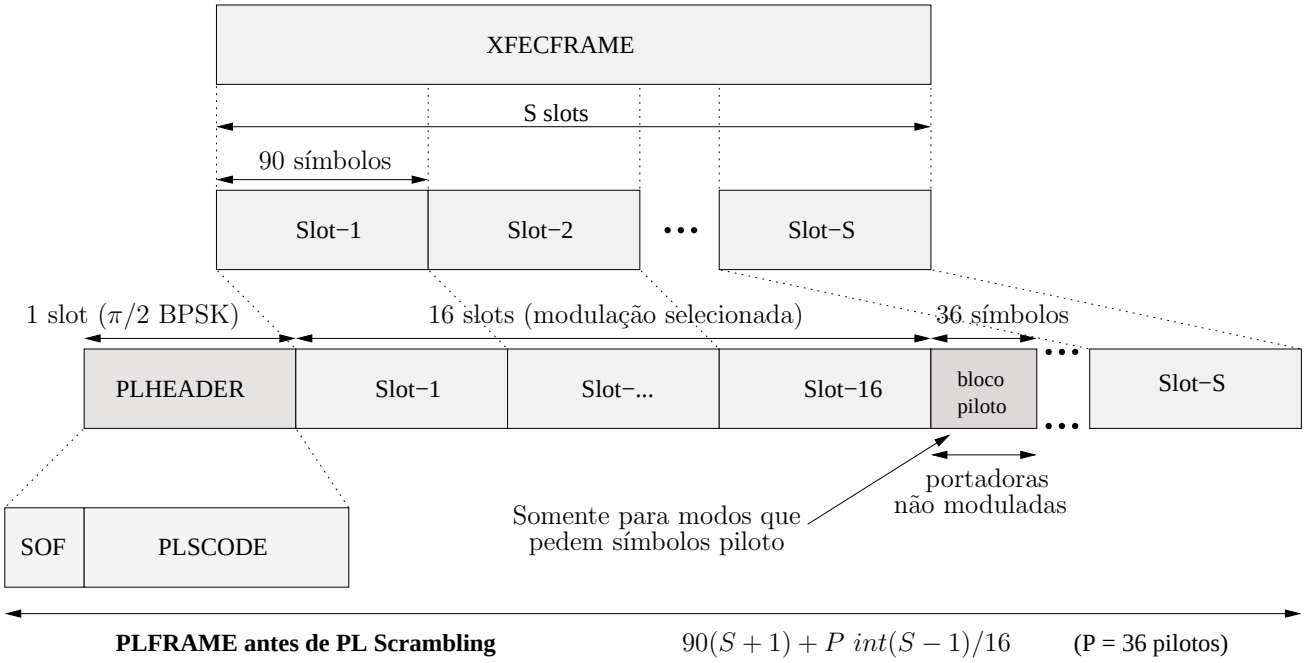


Figura 2.13: Formato do PLFRAME.

A Figura 2.13 mostra a formação de um quadro da camada física PLFRAME a partir de um quadro XFECFRAME, representando graficamente o resultado dos processos mencionados anteriormente. O número S de SLOTS no qual o XFECFRAME é dividido consta na Tabela 2.11 segundo as características do tamanho de quadro utilizado e da eficiência de modulação. Nas próximas subseções, veremos mais detalhes de cada uma das tarefas executadas por este subsistema para produzir o PLFRAME.

	$n_{ldpc} = 64800$ **(quadro normal)**		$n_{ldpc} = 16200$ **(quadro curto)**	
η_{MOD} (bit/s/Hz)	**S**	**η % sem piloto**	**S**	**η% sem piloto**
2	360	99,72	90	98,90
3	240	99,59	60	98,36
4	180	99,45	45	97,83
5	144	99,31	36	97,30

Tabela 2.11: Número S de SLOTS por XFECFRAME.

2.7.1 Inserção de *Dummy* PLFRAMEs

Do início ao final, o sistema DVB-S2 deve manter uma taxa constante de informação para que o receptor não perca o sincronismo. Mas, às vezes, alguns processos ao longo do sistema podem gerar atrasos,como por exemplo o mapeamento de bits na constelação que vimos na Seção 2.6. Para que isso não afete o bom funcionamento do sistema, existe um módulo responsável por gerar quadros de camada física sem valor útil, os chamados *dummy* PLFRAMEs. Toda vez que não é detectada a presença de um XFECFRAME na entrada do subsistema de Formação de Quadros da Camada Física, estes quadros sem informação útil são enviados para manter a taxa transmissão constante.

Um *dummy* PLFRAME deve ser igual a um PLFRAME comum, deve conter um cabeçalho, ou seja, um PLHEADER, como veremos na Seção 2.7.2, e 36 SLOTS de portadoras não moduladas ($I = (1/\sqrt{2}), Q = (1/\sqrt{2})$), dessa maneira o receptor perceberá que se trata de um *dummy* PLFRAME e o descartará quando receber.

2.7.2 Sinalização da Camada Física

Sempre que é formado um quadro, este precisa de um cabeçalho que indique detalhes de seu conteúdo. Assim, quando o sinal for recebido pelo receptor, este poderá se sincronizar com o transmissor e irá saber o conteúdo que o quadro carrega sem precisar ler o sinal inteiro. Este módulo de sinalização da camada física é responsável pela formação do PLHEADER, que é o cabeçalho do quadro transmitido na camada física. O PLHEADER, que ocupa necessariamente um SLOT de 90 símbolos, é composto pelos seguintes campos:

- **SOF** (26 símbolos): indica o começo do quadro. Seu nome em inglês significa *Start Of Frame*;
- código **PLS** (64 símbolos): o código PLS (*Physical Layer Signalling*) é um código binário não sistemático de comprimento 64 e dimensão 7 com distância mínima (d_{min}) igual a 32. Os 7 bits transmitidos formam 2 campos:
  - MODCOD (5 símbolos): identifica a modulação do XFECFRAME e a taxa FEC;
  - TYPE (2 símbolos): identifica o comprimento do FECFRAME (64800 bits ou 16200 bits) e a presença ou ausência de símbolos piloto.

O PLHEADER é representado pela sequência binária $(y_1, y_2, \ldots\ y_{90})$ e estes bits serão transformados em 90 símbolos modulados em $\pi/2$-BPSK. Para realizar esta modulação basta seguir a regra abaixo:

$$I_{2i-1} = Q_{2i-1} = (1/\sqrt{2})(1 - 2y_{2i-1}),\ \text{para}\ i = 1, 2, \ldots, 45 \tag{2.6}$$

$$I_{2i} = -Q_{2i} = -(1/\sqrt{2})(1 - 2y_{2i}),\ \text{para}\ i = 1, 2, \ldots, 45 \tag{2.7}$$

Seguindo as regras descritas pelas equações 2.6 e 2.7, vemos que a constelação de $\pi/2$-BPSK é bem parecida com a constelação de QPSK. Porém, apesar dessa modulação ter quatro símbolos, somente dois valores são representados, os bits zero e um, como na modulação BPSK. Estudando melhor a $\pi/2$-BPSK, observamos que a cada bit modulado, seus símbolos correspondentes se diferenciam do próximo bit modulado por uma rotação de $\pi/2$ radianos [10]. Juntando essas informações, entendemos o porque dessa modulação ser chamada de $\pi/2$-BPSK.

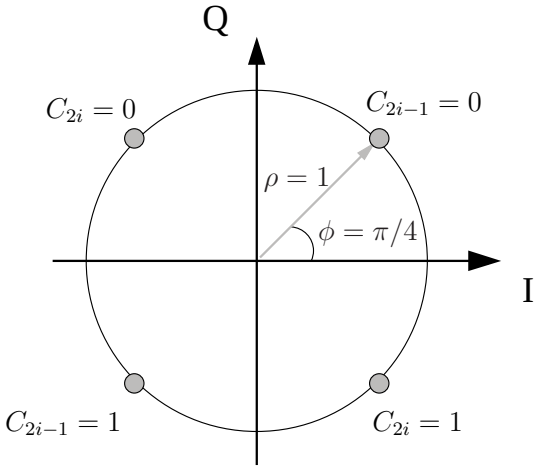


Figura 2.14: Constelação da modulação $\pi/2$-BPSK.

Na figura 2.14, vemos graficamente como são representados os símbolos da modulação $\pi/2$-BPSK, onde os símbolos nomeados de C_{2i-1} são aqueles que representam os bits ímpares modulados por $I_{2i-1} + jQ_{2i-1}$ e os símbolos C_{2i} representam os bits pares modulados por $I_{2i} + jQ_{2i}$. Como explicado na norma referênciada em [11], os símbolos C_{2i-1} e C_{2i} são transmitidos alternadamente e pelo gráfico podemos observar melhor a rotação de $\pi/2$ explicada anteriormente.

Campo SOF

Este campo deve conter a sequência $18D2E82_{\text{HEX}}$ em binário (ou seja, deve conter $01 - 1000 - \ldots - 0010$, sendo o bit a esquerda o bit mais significativo), assim quando o receptor receber esta sequência, saberá que aí começa um novo quadro.

Campo MODCOD

MODCOD é representado por 5 bits que identificam a taxa de código usada pelo sistema, que pode assumir um dentre os valores do conjunto η_C = [1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9, 9/10], e a modulação segundo a eficiência espectral de cada uma delas, η_{MOD} = [2, 3, 4, 5]. A Tabela 2.12 mostra as combinações aceitas pelo sistema e qual é o valor correspondente que o campo MODCOD assume. Na tabela este valor está representado em decimal, mas no quadro o mesmo valor estará no formato binário.

Mode	**MOD COD**	**Mode**	**MOD COD**	**Mode**	**MOD COD**	**Mode**	**MOD COD**
QPSK 1/4	1_D	QPSK 5/6	9_D	8PSK 9/10	17_D	32APSK 4/5	25_D
QPSK 1/3	2_D	QPSK 8/9	10_D	16APSK 2/3	18_D	32APSK 5/6	26_D
QPSK 2/5	3_D	QPSK 9/10	11_D	16APSK 3/4	19_D	32APSK 8/9	27_D
QPSK 1/2	4_D	8PSK 3/5	12_D	16APSK 4/5	20_D	32APSK 9/10	28_D
QPSK 3/5	5_D	8PSK 2/3	13_D	16APSK 5/6	21_D	Reservado	29_D
QPSK 2/3	6_D	8PSK 3/4	14_D	16APSK 8/9	22_D	Reservado	30_D
QPSK 3/4	7_D	8PSK 5/6	15_D	16APSK 9/10	23_D	Reservado	31_D
QPSK 4/5	8_D	8PSK 8/9	16_D	32APSK 3/4	24_D	DUMMY PLFRAME	0_D

Tabela 2.12: Codificação do campo MODCOD [1].

Campo TYPE

Este campo do PLHEADER é composto por 2 símbolos, o bit mais significativo indica o tamanho do FECFRAME usado (se 0 é um quadro normal com 64800 bits, se 1 o quadro usado é curto com 16200 bits). O bit menos significativo de TYPE é utilizado para identificar as configurações de símbolo piloto (o valor 0 significa que não é utilizado piloto e o valor 1 o contrário), vamos explicar a utilidade desses símbolos mais adiante na seção 2.7.3.

Código PL

Os campos TYPE e MODCOD devem ser codificados gerando 64 bits, para isso utilizamos um código bi-ortogonal (64,7) que é construído a partir de um código, também bi-ortogonal, (32,6) como mostra a Figura 2.15. Analisando melhor esta representação gráfica do código PLS, podemos perceber que esta construção garante que os bits ímpares de sua saída terão sempre valores iguais ou opostos aos bits imediatamente anteriores e o que decide entre esses dois valores é o bit b_7, que é o bit menos significativo do campo TYPE.

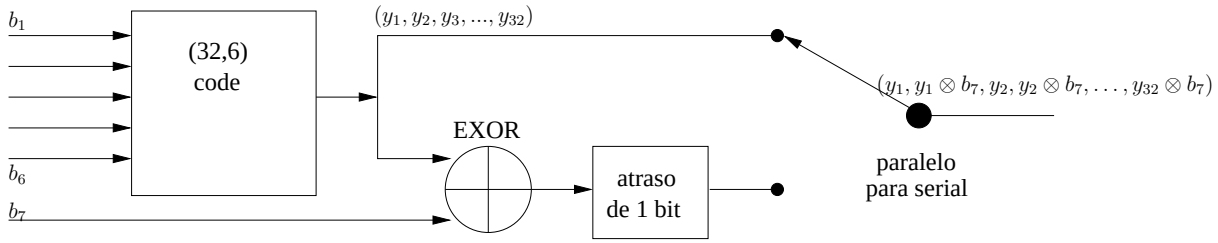


Figura 2.15: Representação gráfica do código PLS.

Existem diversas maneira de se construir um código em bloco linear $[64, 7, 32]$, pode ser como um código BCH extendido, um dual de um código de Hamming extendido, um código de máximo comprimento extendido ou um código Reed-Müller de primeira ordem[12]. O código escolhido pelo padrão DVB-S2 foi o de Reed-Müller e este foi o código implementado no simulador. O campo MODCOD e o bit mais significativo do TYPE são codificados por esse código linear de comprimento 32 cuja matriz geradora está representada em 2.8.

$$G = \begin{bmatrix}
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix} \tag{2.8}$$

Depois de codificar os primeiros seis bits com o código de Reed-Muller e fazer a soma binária dos 32 bits codificados com o bit menos significativo de TYPE, fazemos um *scrambling* nos bits gerados. Para fazer esta operação nos 64 bits obtidos, utilizamos a seguinte sequencia:

01110001100111011000001111001001010100110100001000101101111111010

2.7.3 Inserção de Símbolos Pilotos

O ar, que é meio utilizado pelo satélite para a transmissão de sinal, é um meio ruidoso e algumas de suas características podem causar problemas na recepção deste sinal enviado. Dois desses problemas são a dificuldade para a sincronização entre transmissores e receptores e a perda de informação no sinal causada por interferência ou desvanecimento. Para ajudar a solucionar estes problemas, o sistema DVB-S2 dá ao usuário a possibilidade de usar símbolos pilotos no PLFRAME. Estes símbolos pilotos são símbolos conhecidos de antemão pelos receptores e o transmissor tem a escolha de fazer uso deles ou não, sua escolha é indicada no campo TYPE, como mencionado anteriormente na seção 2.7.2.

Com a perda do sincronismo entre transmissor e receptor, este último desconhece o início do quadro que deve decodificar. Para recuperar a sincronização de quadro, existem alguns métodos propostos pelas normas das referências [1] e [12]. Um dos métodos utilizados envolve a utilização de símbolos piloto para a recuperação da portadora do sinal, que ajuda a sincronização do receptor.

As perdas de informação pelo sinal transmitido causadas pelo desvanecimento ou pela interferência, dificulta a demodulação do sinal pelo receptor. Para ter uma recepção melhor, faz-se o uso de uma técnica que estima o estado do canal. Assim o receptor saberá as modificações impostas ao sinal e fará uma demodulação com menos erros. Para podermos estimar o canal, o transmissor envia de tempos em tempos símbolos piloto. Ao receber estes símbolos, o receptor faz alguns cálculos com eles e com os símbolos pilotos que possui em sua memória e assim é capaz de estimar o canal. Na norma [12], encontramos uma sugestão de estimador de canal para o sistema DVB-S2.

Estes símbolos pilotos são enviados em conjunto em intervalos iguais dentro de um quadro. Um BLOCO PILOTO é composto por 36 símbolos pilotos e cada um deve ser um símbolo não modulado identificado por I= $(1/\sqrt{2})$, Q= $(1/\sqrt{2})$. O primeiro BLOCO PILOTO deve ser inserido após 16 SLOTS contando a partir do PLHEADER, o segundo deve ser inserido após 32 SLOTS e os subsequentes serão inseridos após cada múltiplo de 16. Porém se a posição que o BLOCO PILOTO deve ser inserido coincidir com o começo de

um SOF, ou seja, o início do cabeçalho do próximo quadro, então ele não será transmitido. Quando trabalhamos nos modos VCM ou ACM, a configuração de símbolo piloto pode mudar a cada quadro.

2.7.4 *Scrambling* da Camada Física

Pode ocorrer que no sinal que será enviado haja uma sequência grande de apenas zeros ou de apenas uns e estes bits iguais seguidos podem provocar alguns problemas, como por exemplo um problema de sincronização do receptor. Sem transições binárias em quantidade adequada, o receptor pode perder sua sincronização. Essas sequências de bits repetidos produzem na saída do sistema um espectro de sinal não uniforme. Essa característica pode acentuar o efeito de não-linearidade do canal. Para amenizar esses problemas, podemos fazer um *scrambling* a nível de camada física que irá quebrar estas longas sequências de bits repetidos dispersando a energia do sinal. Ao longo desta seção será explicado como este processo é feito no sistema DVB-S2.

Para fazer essa dispersão de energia, cada PLFRAME, sem incluir o PLHEADER, terá suas amostras (I + jQ) do quadro multiplicadas por uma sequência complexa randômica (C_I + jC_Q) obtendo:

$$I_{Scrambled} = (IC_I - QC_Q); \tag{2.9}$$

$$Q_{Scrambeld} = (IC_Q + QC_I) \tag{2.10}$$

A sequência complexa randômica deve ser inicializada ao final de cada PLHEADER, que é quando começa a parte do novo quadro a ser embaralhado, na Figura 2.16 podemos ver isto claramente. A duração do PLFRAME depende da modulação usada para transmissão, como vimos na Tabela 2.11, portanto a sequência randômica deve ser truncada para ficar com o mesmo comprimento do quadro, uma vez que ela geralmente tem um comprimento maior que este.

As sequências do código de *scrambling* são construídas combinando duas sequências reais (geradas por polinômios geradores de grau 18) para formar uma sequência complexa. As sequências obtidas por este processo geram segmentos de um conjunto de sequências de Gold.

Figura 2.16: PLFRAME após passar pelo módulo de *scrambling*.

Como exemplo, vamos considerar que temos as sequências x e y, sendo a sequência x construída a partir do polinômio primitivo, no corpo de Galois (GF(2)), $1 + x^7 + x^{18}$ e a sequência y construída a partir do polinômio $1 + y^5 + y^7 + y^{10} + y^{18}$.

A sequência que depende da escolha do número do código de *scrambling* é chamado de z_n. Vamos determinar que $x(i)$, $y(i)$ e $z_n(i)$ sejam os i-ésimos símbolos de cada uma das sequências x, y e z_n respectivamente. As sequências x e y são construídas como mostramos a seguir:

- Condições iniciais:
  - x é construída com os primeiros símbolos sendo $x(0) = 1$,
    $x(1) = x(2) = x(3) = \ldots = x(16) = x(17) = 0$;
  - $y(0) = y(1) = \ldots = y(16) = y(17) = 1$.
- Os símbolos seguintes são definidos de forma recursiva da seguinte maneira:
  - $x(i + 18) = x(i + 17) + x(i)$ módulo 2, para $i = 0, \ldots, 2^{18} - 20$;
  - $y(i+18) = y(i+10)+y(i+7)+y(i+5)+y(i)$ módulo 2, para $i = 0, \ldots, 2^{18} - 20$.

A n-ésima sequência do código de Gold z_n, para n = 0,1,2, $\ldots, 2^{18} - 2$, é definida como:

- $z_n(i) = [x((i + n) \text{ módulo } (2^{18} - 1)) + y(i)]$ módulo 2, para $i = 0, \ldots, 2^{18} - 2$.

Essas sequências binárias são convertidas para uma sequência inteira R_n (que assume valores 0, 1, 2 e 3) através da seguinte transformação:

$$R_n = 2z_n((i + 131072) \text{ módulo } (2^{18} - 1)) + z_n(i), \quad \text{para } i = 0, 1, \ldots, 66419$$

A n-ésima sequência complexa do código de *scrambling* é definida como:

$$\mathrm{C}_I(i) + j\mathrm{C}_Q(i) = \exp\left(jR_n(i)\pi/2\right) \tag{2.11}$$

Na Tabela 2.13 temos o resultado das amostras do quadro a ser embaralhado para cada uma das quatro opções de valores de R_n. É fácil chegar a estes resultados considerando que as amostras do quadro e sequência definida pela equação 2.11, que pode assumir os valores 1, j, -1 ou -j.

$\mathbf{R_n}$	$\mathbf{exp(jR_n\pi/2)}$	$\mathbf{I_{Scrambled}}$	$\mathbf{Q_{Scrambled}}$
0	1	I	Q
1	j	-Q	I
2	-1	-I	-Q
3	-j	Q	-I

Tabela 2.13: Possíveis valores para as saídas do módulo PLScrambler [1].

2.8 Modulação em Quadratura e Modelagem de Banda Base

Nesse momento, o sinal está quase pronto para ser transmitido, a não ser por um detalhe: teoricamente a largura de banda de um pulso retangular é infinita, então precisamos limitá-la para poder enviar o sinal na banda disponível. Então após executar o *scrambling* descrito na seção 2.7.4, devemos filtrar o sinal com a raiz quadrada de um coseno-elevado (*raised cosine*) e o fator de *roll-off* α pode ser 0,35 , 0,25 ou 0,20.

A função teórica do filtro em banda base da raiz quadrada do *raised cosine* é definida pelas seguintes expressões:

$$H(f) = 1 \quad \text{para } |f| < f_N(1 - \alpha) \tag{2.12}$$

$$H(f) = \left\{ \frac{1}{2} + \frac{1}{2}\text{sen}\frac{\pi}{2f_N} \left[\frac{f_N - |f|}{\alpha} \right] \right\}^{1/2} \quad \text{para } f_N(1-\alpha) \tag{2.13}$$

$$H(f) = 0 \quad \text{para } |f| > f_N(1+\alpha) \tag{2.14}$$

onde $f_N = \frac{1}{2T_S} = \frac{R_S}{2}$ é a frequência de Nyquist e α é o fator de *roll-off*.

Depois de filtrar o sinal em banda base, fazemos a modulação em quadratura multiplicando as amostras em fase e em quadratura por $\text{sen}(2\pi f_0 t)$ e $\cos(2\pi f_0 t)$ respectivamente, sendo f_0 a frequência da portadora. Os dois sinais resultantes devem ser somados para obtermos o sinal de saída do modulador.

Capítulo 3

Projeto do Simulador DVB-S2

3.1 Introdução

Agora que já lemos sobre o DVB-S2 e conhecemos melhor esse sistema, vamos discutir neste capítulo como construir um simulador de redes, capaz de simular todas as camadas OSI, incluindo esse sistema de transmissão de sinais de televisão. Vamos apresentar as opções de ferramentas disponíveis para a construção desse simulador, quais delas foram as escolhidas e como foi feita a integração dos softwares utilizados.

3.2 Visão Geral do Simulador

Esta dissertação se propõe a desenvolver um simulador capaz de testar todas as camadas OSI de uma transmissão de pacotes IP através do sistema de televisão digital via satélite DVB-S2. A primeira questão a ser discutida é como representar o DVB-S2 nas camadas do modelo OSI. Como vimos no Capítulo 2, o sistema pode receber como entrada diversos tipos de dados, encapsula esses dados em quadros, aplica códigos corretores de erros a eles, mapeia os bits em símbolos de uma das três constelações e gera o sinal RF modulado a ser enviado através do meio físico. Analisando essas funções, podemos implementar o DVB-S2 nas camadas de enlace e física, como mostra a Figura 3.1. A camada de enlace é responsável por tratar erros de transmissão, controlar o fluxo de dados e controlar o acesso ao meio. A camada física é responsável por preparar o sinal para ser enviado por um meio físico.

Depois de analisar o sistema e como vamos implementá-lo dentro do modelo OSI, temos que definir as ferramentas que vamos utilizar para implementar o simulador. Optamos

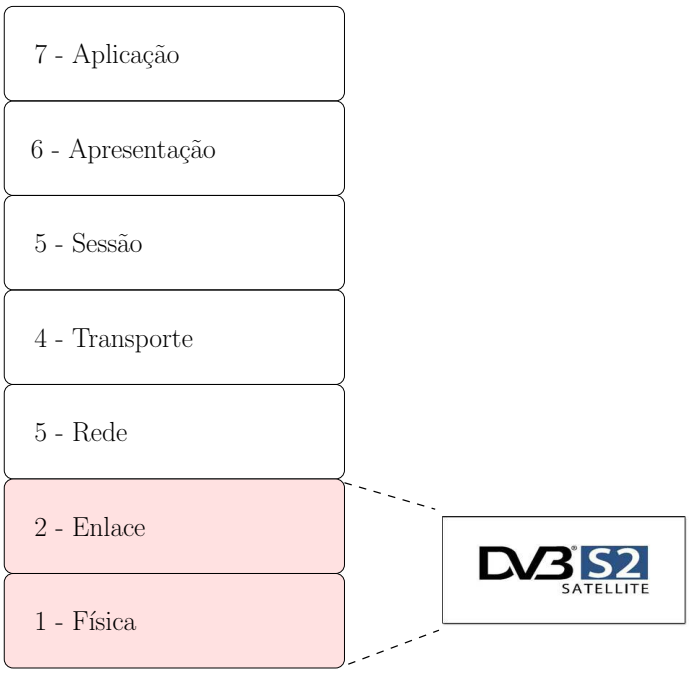


Figura 3.1: Localização do sistema DVB-S2 dentro da arquitetura OSI.

por utilizar dois softwares, o Matlab para fazer toda a parte física relacionada ao simulador DVB-S2 e o NS-2 para simular a parte lógica da transmissão de pacotes IP. Para tornar o simulador um produto que fosse de fácil utilização, estudamos uma forma de fazer a conexão entre esses programas para que eles trabalhassem conjuntamente e de forma transparente para o usuário.

3.3 NS-2

O Network Simulator Version 2 é um simulador de evento discreto popular na área de pesquisa em redes devido a sua flexibilidade e modularidade. Uma característica importante desse simulador é que ele é distribuido gratuitamente e com código fonte aberto, isso possibilita que seus usuários possam fazer os ajustes que acharem necessários e criar, além dos diversos módulos já existentes, outros novos módulos para complementar o ambiente de simulação.

O grupo que criou e mantinha o NS-2 achou que o programa precisava ser melhorado em alguns aspectos e criou um sucessor para o simulador, o NS-3. Esta nova versão do programa possui algumas novidades em relação a versão anterior, por exemplo a linguagem de programação utilizada. O NS-2 utiliza duas linguagens de programação. Seu

core é implementado em C++, uma linguagem compilada e robusta para processar uma grande quantidade de informação, e os comandos para as simulações são escritos em OTcl, uma linguagem interpretável que permite que o usuário faça modificações rápidas nas suas simulações sem a necessidade de ficar compilando todo o código do programa. No NS-3 é utilizado apenas o C++, pois o grupo achou que dessa maneira é mais fácil "*debuggar*" o programa. Outras tantas modificações foram feitas na nova versão com intuito de melhorar, porém a versão mais utilizada continua sendo a anterior. Apesar das duas versões serem bastante complexas, o NS-2 é mais bem documentado que seu sucessor, tendo mais livros, *blogs*, projetos e outros tipos de informação disponíveis a seu respeito, e possui mais usuários experientes. Dessa maneira é mais provável encontrar soluções para as dificuldades que encontramos ao fazer um novo módulo para o NS-2 do que para o NS-3.

Uma outra preocupação que tivemos ao criar o simulador do DVB-S2 para ser incluído nesse simulador de redes, foi fazer isso de uma maneira que fosse transparente para o usuário, de fácil utilização e que não alterasse a distribuição oficial do NS-2. Para criar um simulador exatamente dessa maneira, utilizamos o AutoTools para criar uma biblioteca dinâmica que contém o módulo que implementa o DVB-S2. Assim, precisamos apenas adicionar alguns comandos a mais no script que modela a simulação para incluir os módulos do DVB-S2 e seus parâmetros e, no momento que a simulação for executada, a biblioteca é carregada e as funções do DVB-S2 são chamadas.

Para implementar a biblioteca dinâmica para o DVB-S2, utilizamos como referência o projeto NS-MIRACLE [13], um pacote que permite a criação de bibliotecas dinâmicas para implementar novos módulos. Podemos ver em [14] vários trabalhos realizados com o NS-MIRACLE, como por exemplo simulações para teste de protocolos de transmissão submarina, de transmissão de vídeo e outros mais. No Anexo A, encontramos em detalhes como fazer uma biblioteca dinâmica para o NS-2 segundo [13].

3.3.1 Módulo para satélite

Para conectar o que foi implementado em Matlab com o código do NS-2, precisamos primeiro estudar a estrutura que já existe no simulador de redes para que saibamos como podemos usar o código feito para simular a parte física. Como queremos simular a rede do DVB-S2 que é um padrão para transmissão via satélite, teremos que usar enlaces estabelecidos por um satélite. No NS-2 encontramos esse tipo de enlace e utilizamos este módulo de

satélite como base para construir o simulador para este sistema. Antes de explicar como foi construído o simulador para DVB-S2, precisamos entender como funciona este módulo.

O NS-2 tem dois tipos de satélites geoestacionários que podem ser modelados, o "geo" que modela as camadas de uma forma mais completa e o "geo-repeater" que é um mero repetidor modelado como um nó de satélite degenerado, pois seu enlace possui apenas a camada física. O enlace usado como base para criar o simulador foi o do "geo", que está representado pela Figura 3.2. Cada bloco dessa figura representa o objeto de uma classe do enlace e o NS-2 nos permite configurar as classes do enlace (LL de *Link Layer*), da fila (*Queue*), da camada de acesso ao meio (MAC de *Medium Access Control*), da camada física (Phy) e também outros valores relacionados ao enlace. Os objetos da classe *SatNode* que modelam os nós de satélite são os mesmos que modelam os nós dos terminais terrestres, porém se distinguem na posição, no gerênciamento de *handoff* e nos objetos conectados em seu enlace.

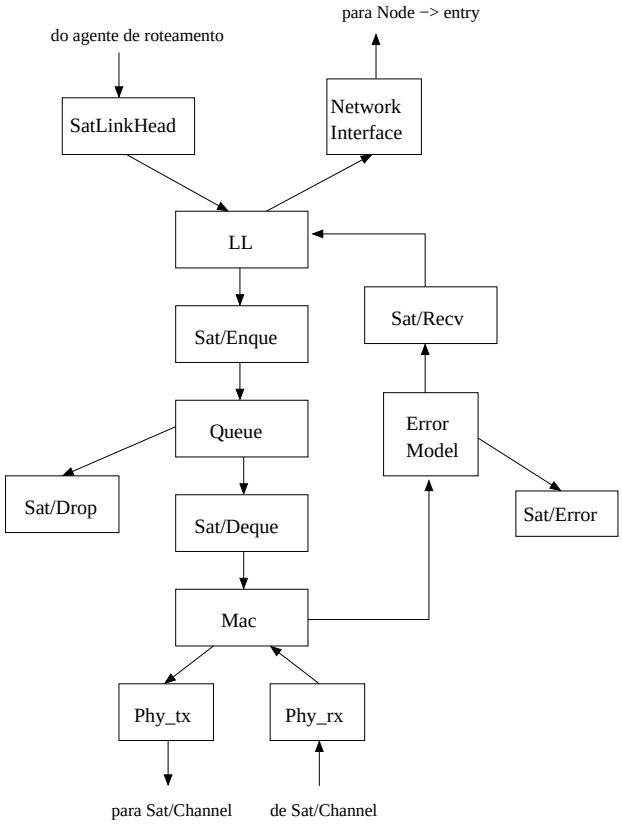


Figura 3.2: Enlace de satélite do NS-2.

Quando um pacote é enviado pelo satélite, o ponto de entrada desse pacote transmitido é o objeto *SatLinkHead* que possui ponteiro para os outros objetos seguintes. O próximo

objeto no percurso do pacote é o LL, onde os protocolos da camada de enlace são definidos e o endereço MAC é atribuido. A camada seguinte é a MAC, o objeto que controla essa parte se comunica com o objeto que implementa a fila e assim controla o envio de pacotes para a próxima camada. É este objeto que calcula também o tempo de transmissão do pacote. Depois temos a camada física implementada pelo objeto *Phy_tx* que envia o pacote para o canal. O objeto *SatChannel* que implementa o canal não faz qualquer tipo de processamento no pacote, apenas transmite uma cópia dele para a camada física do nó receptor, que é o *Phy_rx*.

Na recepção do pacote em um nó, notamos algumas diferenças. Ainda observando a Figura 3.2, podemos ver que o pacote que foi transmitido pelo objeto do canal *SatChannel* é recebido pelo objeto *Phy_rx*. O pacote então é enviado para a camada MAC, onde o objeto que a implementa mantém o pacote pelo tempo que durar sua transmissão, que foi calculado pelo objeto equivalente no nó transmissor. Em seguida o pacote é enviado para o objeto *Error Model*, o qual é definido por uma distribuição escolhida pelo usuário. Segundo essa distribuição, o objeto decide se o pacote foi recebido com sucesso ou foi recebido com erro e por isso é perdido. No caso de falha na recepção, o pacote é passado para o objeto *Sat/Error* e a transmissão desse pacote termina, no caso de sucesso, o pacote é enviado para *Sat/Recv*. Considerando que o pacote tenha sido recebido sem erros, o próximo objeto a receber o pacote é o LL, que repassa para as camadas superiores depois de processar o atraso, caso este tenha um valor diferente de zero.

Como analisamos anteriormente, as melhores camadas para alocar o DVB-S2 são de enlace e física, mas implementamos o sistema todo apenas na camada de enlace para tornar o programa menos complexo. A classe que implementa a camada de enlace do satélite tem duas funções que foram modificadas para atender nossas necessidades, a função *SendDown* e a função *SendUp*. Na Figura 3.3 podemos ver como estas funções estabelecem os enlaces de satélite no simulador.

A função *SendDown* estabelece a conexão da camada de enlace com as camadas inferiores, ou seja, transmite os pacotes do nó para o canal. Como o padrão DVB-S2 descreve apenas a conexão *downlink*, do satélite para os terminais, nesta camada implementamos o processamento feito pelo sistema nos pacotes transmitidos pelo satélite. Caso estivéssemos utilizando o canal de retorno via satélite padronizado pelo DVB-RCS (*Digital Video Broadcasting - Return Channel via Satellite*), este seria implementado pela função *SendDown* que

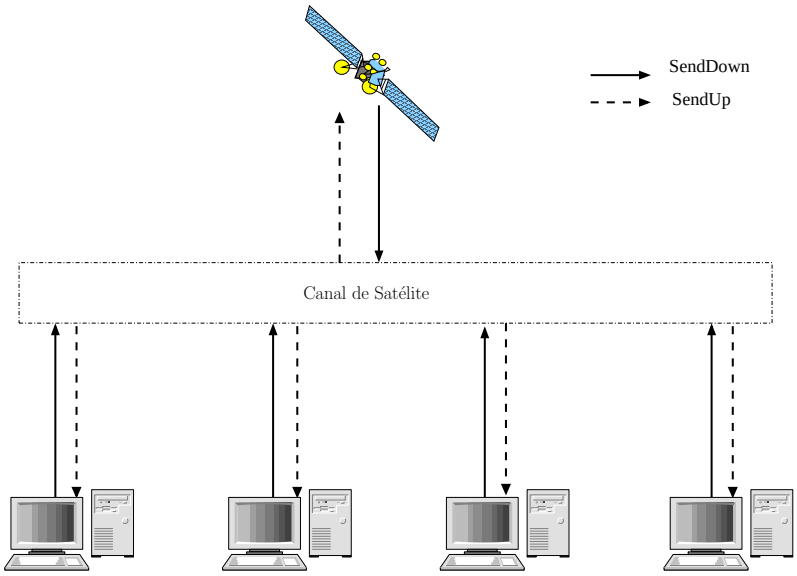


Figura 3.3: Exemplo de um cenário de simulação.

é chamada pelo nó terminal para enviar dados para o satélite.

A função *SendUp* estabelece a conexão das camadas inferiores para a camada de enlace, ou seja, transmite os pacotes do canal para o nó, exatamente a direção inversa da função *SendDown*. Nesta função implementamos o processamento realizado nos pacotes recebidos nos terminais, realizando os procedimentos necessários para obter a informação transmitida pelo satélite.

3.3.2 Geração de Tráfego de Dados

A implementação do *data payload*, como é chamado o conjunto de dados carregados por um pacote IP, no NS-2 é diferente do que acontece nas redes de computadores reais. Quando um usuário da rede pretende enviar dados, estes são transformados em bits, encapsulados em pacotes e então enviados até o receptor que irá transformar esses bits novamente nos dados originais, supondo que estes tenham sido transmitidos sem sofrer erros. Numa simulação feita no NS-2, raramente os dados que se deseja transmitir são armazenado no pacote como *data payload*, o programa mantém as informações sobre o usuário e sobre os dados que ele quer transmitir no cabeçalho do pacote e envia apenas este cabeçalho para o receptor, sem carregar com ele o *data payload* Apesar de normalmente os dados não serem transmitidos nas simulações, o NS-2 oferece uma opção do pacote carregar o *data payload* na transmissão [15].

Como desejamos estudar o processamento de sinais na camada física implementada no nosso simulador do padrão DVB-S2, precisamos gerar dados aleatórios para enviar nos pacotes transmitidos. Sendo assim, criamos uma classe chamada *RandomPayload* que é derivada da classe do NS-2 *AppData* e esta nova classe gera dados aleatórios que são adicionados aos pacotes criados pelo simulador.

Como já foi dito, não implementamos o canal de retorno via satélite que é padronizado pela norma do DVB-RCS [11]. Assim sendo, utilizamos apenas pacotes UDP para testar o simulador, pois se fizéssemos uso de pacotes TCP, não teríamos como fazer um teste completo, pois a transmissão de pacotes TCP requer um canal de retorno por este ser um protocolo orientado à conexão, ao contrário do UDP. Dessa maneira, decidimos fazer uma nova classe *RandomDataCBRTraffic* que deriva da classe *TrafficGenerator* pertecente ao NS-2. Esta classe do simulador de redes implementa um tráfego CBR que transmite pacotes UDP e a nova classe adiciona, dentre outras novas funções, a geração de pacotes com dados aleatórios. As outras funcionalidades da *RandomDataCBRTraffic* serão detalhadas na Sessão 3.5.

3.3.3 Canal de Satélite

O canal de satélite implementado pela classe *SatChannel* não realiza qualquer tipo de processamento no pacote recebido por ela, pois as simulações no NS-2 não costumam trabalhar com pacotes contendo *data payload*. Os pacotes que trafegam no simulador possuem apenas o cabeçalho, como foi visto anteriormente. O canal no NS-2 apenas faz uma cópia do pacote e entrega esta para o nó receptor.

Para estudar o processamento realizado na transmissão, precisamos de um canal que modifique os dados que os pacotes carregam neste simulador do sistema DVB-S2 que implementamos. Vimos na Sessão 3.3.2 que foi criada a classe *RandomPayload* que gera dados aleatórios que são adicionados aos pacotes. Então implementamos uma classe *DvbsChannel* que irá modificar estes pacotes simulando um canal. Esta nova classe chama o Matlab, como veremos na Sessão 3.5, para executar as funções do canal sobre o pacote transmitido.

O canal de satélite no NS-2 implementado pela classe *SatChannel* executa algumas funções que são essenciais para a simulação, como por exemplo passar o pacote para a próxima camada do enlace. Para não termos que refazer todo esse código, chamamos esta classe já existente no NS-2 dentro da nova classe *DvbsChannel* e assim aproveitamos essas funções que são essenciais para o funcionamento do simulador.

3.3.4 Recepção dos Pacotes

A Figura 3.4 mostra o cabeçalho de um quadro UDP e nele podemos ver os diversos campos que levam informações específicas sobre ele. Podemos observar que existe um campo que se chama *checksum*, com este campo o receptor é capaz de verificar a integridade do quadro. Se este campo for usado para proteger o cabeçalho de erros, então é feita uma aritmética de complemento a um e o resultado é colocado nos 16 bits deste campo. Caso *checksum* não seja usado, este campo é preenchido com zeros.

bits	1 – 8	9 – 16	17 – 24	25 – 32
0	Endereço IP de Origem			
32	Endereço IP de Destino			
64	Zeros	Protocolo	Tamanho UDP	
96	Porta de Origem		Porta de Destino	
128	Tamanho		*Checksum*	
160+	Dados			

Figura 3.4: Cabeçalho de um pacote UDP.

No NS-2, o objeto que é responsável por determinar se o pacote é recebido com sucesso ou não é o *Sat/Error*. Este objeto usa como base uma distribuição, que pode ser escolhida pelo usuário, para determinar se o pacote foi recebido ou não. Como os pacotes criados na simulação não carregam dados e não é realizado nenhum tipo de processamento neles, então é usado este método para determinar a recepção dos pacotes.

No simulador do DVB-S2, utilizamos este campo do cabeçalho para verificar a integridade do mesmo e assim determinamos se um pacote será aceito ou não. Se o cabeçalho do quadro estiver corrompido e o *checksum* acusar erro, então o pacote será descartado, como descreve [16]. Caso a verificação do *checksum* não falhe, então o pacote é recebido e transmitido para as camadas superiores.

3.4 Matlab

3.4.1 Conectando o Matlab a um Programa em C++

Os programas em Matlab são implementados em uma linguagem interpretável própria e o NS-2 foi implementado em C++. Então tivemos que pesquisar a melhor maneira de um programa escrito em C++ chamar e executar as funções que realizam o processamento estabelecido pelo padrão DVB-S2 no Matlab.

Uma opção que podemos usar no simulador é o *Matlab Coder*. Este produto do Matlab tem várias funcionalidades, entre elas temos a transformação de funções escritas para Matlab em bibliotecas estáticas ou executáveis em C e C++ que podem ser executadas sem a necessidade de uma biblioteca específica do Matlab. Com o *Matlab Coder*, também podemos fazer a tradução do algoritmo desenvolvido no Matlab para as linguagens C e C++ gerando códigos legíveis e portáveis. Essa segunda funcionalidade seria uma boa opção para o simulador se este produto aceitasse todas as funções do Matlab, o que não acontece na realidade. O *Matlab Coder* aceita apenas algumas funções simples do Matlab e não tem suporte para diversas funções das *toolboxes* de Sistemas de Comunicação e de Processamento de Sinais. Outra desvantagem é que o código em C++ fica muito complexo e isso não é bom caso tenhamos que fazer ajustes para adaptar as funções geradas a um programa já existente, que é o nosso caso.

Para resolver a comunicação entre o NS-2 e o Matlab, podemos usar o *Matlab Engine*. Este produto abre a partir de um programa em C ou C++ uma sessão do Matlab. A biblioteca do *Engine* possui funções que permitem que um algoritmo em C ou C++ abra e feche um processo que executa o Matlab, envie e receba dados da sessão do Matlab e execute comandos como se estivesse usando-os em um processo convencional do Matlab. Uma desvantagem desse produto é a necessidade do usuário do simulador ter o Matlab instalado em sua máquina. Mas a grande vantagem é que as funções implementadas no Matlab para a parte física do simulador podem ser usadas sem precisar de nenhuma modificação e, uma vez aprendendo a usar as funções da biblioteca do *Matlab Engine*, a manipulação das funções do Matlab se torna mais fácil para o programador.

Ao usar o *Matlab Engine*, vamos precisar usar a biblioteca *Matrix C/C++* para nos auxiliar a manipular as variáveis dentro de uma sessão. Em um programa escrito em C++ utilizamos variavéis que carregam tipos de dados bem conhecidos como *char*, *int* e outros,

mas as variáveis no Matlab possuem uma estrutura diferente e esta biblioteca nos ajuda a manipular essas variáveis. No Apêndice B, há mais detalhes sobre o *Matlab Engine* e a biblioteca *Matrix C/C++*.

3.4.2 Estrutura do Simulador DVB-S2 no Matlab

Toda a parte de processamento de sinais realizada pelo padrão DVB-S2 e pelo canal foi implementada no Matlab e a estrutura do simulador está representada pelo diagrama de blocos da Figura 3.5. Cada bloco desse diagrama possui o nome de uma das funções que executa uma parte do simulador. As setas tracejadas na figura indicam o envio de variáveis de configuração da simulação para algumas funções e as setas contínuas indicam a direção do tráfego de dados. Podemos observar também que no *ReceiveDvbs2* temos algumas setas contínuas com traços duplos, isso acontece porque nas simulações feitas apenas no Matlab consideramos dois tipos de receptor, um que conhece previamente o tamanho dos pacotes que recebe e um outro que não conhece esse tamanho e obtém essa informação através dos cabeçalhos dos pacotes recebidos.

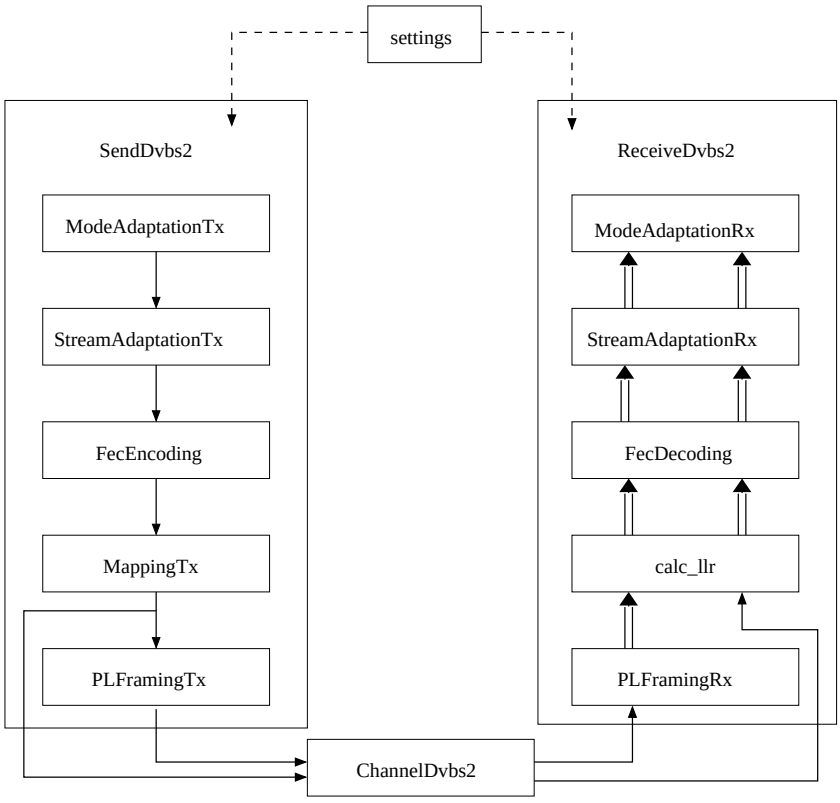


Figura 3.5: Diagrama de blocos do simulador DVB-S2.

A divisão do transmissor e do receptor do DVB-S2 foi implementada segundo a divisão em subsistemas que foi apresentada no Capítulo 2 e todas as funcionalidades deste padrão são realizadas na simulação, exceto o último *scrambling* apresentado na Sessão 2.7.4 e a modulação em banda base e em quadratura da Sessão 2.8. Os trabalhos publicados que implementam este padrão são capazes de simular até o subsistema de mapeamento nas constelações, enquanto que o simulador apresentado nesta dissertação realiza o processamento dos dados até o subsistema de Formação de Quadros da Camada Física inclusive. Com este simulador somos capazes de estudar a transmissão de quadros normais e curtos, outra vantagem se comparado a outros simuladores do DVB-S2, uma vez que encontramos resultados apenas para transmissão de quadros normais.

Voltando à Figura 3.5, observamos que dados de dois pontos diferentes são enviados para o canal. O primeiro local onde os dados são coletados é logo após o *MappingTx*, aí temos os XFECFRAMES que são enviados para o canal e depois recebidos no receptor por *calc_llr*. A partir deste ponto, os pacotes receberão o processamento de sinal necessário para que o nó receptor receba a informação original, se possível. Simulando dessa maneira, podemos comparar o nosso simulador com o simulador de DVB-S2 de outros pesquisadores. O outro ponto onde os dados são coletados é na saída do *PLFramingTx*, temos os PLFRAMES que são transmitidos pelo canal e então enviados para *PLFramingRx*, a partir daí receberão também o processamento de sinal imposto pelo receptor. Utilizaremos os resultados dessa simulação que envia PLFRAMES para comparar com a simulação da transmissão dos FECFRAMES.

As funções que compõem o simulador do DVB-S2 foram implementadas utilizando principalmente funções dos *toolboxes* de Sistemas de Comunicação e de Processamento de Sinais. A primeira função a ser executada é a *settings*, ela é responsável por gerar as matrizes, vetores e objetos que são essenciais para a simulação segundo as configurações passadas pelo usuário, como os objetos que determinam os códigos corretores de erro, vetores que carregam informação sobre as modulações usadas e outros dados importantes.

A primeira função que executa o processamento descrito pelo padrão DVB-S2 é a *ModeAdaptationTx*. Ela calcula o CRC-8 dos dados que o usuário deseja enviar e gera o cabeçalho de banda base com as informações referente aos dados, como foi mostrado na Figura 2.3 da Sessão 2.3.6. Sua função equivalente no receptor, a *ModeAdaptationRx*, verifica os bits de paridade do CRC-8 para saber se a informação foi corrompida.

A próxima função a ser executada é a *StreamAdaptationTx* e ela adiciona o preen-

chimento (ou *padding*) caso seja necessário e faz o primeiro embaralhamento dos bits (ou *scrambling*). A função *StreamAdaptationRx*, faz o processo inverso no receptor, desfaz o embaralhamento e retira o preenchimento, mas, para retirar os zeros adicionados pelo sistema para preencher o quadro, essa função precisa ler o cabeçalho para conseguir o tamanho do quadro que foi enviado.

Na sequência, temos a *FecEncoding* que processa a codificação de canal com os códigos BCH e LDPC e também realiza o *bit interleaving*, que é bem parecido com o embaralhamento de bits e está descrito na Sessão 2.5.3. O Matlab possui uma função chamada *dvbs2ldpc* que retorna a matriz de cheque de paridade para o código LDPC segundo a norma do DVB-S2 para uma taxa de código escolhida pelo usuário. Essa função retorna a matriz estabelecida pelo padrão apenas para os quadros normais, então modificamos esta função criando uma nova função nomeada de *dvbs2ldpc_complete* que retorna a matriz de cheque de paridade para os quadros curtos também. Dessa maneira, na função *settings*, geramos os objetos que fazem a codificação LDPC para os dois tipos de quadros usados pelo sistema e usamos estes objetos na *FecEncoding*. Na função análoga situada no receptor, a *FecDecoding*, fazemos o processo inverso. Desfazemos o *bit interleaving* e decodificando os dados com os objetos decodificadores gerados pela *settings*.

Em seguida, temos o mapeamento dos bits em símbolos de uma das três constelações estabelecidas na norma do DVB-S2, que são QPSK, 8PSK, 16APSK e 32APSK. Para executar estas modulações, utilizamos o algoritmo *DVBS2Constellation*, escrito por Bernhard Schmidt, que está disponível no site da *MathWorks* [17], empresa que criou o Matlab. Neste site há um local onde pesquisadores podem divulgar seus algoritmos feitos em Matlab ou baixar eles para usar em suas pesquisas. Alguns desses algoritmos que são muito procurados e são recomendados por funcionar bem, são depois integrados ao Matlab como funções oficiais do programa. No receptor, a demodulação foi feita utilizando um algoritmo LLR (*Log Likelihood Ratio*), implementado na função *calc_llr* [18] que também foi escrita pelo mesmo autor da função *DVBS2Constellation*.

A função que implementa o último subsistema do DVB-S2 é a *PLFramingTx*. Esta função gera o quadro PLFRAME, como mostra a Figura 2.13 da Sessão 2.7. Ela cria o cabeçalho desse quadro, modula-o com $\pi/2$-BPSK e adiciona os símbolos piloto no quadro conforme o padrão do DVB-S2 estabelece. É importante lembrar que parte do cabeçalho do PLFRAME é codificado com Reed-Müller, cuja codificação é simplesmente uma multiplicação

da mensagem com a matriz geradora do código. No receptor temos a *PLFramingRx*, que faz o processo inverso da *PLFramingTx*. Primeiro o receptor executa a demodulação $\pi/2$-BPSK do cabeçalho e então decodifica parte do cabeçalho para obter as informações sobre a modulação, a taxa do código LDPC e o uso de pilotos. Para implementar a decodificação de uma palavra codificada com Reed-Müller, usamos a decodificação lógica majoritária. Usamos como base uma função implementada por Muthiah Annamalai, que estão implementadas em Octave e podem ser baixadas em [19] e [20]. Esta função implementa a decodificação para o código Reed-Müller segundo o algoritmo descrito em [21] e, no nosso simulador, implementamos um decodificador para o caso específico do cabeçalho do PLFRAME estabelecido pelo padrão DVB-S2.

3.5 NS-2 e Matlab

Agora que já conhecemos a estrutura do simulador do DVB-S2 no NS-2 e no Matlab, vamos ver os detalhes dessa conexão entre as funções dos dois programas.

A transmissão de quadros no NS-2 começa com a geração do tráfego de dados, que no nosso simulador é executado pela classe *RandomDataCBRTraffic*. A classe mãe dela que é a *TrafficGenerator*, possui algumas variáveis de configuração que permite que o usuário escolha, por exemplo, o tamanho do pacote que será enviado. Na classe *RandomDataCBRTraffic*, adicionamos mais algumas variáveis permitindo a escolha da modulação, da SNR e do uso ou não de símbolos piloto. Quando a função que inicia o tráfego de dados é executada, ela chama a função do Matlab *settings* que gera todas matrizes e objetos necessários para a simular aquela transmissão. Como vimos na Sessão 3.3.2, os pacotes do NS-2 em geral não possuem dados. É na classe *RandomDataCBRTraffic* que chamamos a classe *RandomPayload* para gerar bits aleatórios que fazem papel do nosso payload na simulação.

Algumas considerações também são feitas na função *SendDown* da classe que implementa a camada MAC no NS-2. Nesta função criamos os quadros na sessão do Matlab que serão transmitidos e também os cabeçalhos com as informações contidas nos cabeçalhos dos pacotes do NS-2. A função *SendDown* chama a função *SendDvbs2* do Matlab que implementa todos os subsistemas do transmissor do DVB-S2. O quadro retornado por essa função é então recebido pelo NS-2 e guardado em uma estrutura relacionada a cada pacote do simulador pelo seu número de identificação.

As outras funções agregam menos tarefas que estas que mencionamos acima. A função *SendUp* chama a função *ReceiveDvbs2* do Matlab e esta executa todos os processos dos subsistemas do receptor do DVB-S2. A classe *DvbsChannel* implementa o canal que soma ao sinal um ruído branco gaussianoao sinal segundo uma razão sinal-ruído estabelecida pelo usuário criando um canal AWGN (*Additive White Gaussian Noise*), que é executado pela função *ChannelDvbs2* no Matlab. Por fim, temos a classe *DvbsErrorModel* que verifica se o pacote recebido está corrompido ou não com a ajuda do Matlab e então decide se passa o pacote para as camadas superiores ou se descarta ele.

Capítulo 4

Simulações e Resultados

4.1 Introdução

No Capítulo 3, foi apresentado o simulador, que é composto por dois *softwares*, o Matlab e NS-2. Para implementar este simulador, construímos no Matlab toda a parte física, que é composta por transmissor e receptor, segundo o padrão DVB-S2, e por um canal. Este simulador da parte física, construído no Matlab, foi adicionado ao NS-2 para implementarmos um simulador completo para o padrão DVB-S2.

Como a implementação do simulador envolve duas etapas referentes a cada um dos *softwares* utilizados, fizemos dois tipos de simulação. Na Sessão 4.2, temos o resultado de simulações feitas utilizando somente o Matlab para testar a parte física do simulador. Com estas simulações vamos verificar se o sistema DVB-S2 implementado está de acordo com os resultados esperados pela norma, se há diferença nas curvas de BER para as transmissões de FECFRAMES e PLFRAMES e como é o comportamento do receptor para os casos de ele conhecer e não conhecer previamente os tamanhos dos pacotes recebidos.

Na Sessão 4.3, temos as simulações realizadas com o simulador completo, ou seja, com o Matlab e o NS-2 funcionando conjuntamente. Nestas simulações, podemos estudar como se comportam os quadros enviados pelo sistema DVB-S2 e calcular parâmetros da simulação de rede, como o *throughput* e a porcentagem de pacotes recebidos com sucesso.

4.2 Simulação I

Nesta simulação utilizamos a estrutura apresentada na Figura 3.5 do Capítulo 3. Utilizamos as quatro modulações disponíveis no padrão DVB-S2, transmitimos FECFRAMES e PLFRAMES normais e curtos, que são as saídas dos subsistemas *MappingTx* e *PLFramingTx*. Ainda consideramos dois tipos de receptores, um que não possui nenhuma informação prévia sobre o tamanho dos pacotes que está recebendo e outro receptor que possui esta informação.

Com esta simulação queremos verificar se o sistema DVB-S2 implementado está de acordo com os resultados apresentados pelo padrão. Para isso fizemos a transmissão de FECFAMES, para podermos fazer comparações com os valores apresentados na norma. Fizemos também transmissão com PLFRAMES para verificar se existe alguma alteração significativa se comparada à transmissão de FECFRAMES e se o código corretor de erros aplicado no cabeçalho do subsistema da camada física do DVB-S2 é suficientemente robusto para proteger a informação. Simulando com os dois tipos distintos de receptores, estudamos se é possível realizar transmissões sem que o receptor saiba o tamanho dos quadros recebidos ou se somente é possível enviar quadros cujos tamanhos já são conhecidos pelo receptor.

Os parâmetros utilizados na simulação estão na Tabela 4.1.

Parâmetros da Simulação	Valores
Modulações	QPSK, 8PSK, 16APSK e 32APSK
Tamanho do quadro	64800 (Normal) e 16200 (Curto)
Tipo de Entrada do Codificador LDPC	Bit
Tipo de Saída do Decodificador LDPC	*Information Part*
Tipo de Decisão do Decodificador LDPC	*Hard Decision*
Número de Iterações do Decodificador LDPC	50
Algoritmo LLR	LLR Aproximado

Tabela 4.1: Parâmetros da simulação.

4.2.1 Resultados

Os resultados dessa simulação estão separados por modulação e cada uma delas está separada por tipo de quadro transmitido, quadros normais formados por 64800 bits ou quadros curtos com 16200 bits. Também utilizamos nas simulações dois tipos de receptores, um que conhece previamente o tamanho dos quadros e outro que não possui essa informação.

Nas Figuras 4.1, 4.2, 4.5, 4.6, 4.9, 4.10, 4.13 e 4.14 temos os resultados para a trans-

missão de quadros normais usando as modulações QPSK, 8PSK, 16APSK e 32APSK respectivamente. Fazendo uma comparação dos resultados das transmissões usando FECFRAMES e receptores com conhecimento prévio do tamanho dos quadros com os apresentados pelo trabalho referênciado em [22], observamos que os valores encontrados em sua maioria estão próximos dos encontrados no trabalho apresentado em [22].

Comparando ainda os gráficos que apresentam as simulações de transmissão de quadros normais, podemos notar que há diferença entre os receptores com e sem conhecimento prévio do tamanho dos quadros. Nas SNRs mais baixas temos diferenças nos valores de BER, mas depois de um certo valor de SNR as curvas são muito parecidas. A diferença nas SNRs mais baixas se deve ao fato de como a decisão é feita na recepção do quadro. O receptor que não conhece os tamanhos dos pacotes recebidos, verifica os bits de paridade do campo CRC-8 para decidir se aquele quadro deve ser aceito ou não. Caso o quadro recebido não seja aceito, este é jogado fora e a BER é calculada como se o quadro inteiro estivesse errado, por isso temos valores de BER bem próximos de 100%. Ao comparamos as transmissões de FECFRAMES e PLFRAMES, não notamos diferenças entre as curvas BER, então o processamento realizado no subsistema de Formação de Quadros da Camada Física não tem influência significativa para a recepção dos dados.

As Figuras 4.3, 4.4, 4.7, 4.8, 4.11, 4.12, 4.15 e 4.16 apresentam os resultados para a transmissão de quadros curtos usando as modulações QPSK, 8PSK, 16APSK e 32APSK respectivamente. Comparando estes resultados com os aqueles das transmissões utilizando quadros normais. Estes resultados estão de acordo com o esperado, uma vez que o padrão do DVB-S2, referênciado em [1], informa que para quadros curtos há uma degradação de 0,2 a 0,3 dB comparado aos valores em que os quadros normais possuem uma performance quase livre de erros.

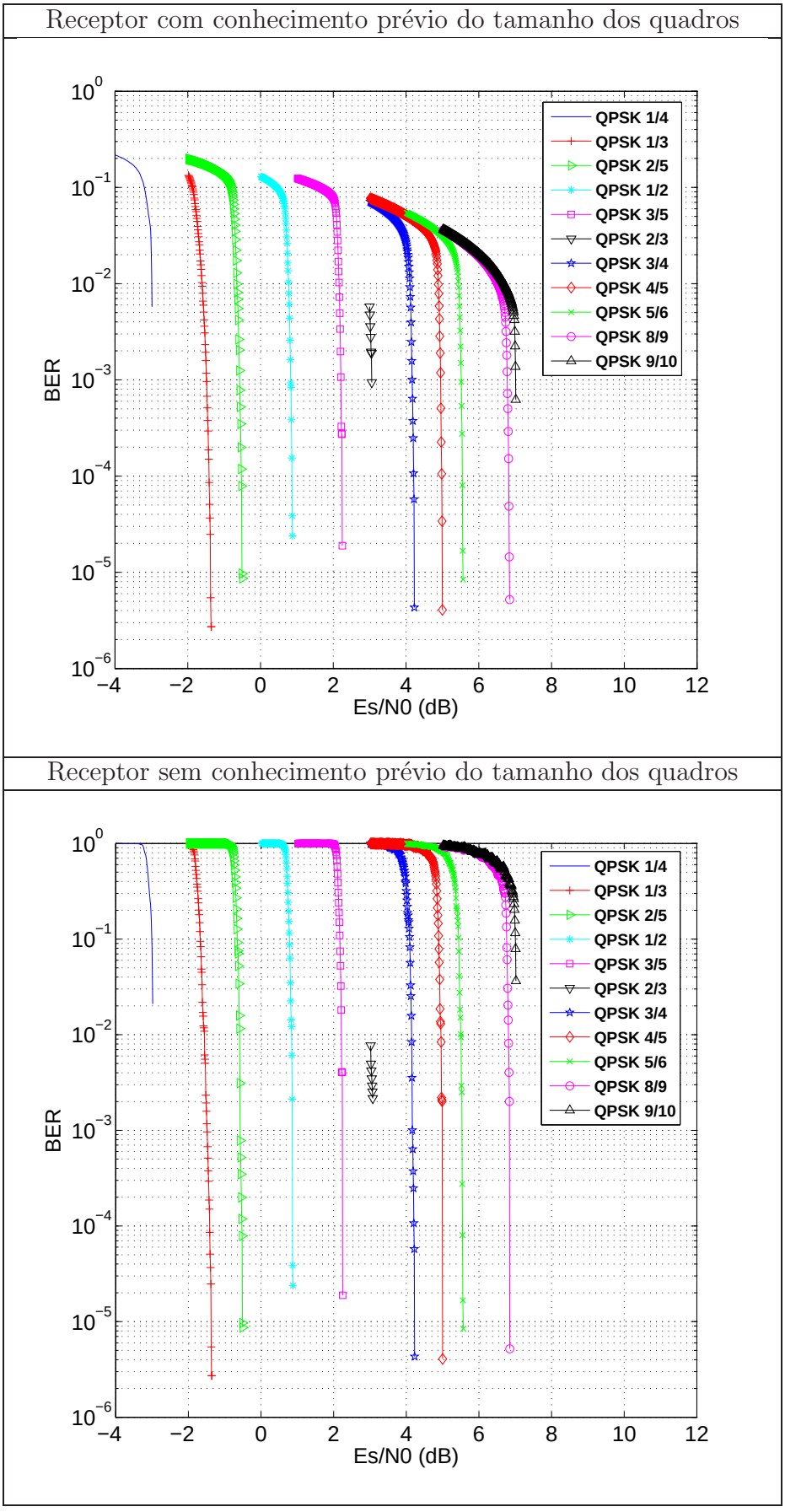


Figura 4.1: Transmissão de FECFAMES Normais com modulação QPSK.

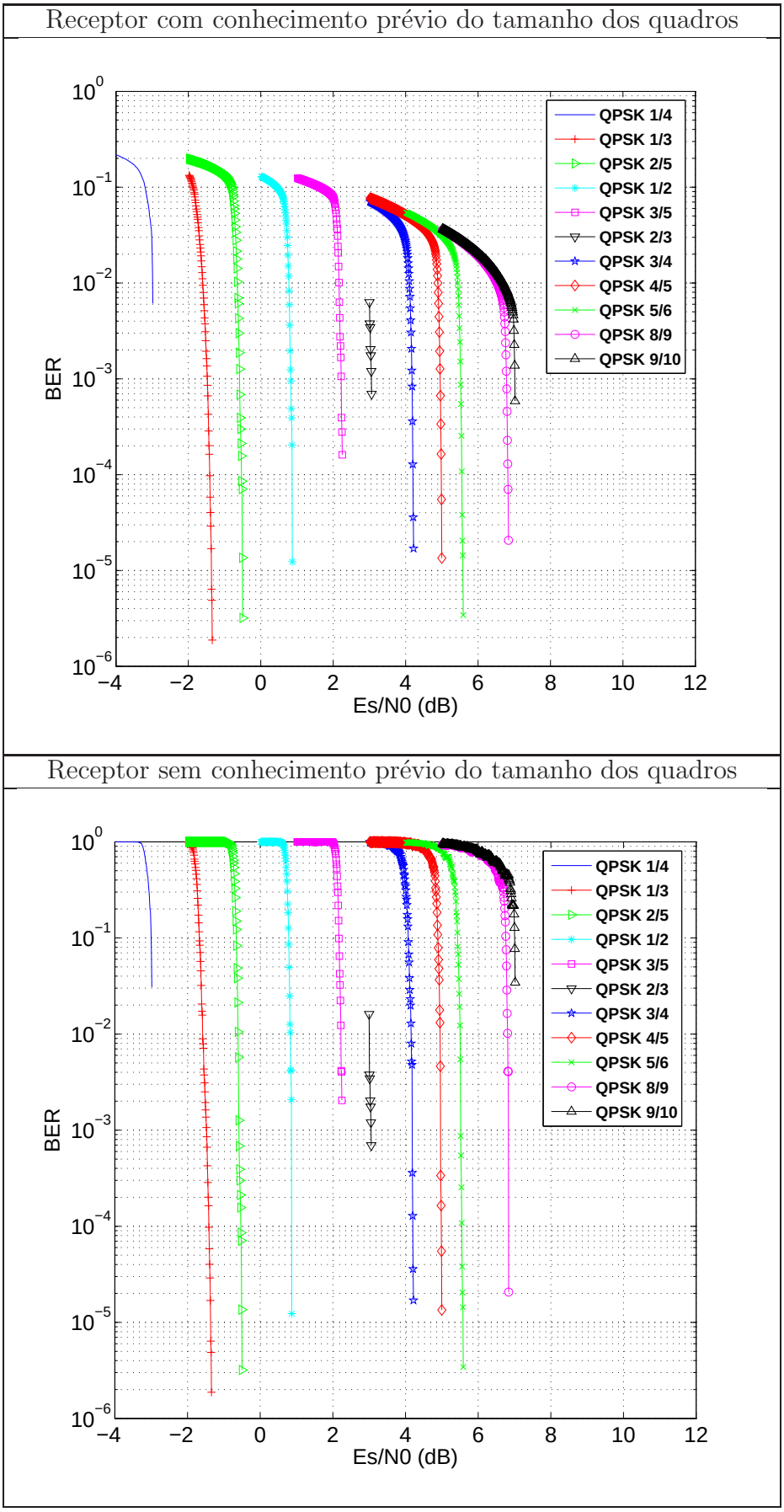
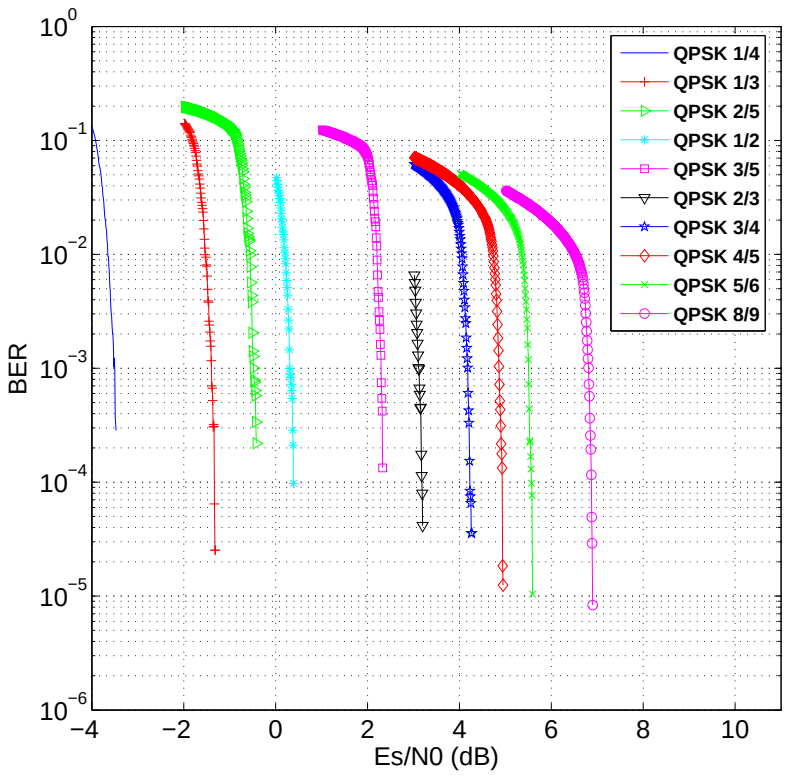


Figura 4.2: Transmissão de PLFRAMES Normais com modulação QPSK.

Receptor com conhecimento prévio do tamanho dos quadros



Receptor sem conhecimento prévio do tamanho dos quadros

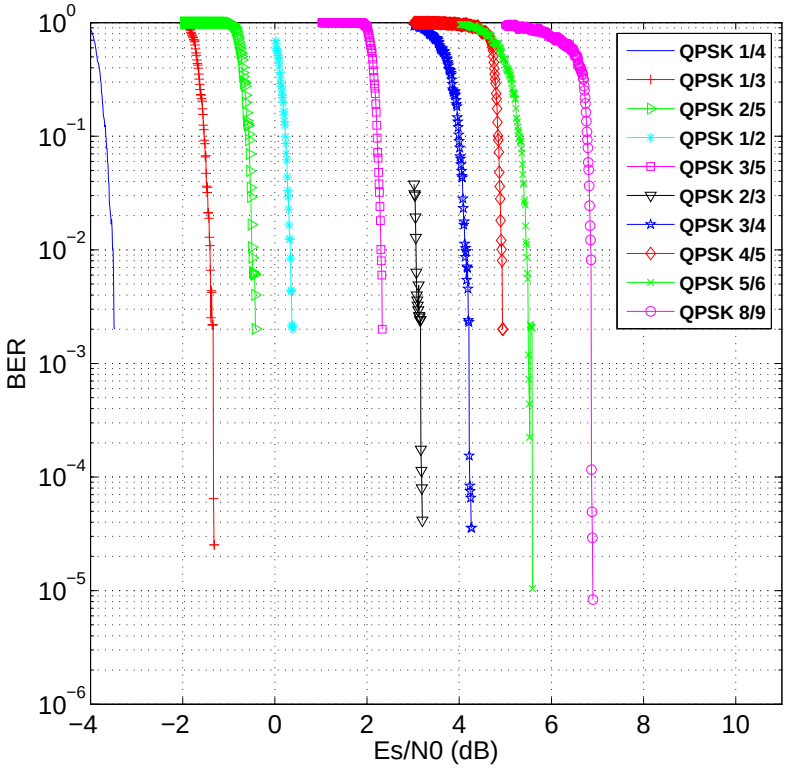
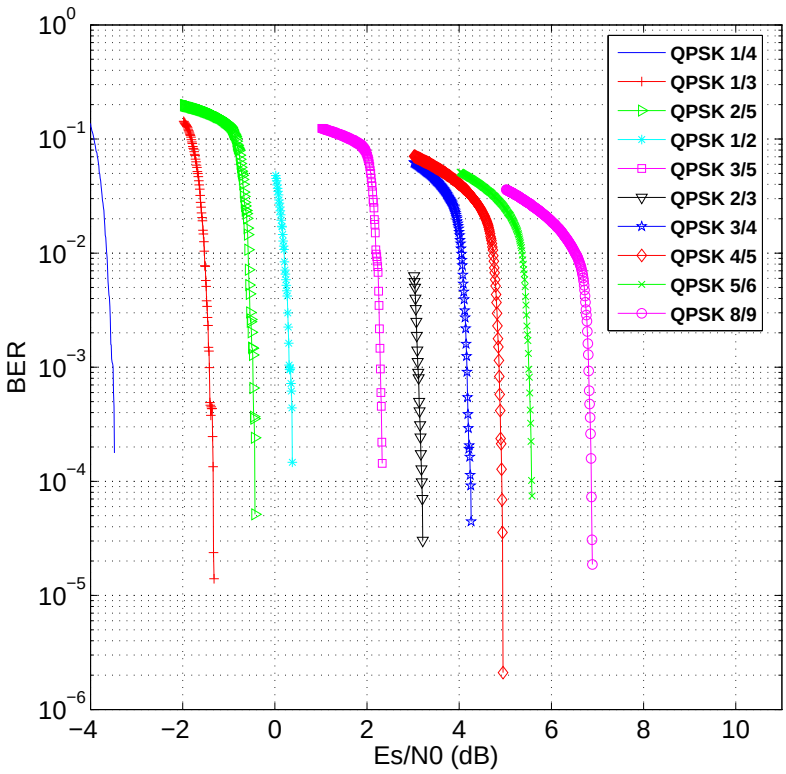


Figura 4.3: Transmissão de FECFRAMES Curtos com modulação QPSK.

Receptor com conhecimento prévio do tamanho dos quadros



Receptor sem conhecimento prévio do tamanho dos quadros

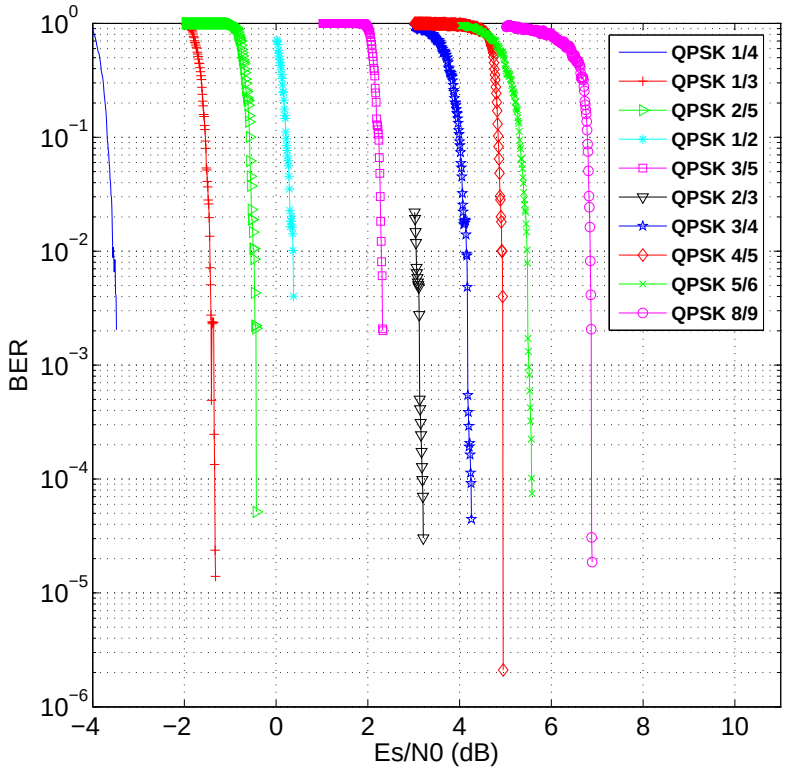


Figura 4.4: Transmissão de PLFRAMES Curtos com modulação QPSK.

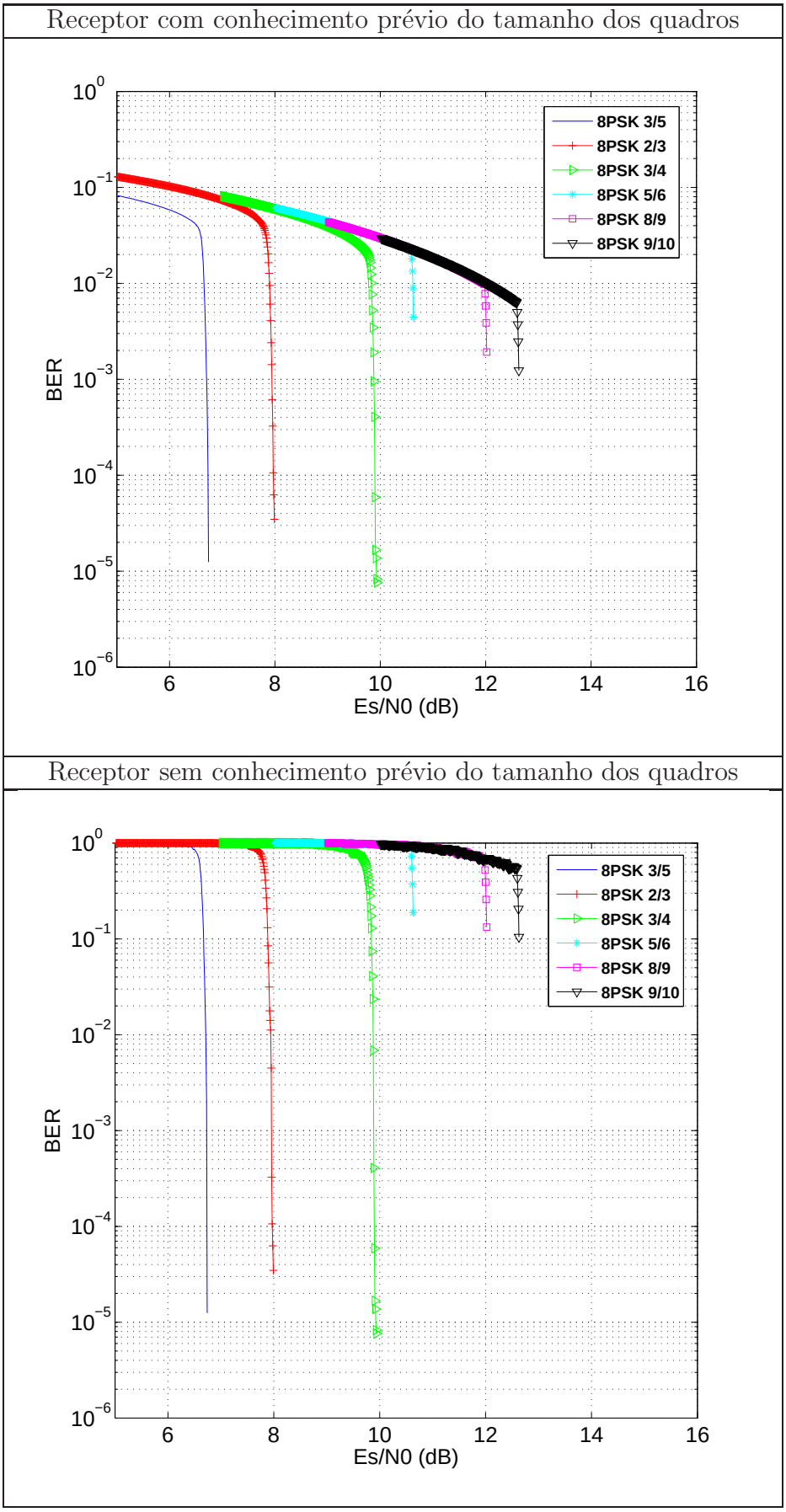


Figura 4.5: Transmissão de FECFRAMES Normais com modulação 8PSK.

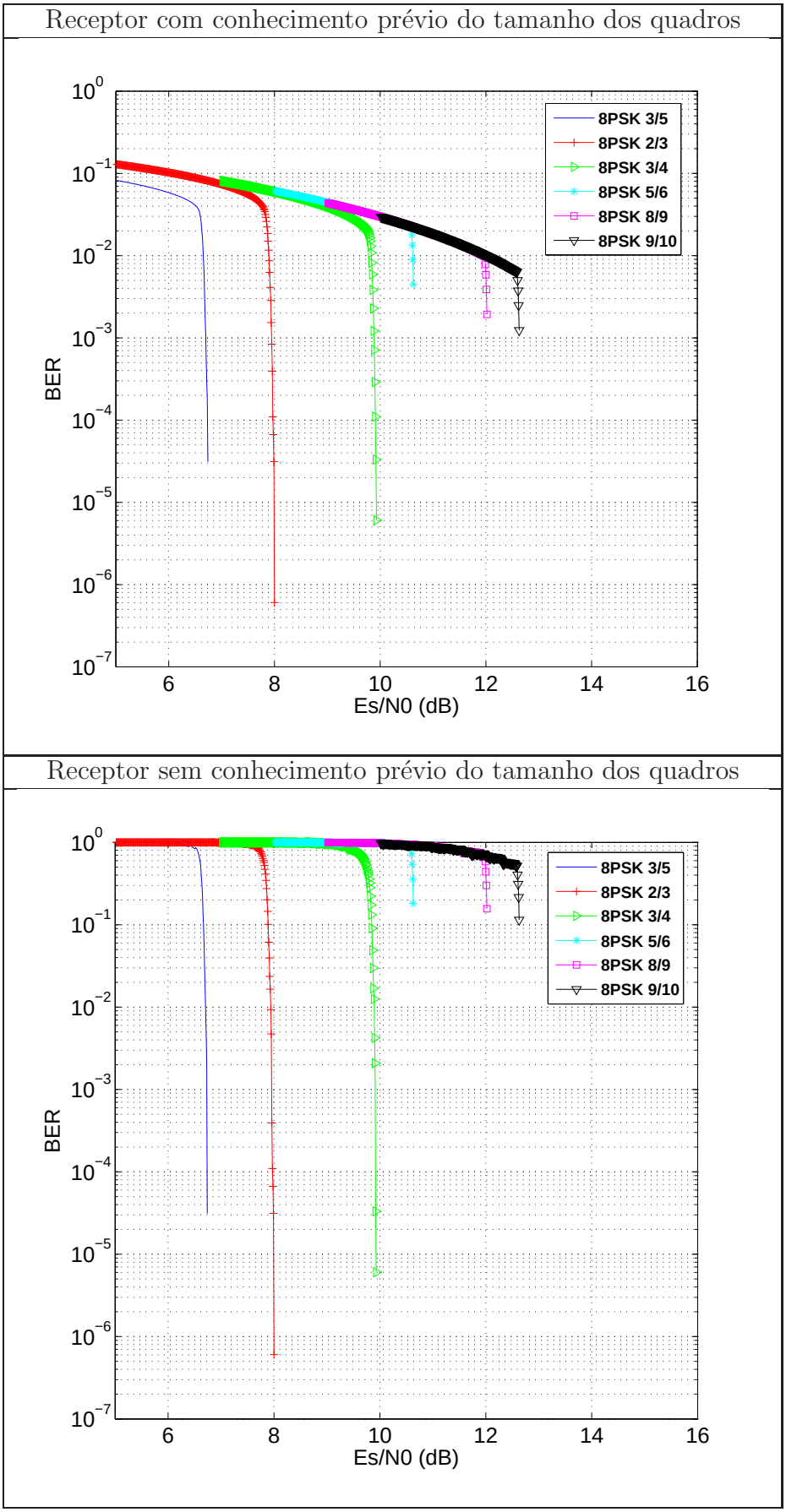


Figura 4.6: Transmissão de PLFRAMES Normais com modulação 8PSK.

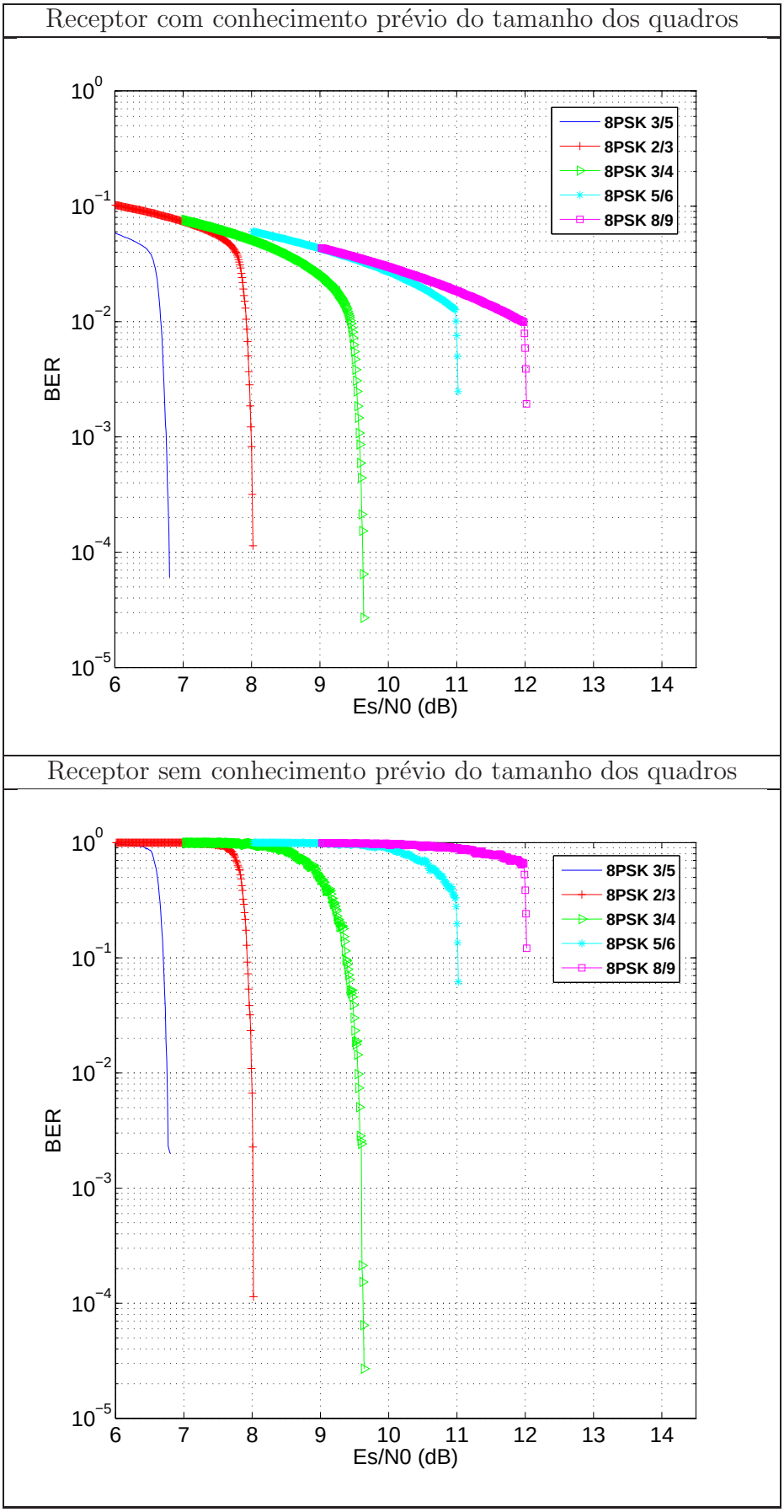


Figura 4.7: Transmissão de FECFRAMES Curtos com modulação 8PSK.

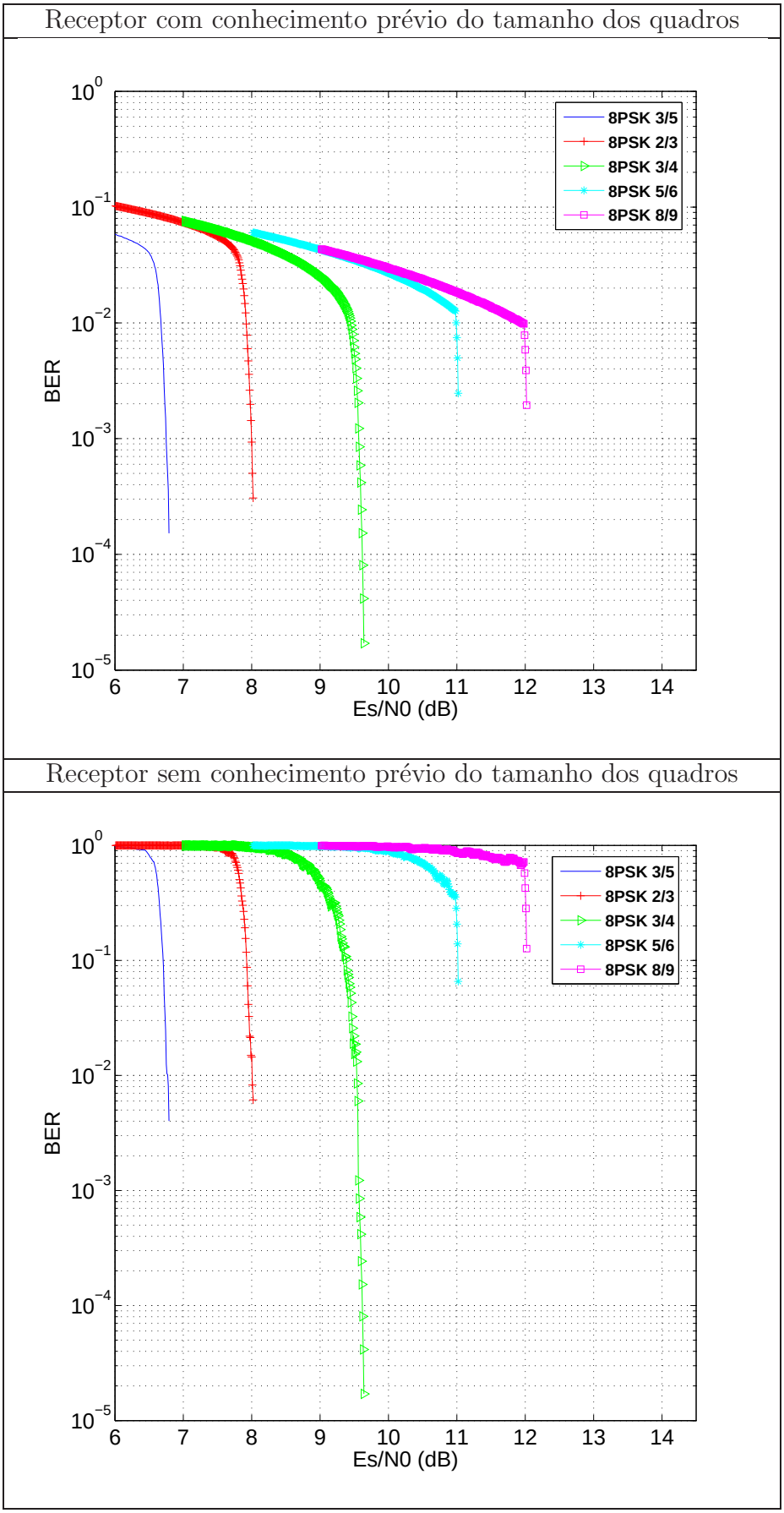


Figura 4.8: Transmissão de PLFRAMES Curtos com modulação 8PSK.

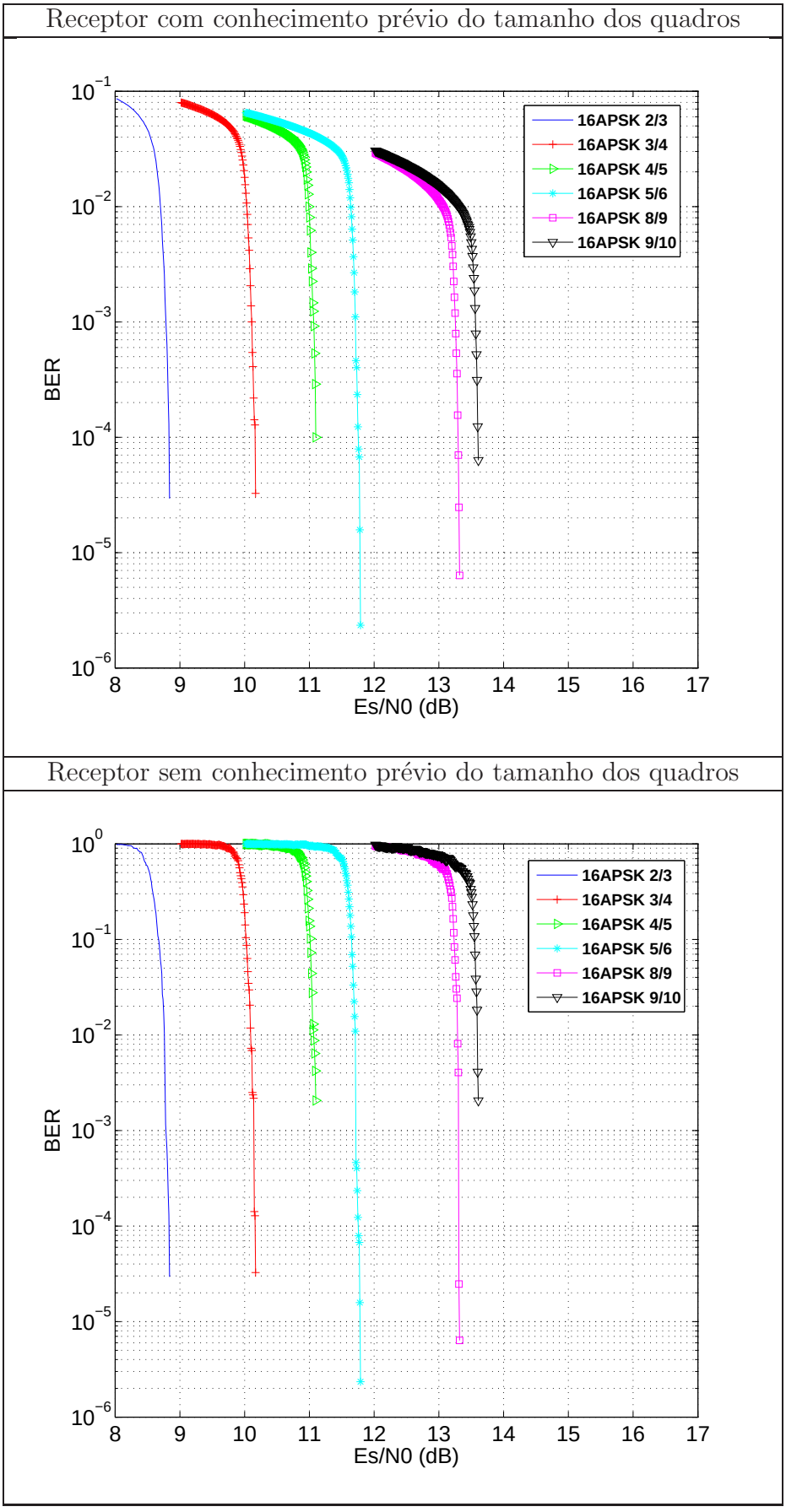


Figura 4.9: Transmissão de FECFRAMES Normais com modulação 16APSK.

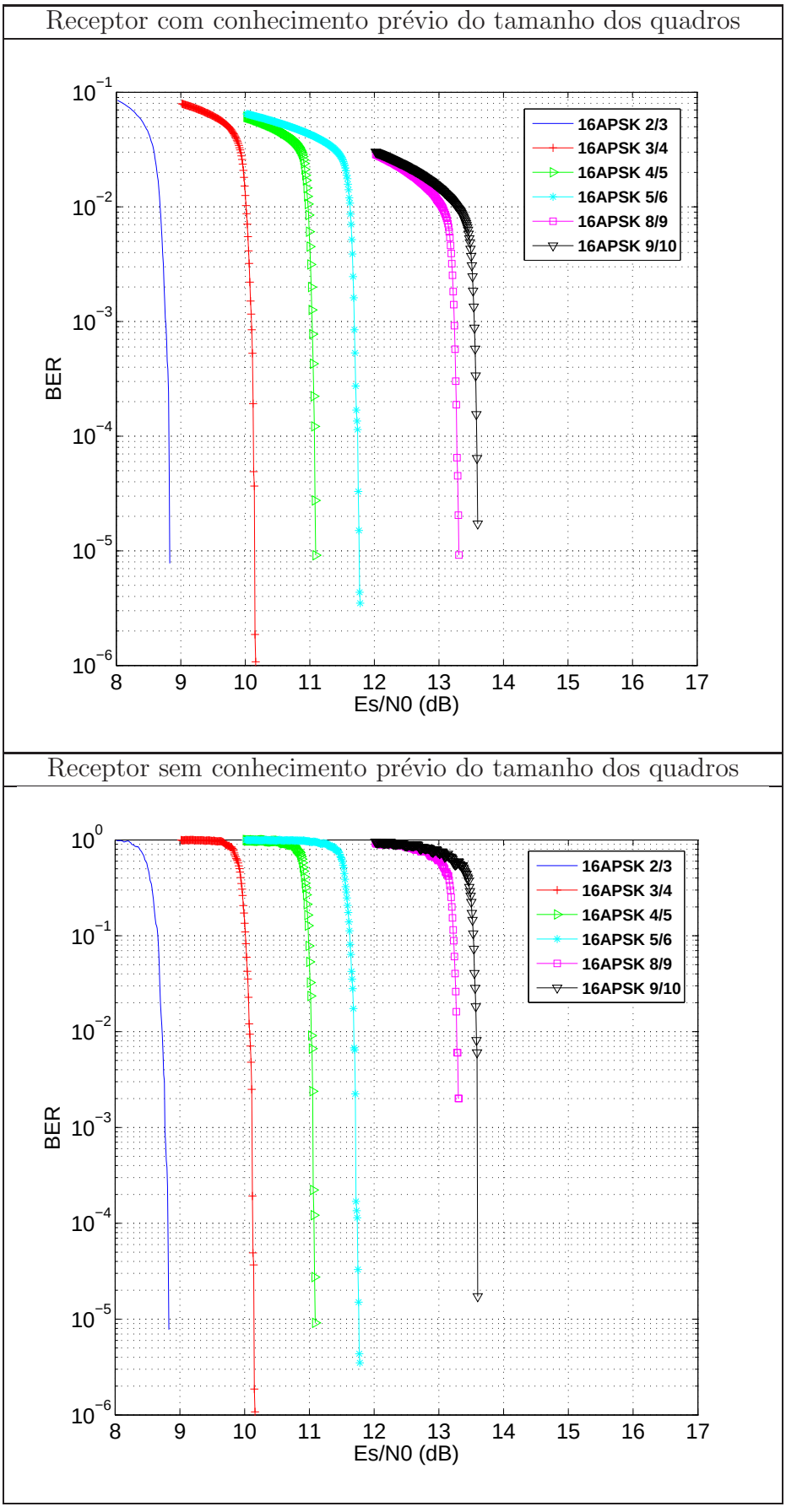


Figura 4.10: Transmissão de PLFRAMES Normais com modulação 16APSK.

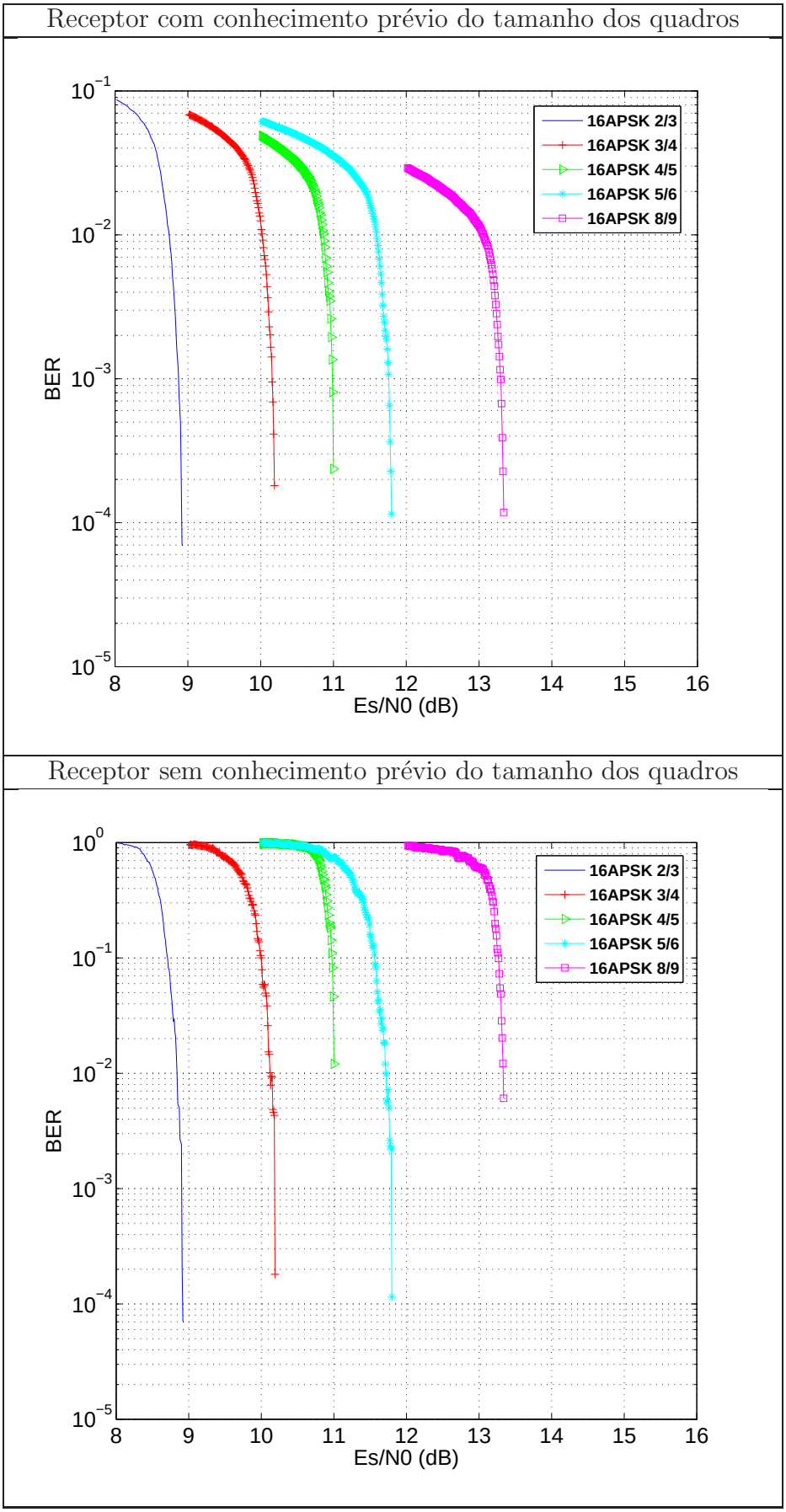


Figura 4.11: Transmissão de FECFRAMES Curtos com modulação 16APSK.

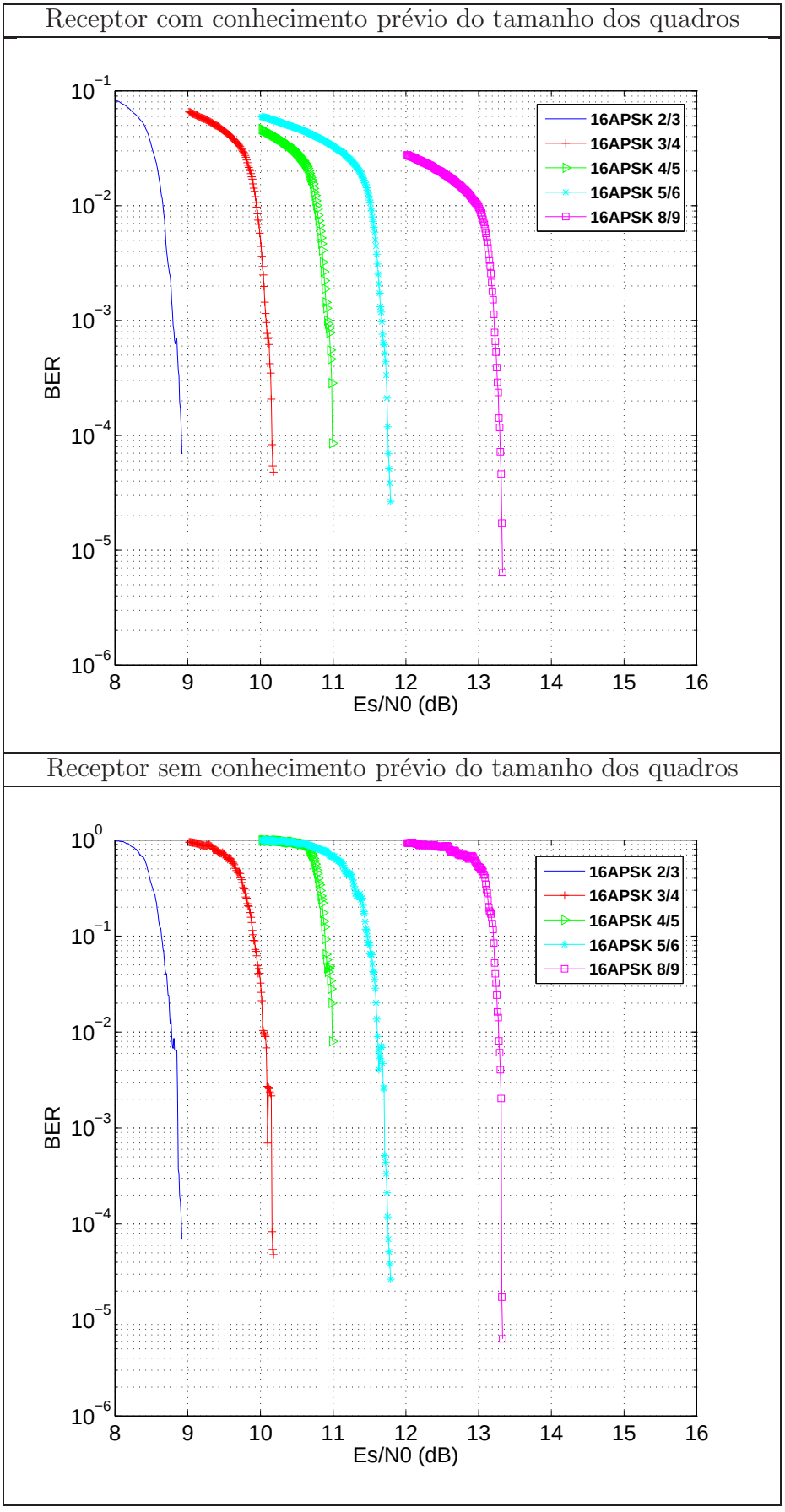


Figura 4.12: Transmissão de PLFRAMES Curtos com modulação 16APSK.

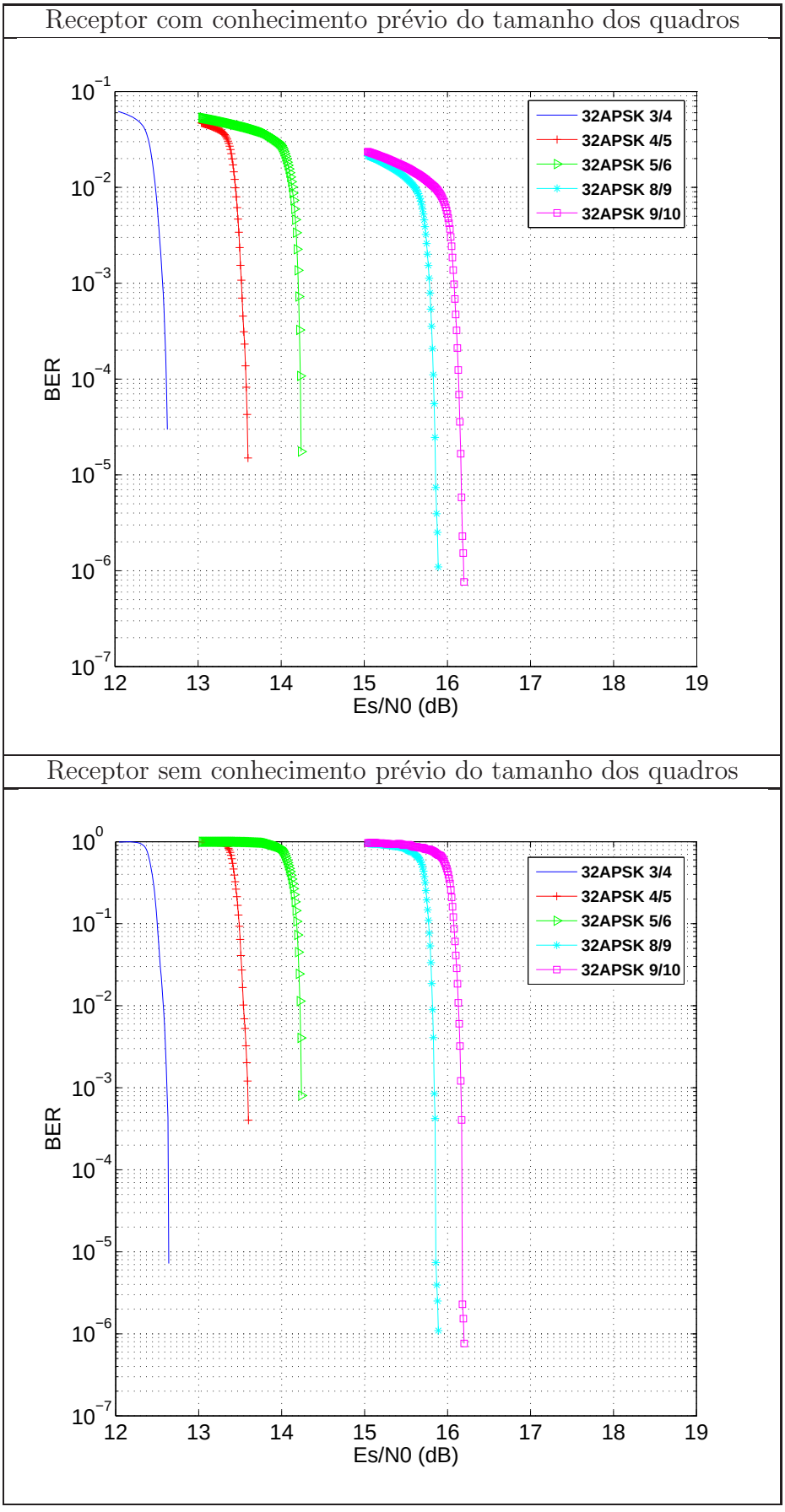


Figura 4.13: Transmissão de FECFRAMES Normais com modulação 32APSK.

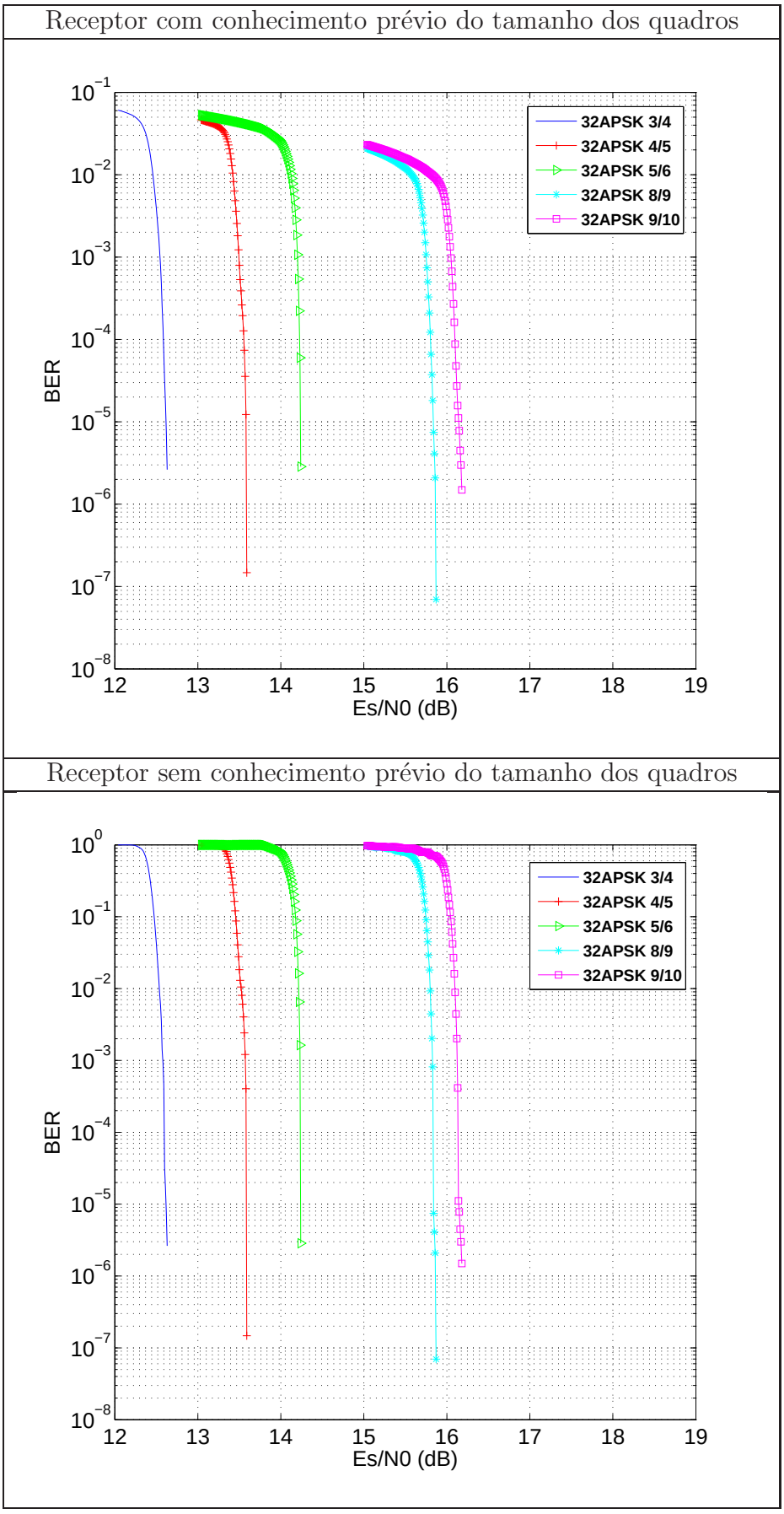


Figura 4.14: Transmissão de PLFRAMES Normais com modulação 32APSK.

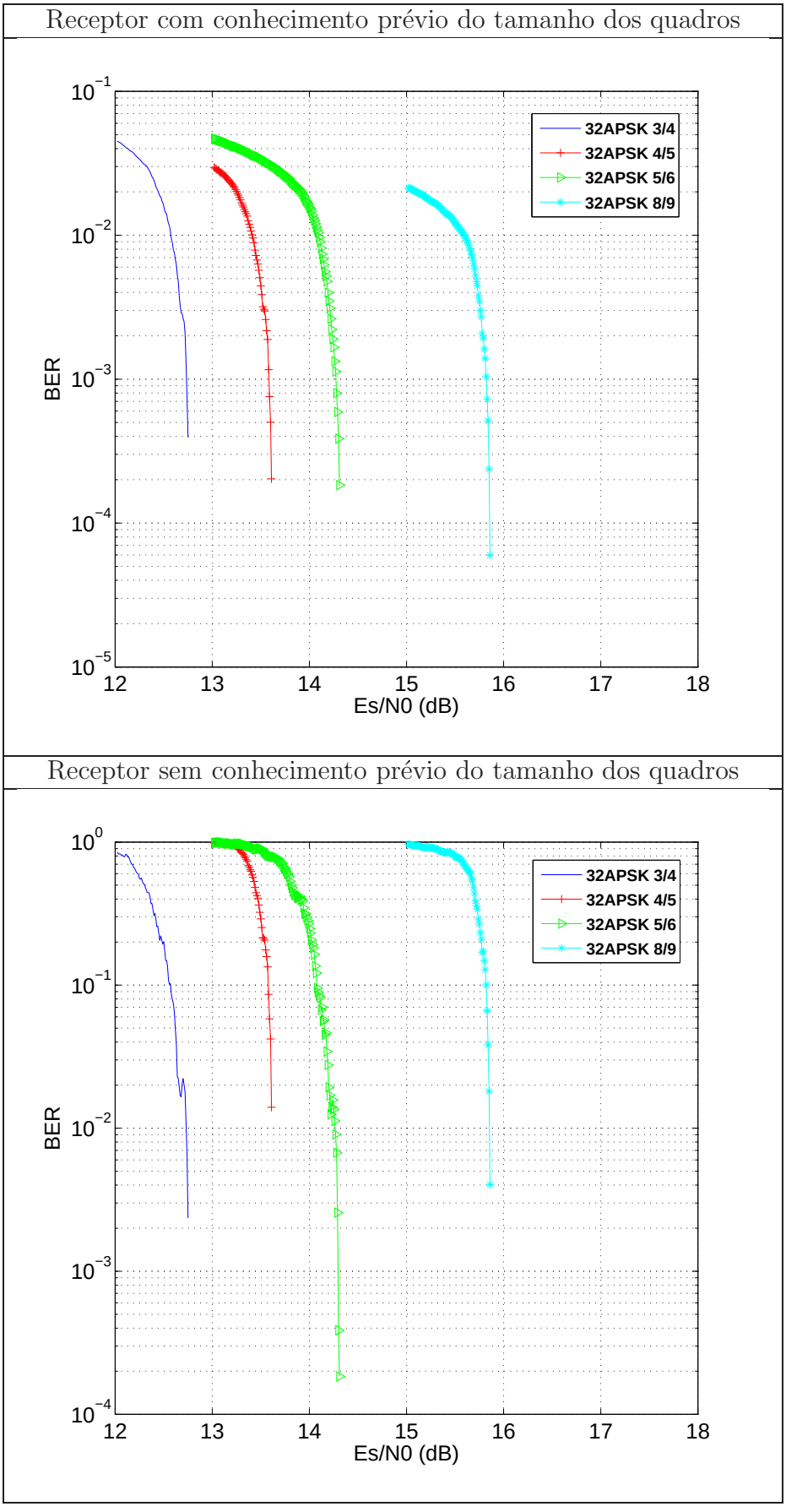


Figura 4.15: Transmissão de FECFRAMES Curtos com modulação 32APSK.

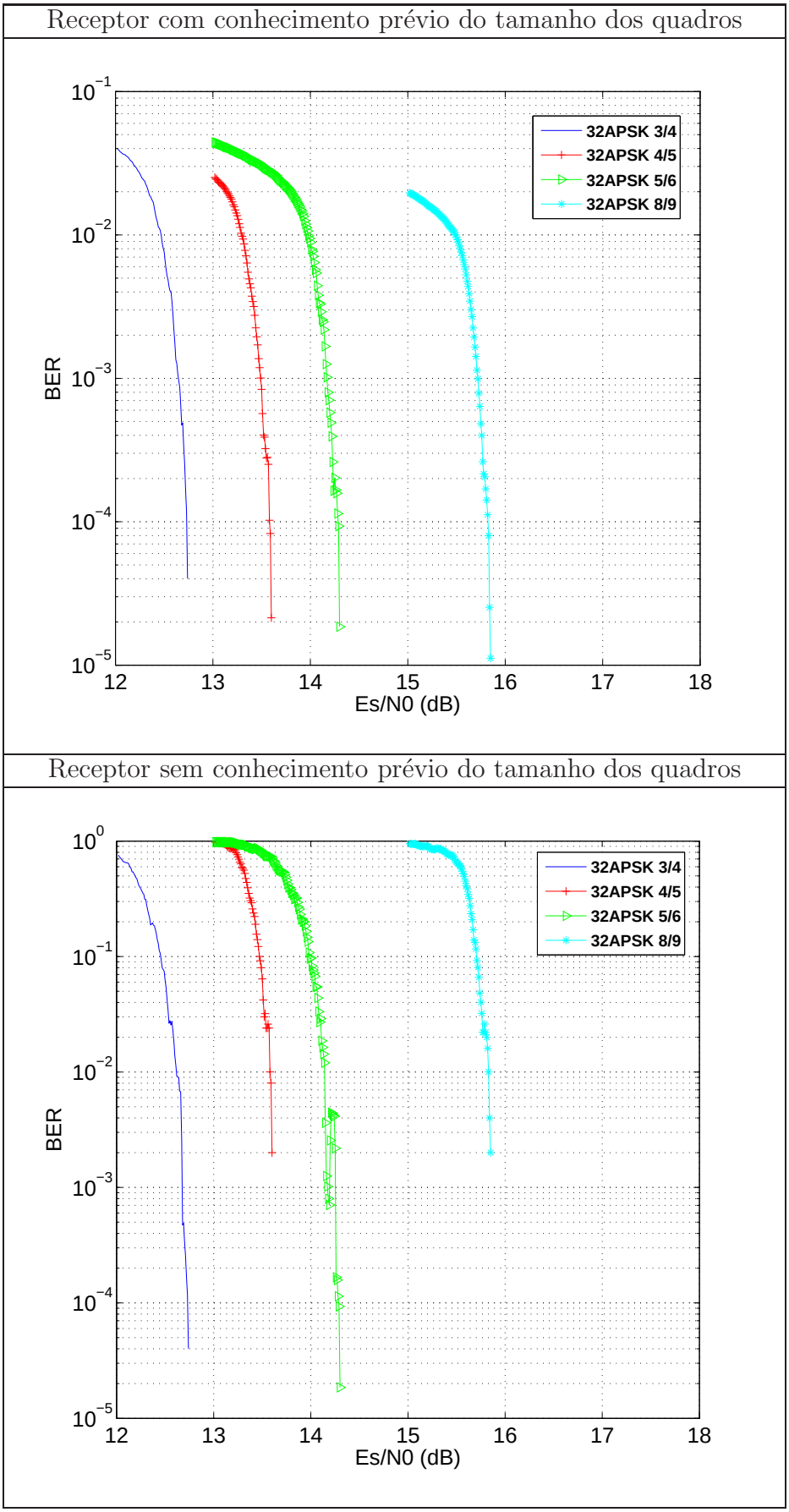


Figura 4.16: Transmissão de PLFRAMES Curtos com modulação 32APSK.

4.3 Simulação II

Nesta segunda simulação, testamos o simulador com o Matlab e o NS-2 funcionando conjuntamente. Na Seção 4.2, a simulação foi realizada para a transmissão de FECFRAMES e PLFRAMES e foram testados dois tipos de receptor, um que possui informações sobre o tamanho do pacote e outro que não possui essas informações. Nessa simulação foi testada apenas a transmissão de PLFRAMES e foi utilizado um receptor que não possui informações prévias sobre o pacote.

Para a simulação, utilizamos um fluxo de pacotes UDP, por não ser uma transmissão confiável, ou seja, ao contrário do TCP, o receptor não troca mensagens com o emissor para avisar sobre a recepção dos pacotes. Como não há canal de retorno no simulador, não temos como testar nessa simulação a troca de pacotes TCP.

Além dos códigos corretores de erro utilizados pelo padrão DVB-S2, mencionados na Seção 2.5, o simulador apresentado nessa dissertação usa o campo *checksum* do cabeçalho UDP para verificar a integridade da informação carregada pelo pacote, como está explicado em 3.3.4.

Com a análise dos resultados obtidos, esperamos verificar, através da observação do *throughput* da rede, se a transmissão de pacotes de Internet é viável.

4.4 Resultados

As Figuras 4.17, 4.19, 4.21, 4.23 apresentam o *throughput* para os seguintes pares de modulação e taxa respectivamente: QPSK 2/5, 8PSK 3/5, 16APSK 3/4 e 32APSK8/9. Os pares de modulação e taxa usados pelo sistema DVB-S2 são também conhecidos por ModCods por causa do campo do cabeçalho da camada física que leva o mesmo nome, como foi apresentado na Seção 2.7.2 e cujos valores podem ser encontrados na Tabela 2.12.

A simulação foi feita utilizando valores de Es/N0 escolhidos levando em consideração as curvas de BER apresentadas na Seção 4. Podemos observar que o *throughput* aumenta segundo o ModCod que foi utilizado para a transmissão, pois quanto maior a taxa do código LDPC, maior o tamanho do pacote que pode ser enviado.

As Figuras 4.18, 4.20, 4.22 e 4.24 representam as taxas de pacotes recebidos com sucesso. Comparando as curvas obtidas na Seção 4.2, pode-se concluir que o simulador de redes para o padrão DVB-S2 está funcionando conforme esperado.

O padrão que define o DVB-S2 permite o uso de 4 modulações para transmissão e 11 taxas de codificação LDPC, totalizando 44 possíveis combinações e, dessas opções, o padrão permite o uso 28 ModCods. Levando em consideração a complexidade dos terminais e o uso limitado de alguns desses ModCods, existem estudos que sugerem reduzir o número de pares de modulação e taxa para que o sistema tenha um melhor desempenho. Um desses estudos é apresentado em [23], no qual os autores sugerem que esse conjunto seja reduzido para 7 ou 5 ModCods. Comparando as curvas de BER, observamos que algumas delas estão muito próximas para alguns ModCods. Podemos testar esses casos no simulador de redes para o padrão DVB-S2, analisando essas curvas através de seus *throughputs*. Analisando um desses casos para os ModCods 16APSK 9/8 e 32APSK 4/5, observa-se que essas curvas de BER são muito próximas, em torno de 13.5 dB, como pode ser observado nas Figuras 4.10 e 4.14. Observando seus respectivos *throughputs*, que estão representados nas Figuras 4.25 e 4.27, verificamos que a transmissão utilizando 16APSK 9/8 alcança um *throughput* mais alto em uma relação sinal ruído mais baixa que a transmissão com 32APSK 4/5. Considerando que 16APSK é uma modulação mais robusta que a 32APSK e que ela alcança um *throughput* mais alto em um meio com mais ruído, a escolha da primeira modulação é a melhor opção entre os dois ModCods.

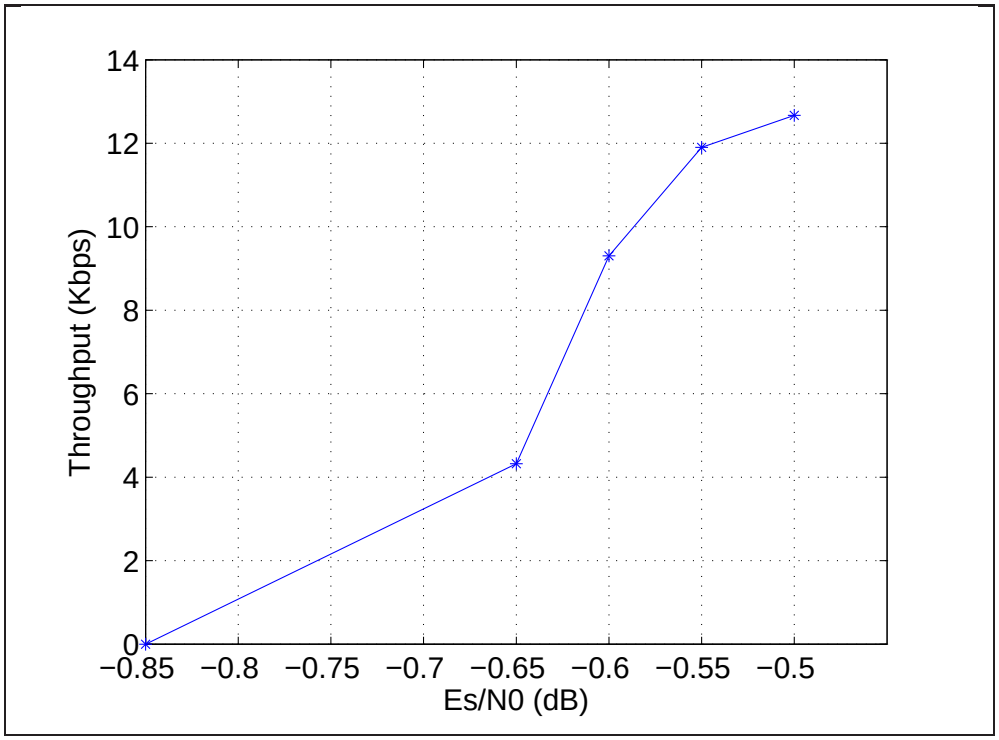


Figura 4.17: *Throughput* da rede para modulação QPSK 2/5.

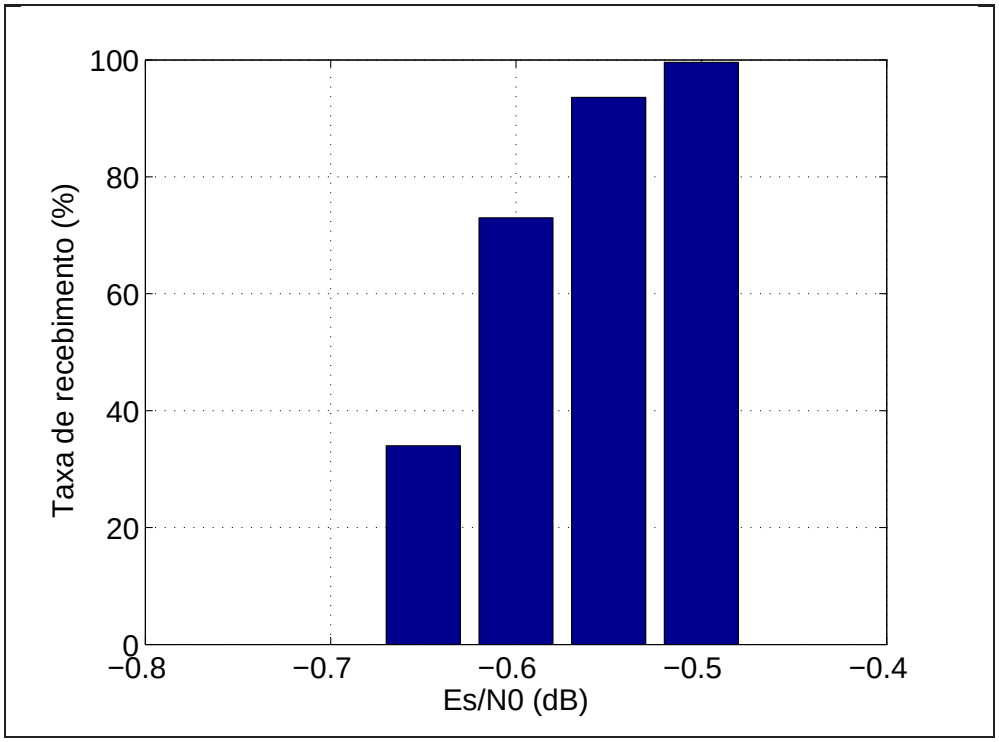


Figura 4.18: Quantidade de Pacotes recebidos com sucesso para modulação QPSK 2/5.

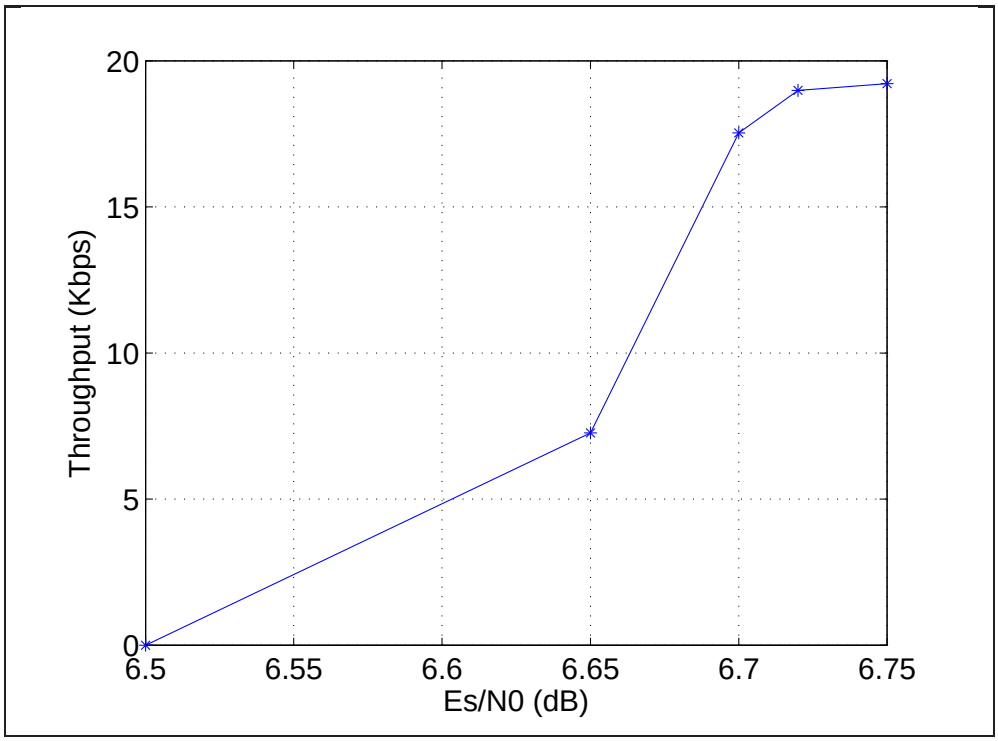


Figura 4.19: *Throughput* da rede para modulação 8PSK 3/5.

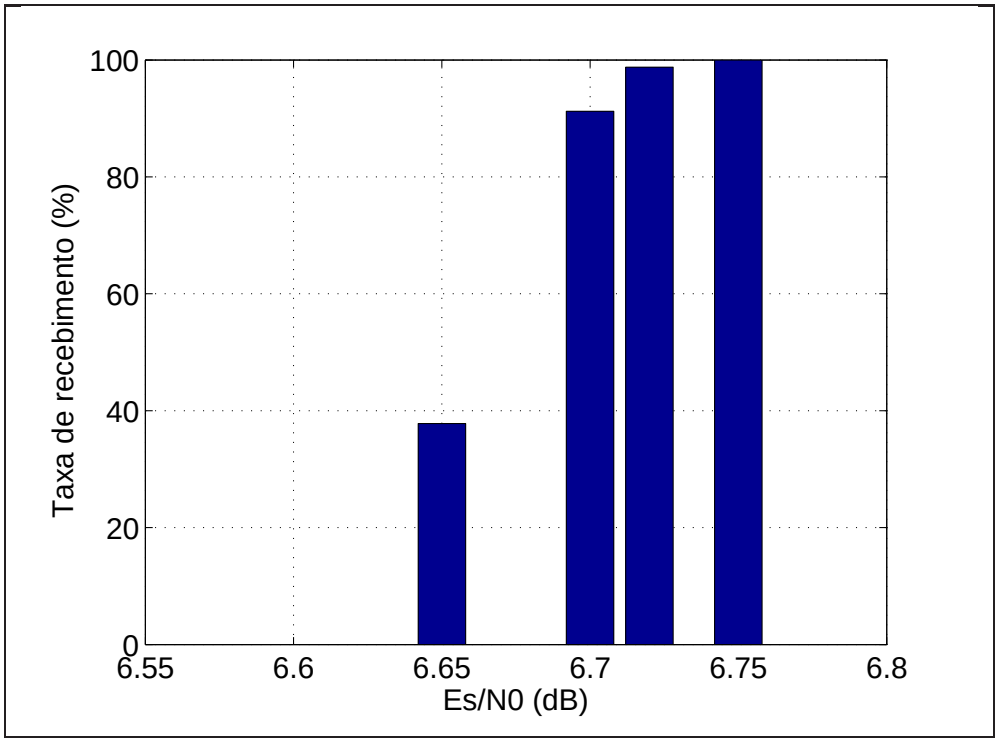


Figura 4.20: Quantidade de Pacotes recebidos com sucesso para modulação 8PSK 3/5.

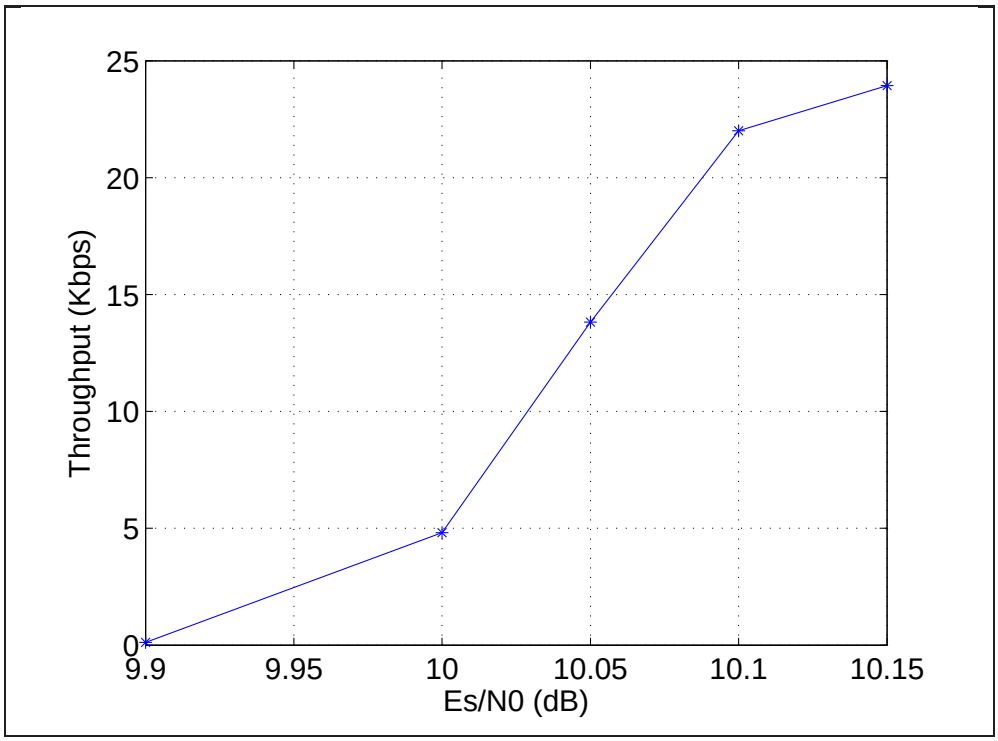


Figura 4.21: *Throughput* da rede para modulação 16APSK 3/4.

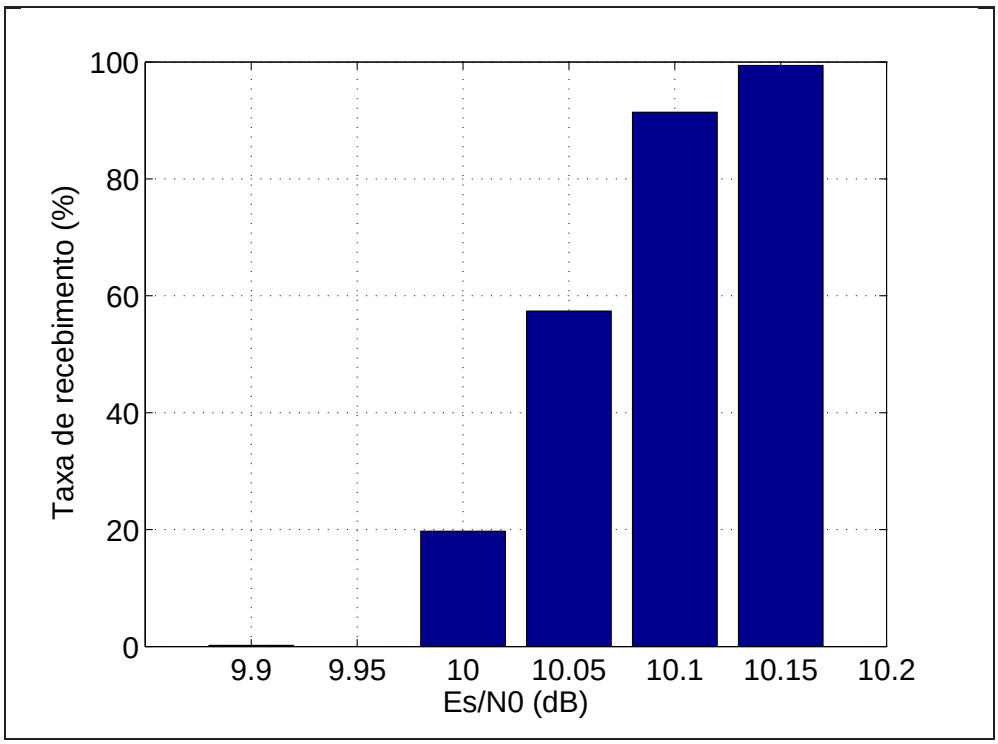


Figura 4.22: Quantidade de Pacotes recebidos com sucesso para modulação 16APSK 3/4.

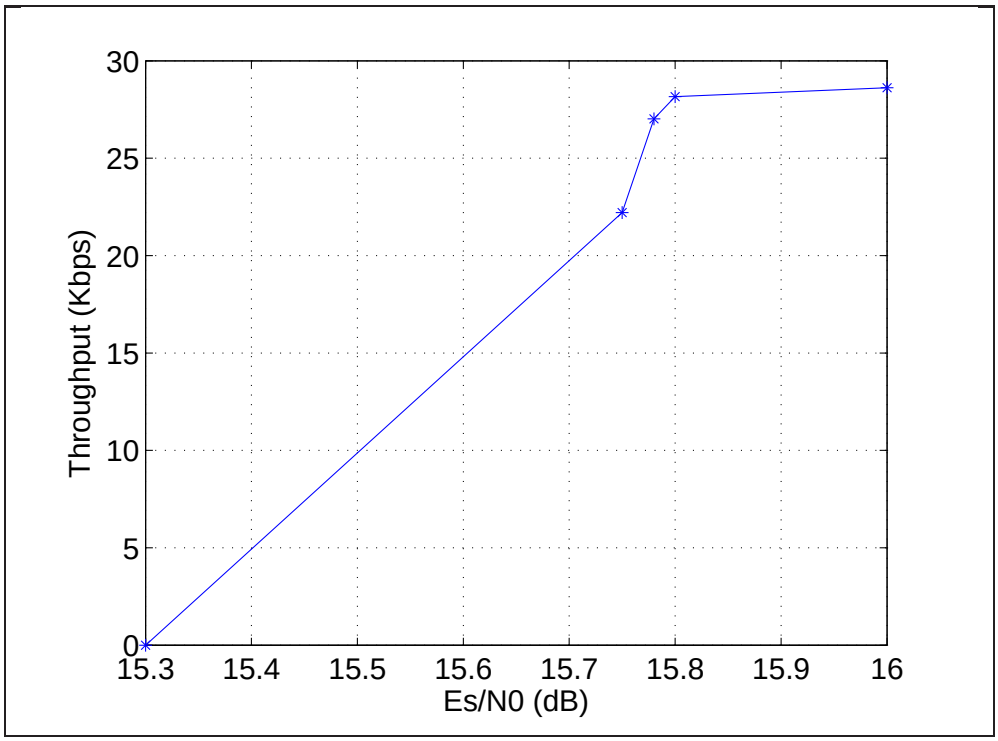


Figura 4.23: *Throughput* da rede para modulação 32APSK 8/9.

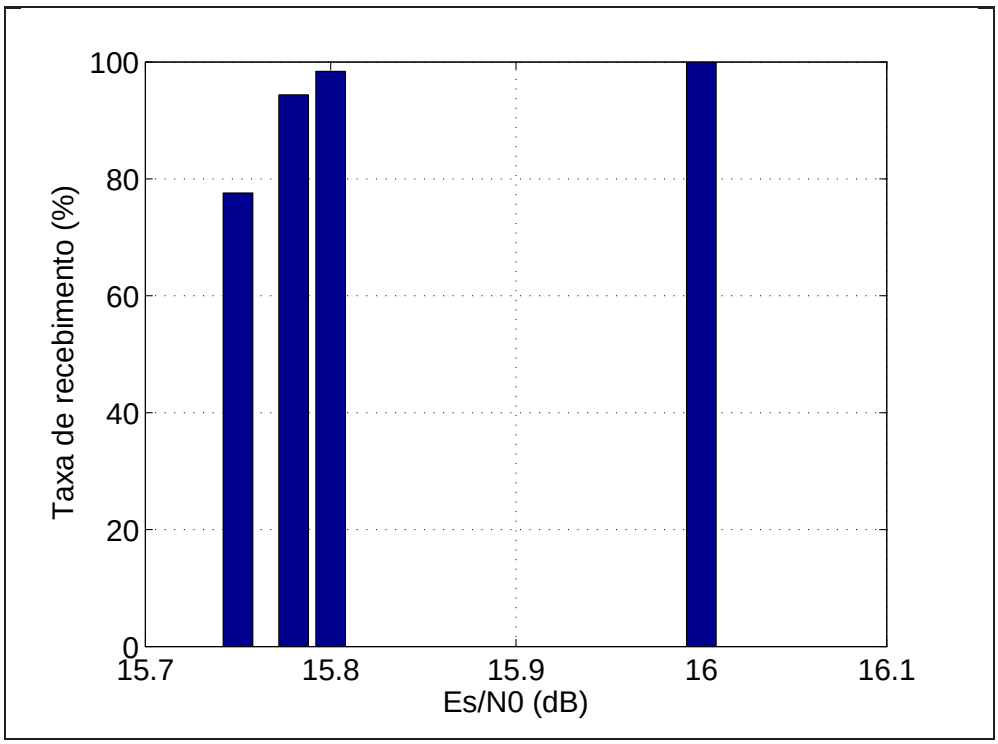


Figura 4.24: Quantidade de Pacotes recebidos com sucesso para modulação 32APSK 8/9.

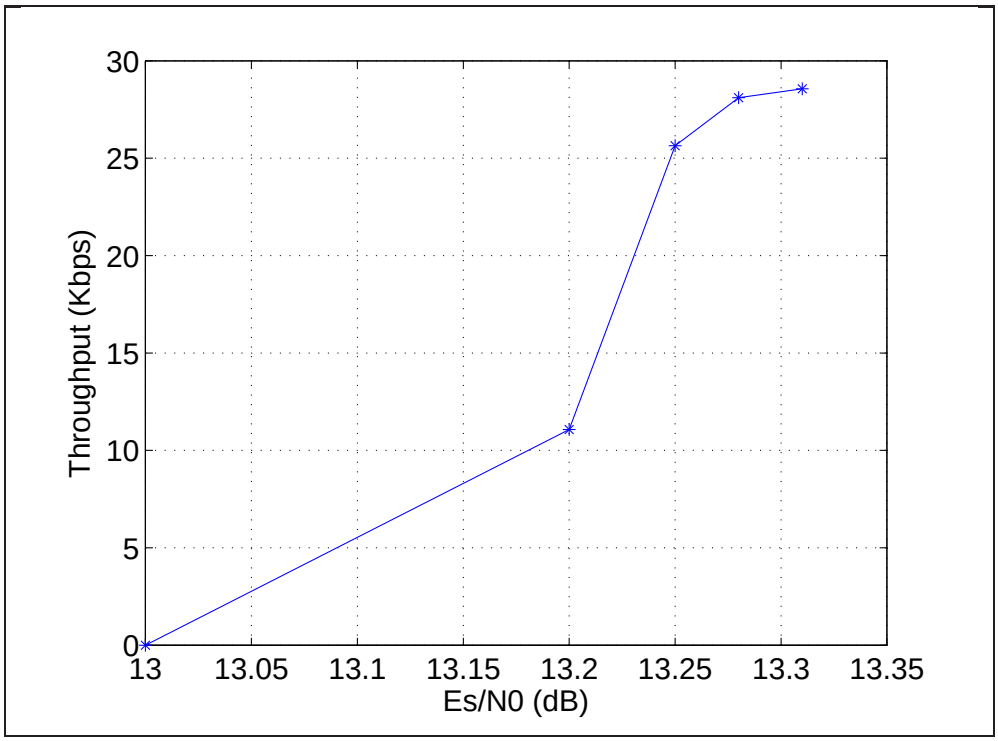


Figura 4.25: *Throughput* da rede para modulação 16APSK 8/9.

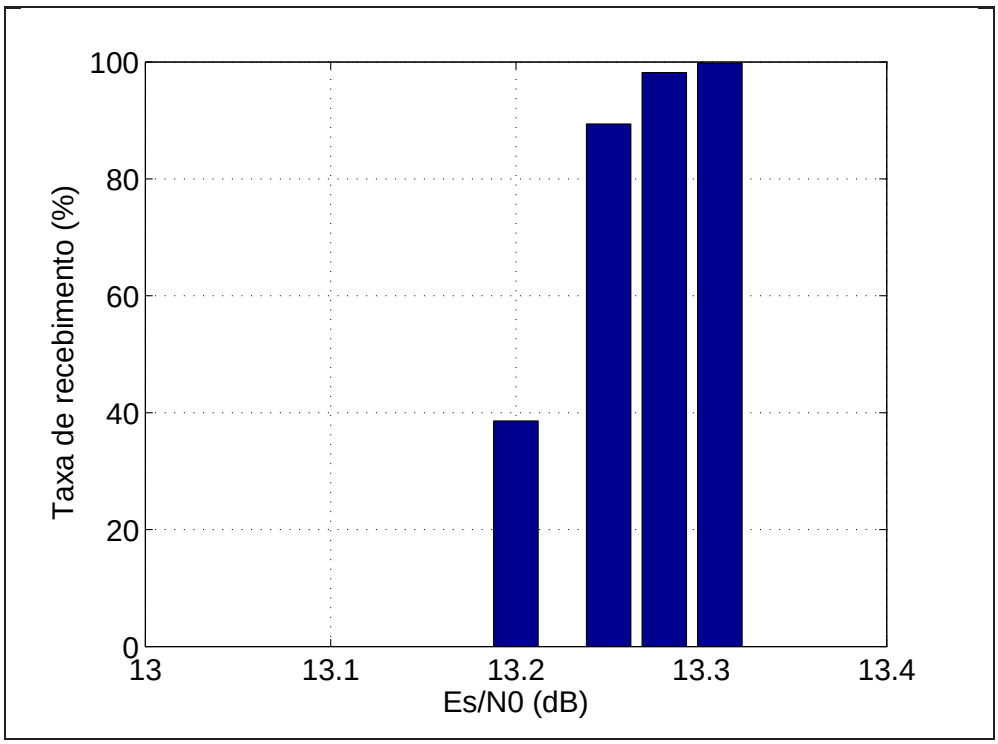


Figura 4.26: Quantidade de Pacotes recebidos com sucesso para modulação 16APSK 8/9.

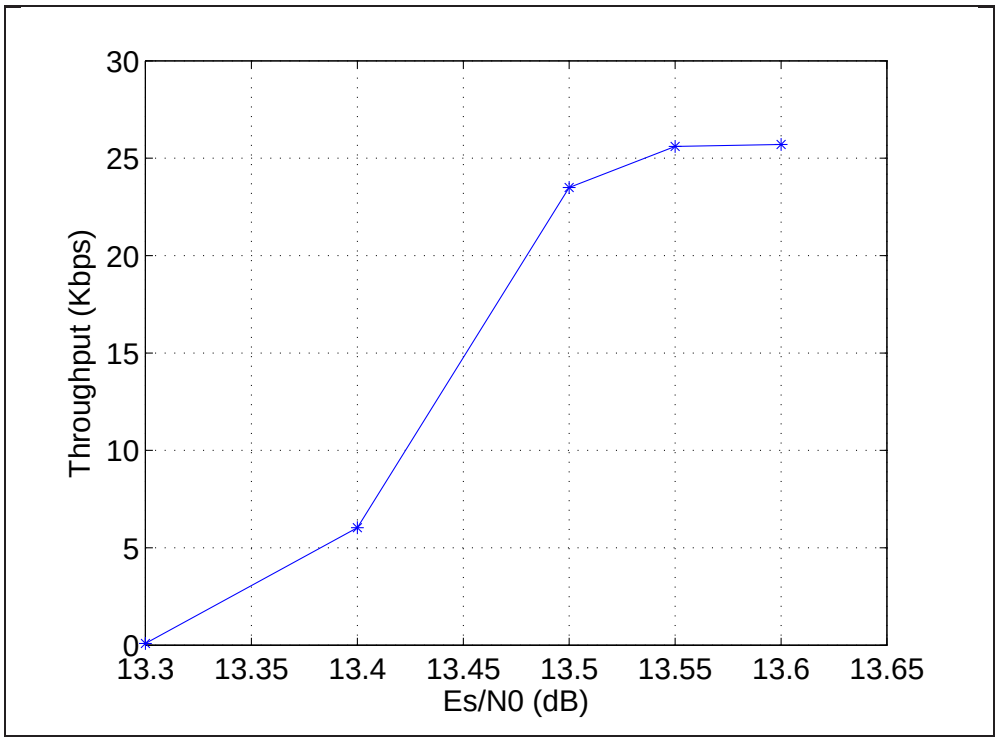


Figura 4.27: *Throughput* da rede para modulação 32APSK 4/5.

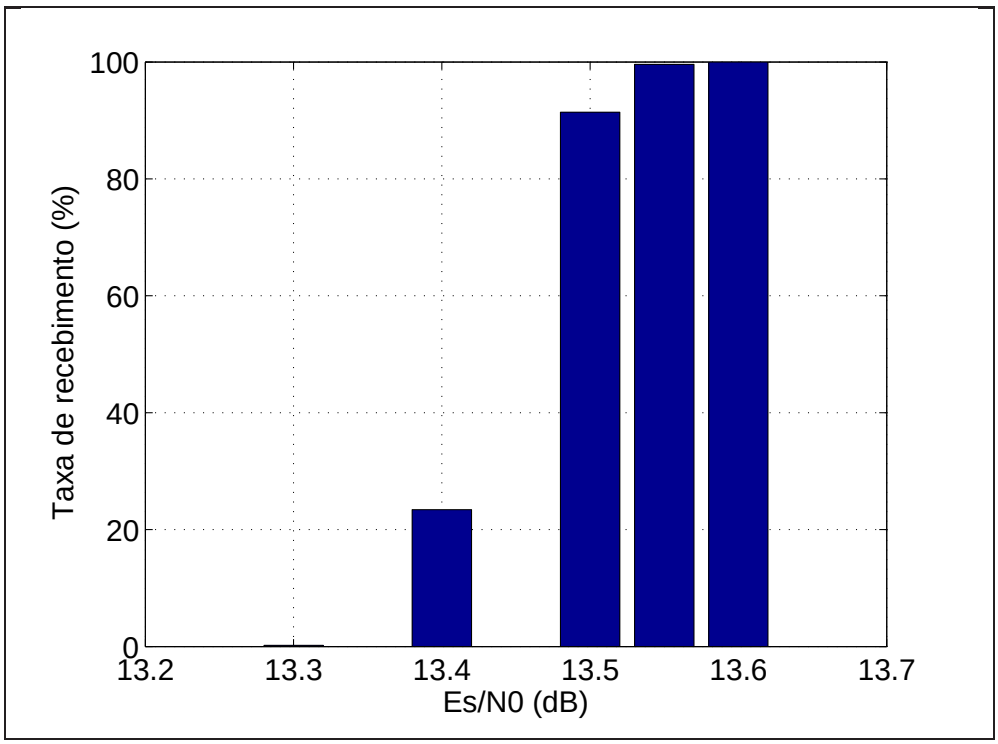


Figura 4.28: Quantidade de Pacotes recebidos com sucesso para modulação 32APSK 4/5.

Capítulo 5

Conclusão

Esta dissertação tem como um de seus objetivos criar um simulador capaz de testar a transmissão de pacotes IP por todas as camadas do modelo OSI utilizando um sistema de comunicações que pode ser definido pelo usuário. Escolhemos como sistema de comunicação o padrão DVB-S2 para transmissão de sinais de televisão via satélite. Este trabalho não visa somente testar este sistema para transmissão de Internet, mas também a criação de um pacote que permita este teste em qualquer sistema de comunicação, desde que este seja implementado em Matlab.

No Capítulo 1, foram apresentados uma breve história da televisão e o surgimento o projeto DVB. Foi mostrado também outras funcionalidades da transmissão de sinais de televisão digital e que há possibilidade de se transmitir diversos tipos de sinais, como pacotes IP, através dos sistemas definidos pelos padrões DVB.

No Capítulo 2, o padrão DVB-S2 é apresentado em detalhes, cada subsistema tem suas funções explicadas nesse capítulo e podemos saber como o sinal é modificado e que quadros são formados em cada um destes subsistemas.

No Capítulo 3, tem-se o simulador de redes completo para o padrão DVB-S2. O simulador foi construído utilizando dois *softwares*, o Matlab para a implementação de toda a parte física que inclui o padrão DVB-S2 e o canal, e o NS-2, que é responsável pela simulação da parte lógica da transmissão dos dados. As funções de cada uma dessas partes também são apresentados. Esse capítulo explica como fizemos esses softwares funcionarem juntos, formando um só simulador, e de forma transparente.

No Capítulo 4, apresentamos as simulações realizadas e os resultados obtidos. A primeira simulação se propõe a estudar o padrão DVB-S2 e verificamos que o simulador

implementado está condizente com o que o padrão estabelece, uma vez que os resultados obtidos estão bem próximos dos esperados. Podemos concluir também que não há grandes diferenças na transmissão de FECFRAMES e de PLFRAMES, e com isso comprovamos a eficácia do código corretor de erros aplicado no cabeçalho do subsistema da camada física do DVB-S2. Vimos também que podemos usar sem problema um receptor que não conhece o tamanho dos quadros recebidos. Os resultados obtidos com o simulador construído com o NS-2 e o Matlab permite comparar os *throughputs* para diferentes ModCods e analisar a quantidade de dados que cada um pode transmitir. A observação desse parâmetro facilita a comparação entre ModCods para um certo valor de taxa sinal ruído e permite sugerir um subconjunto de Modcods para utilização do sistema para transmissão de dados.

5.1 Trabalhos Futuros

Apesar da dissertação ter proposto um trabalho grande, integrando dois *softwares*, estudando cada um deles para fazer com que eles funcionem harmoniosamente e implementar diversas funções nestes dois *softwares* separadamente, ainda há funcionalidades que podem ser incluídas neste simulador.

Este simulador pode se tornar mais completo se implementarmos um canal que seja mais parecido com o canal real de satélite. Assim poderemos testar o uso de símbolos pilotos, que estão implementados no simulador, porém não foram utilizados nas simulações apresentadas aqui pois usamos um canal AWGN, co o qual não é necessário estimar o canal.

Para tornar este simulador ainda mais completo, podemos implementar também o padrão DVB-RCS que estabelece um canal de retorno via satélite para o padrão DVB-S2. Inserindo as funções do DVB-RCS no simulador, poderíamos testar também a transmissão de pacotes TCP, além da transmissão dos pacotes UDP.

Ainda podemos sugerir diversos sistemas de comunicação, como por exemplo outros padrões do DVB, que, se implementados no Matlab, podem ser utilizados para testar a transmissão de Internet. Uma vez que o pacote que integra o Matlab e o NS-2 está pronto e temos o simulador DVB-S2 como exemplo, muitos outros simuladores podem ser criados dessa mesma maneira.

Referências Bibliográficas

[1] ETSI. *Digital Video Broadcasting(DVB); Second generation framing structure, channel coding and moculation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)*, 2013. ETSI EN 302 307 v1.3.1 (2013-03). Acessado em 18 de setembro de 2013: `http://www.etsi.org/deliver/etsi_en/302300_302399/302307/01.03.01_60/en_302307v010301p.pdf`.

[2] WU, Y., HIRAKAWA, S., REIMERS, U. H., et al. "Overview of Digital Television Development Worldwide", *Proceedings of the IEEE*, 2006.

[3] NOJIRI, Y., SHOGEN, K. "The Challenge to Realize New Television Services at NHK Science and Technology Research Laboratories - HDTV Service with Satellite Broadcasting." *IEEE*, 2010.

[4] ETSI. *Framing structure, channel coding and modulation for 11/12 GHz satellite services*, 1997. ETSI EN 300 421 v1.1.2 (1997-08). Acessado em 2 de outubro de 2013: `http://www.etsi.org/deliver/etsi_en/300400_300499/300421/01.01.02_60/en_300421v010102p.pdf`.

[5] ETSI. *Framing structure, channel coding and modulation for cable systems*, 1998. ETSI EN 300 429 v1.2.1 (1998-04). Acessado em 9 de outubro de 2013: `http://www.etsi.org/deliver/etsi_en/300400_300499/300429/01.02.01_60/en_300429v010201p.pdf`.

[6] ETSI. *Framing structure, channel coding and modulation for digital terrestrial television*, 2009. ETSI EN 300 744 v1.6.1 (2009-01). Acessado em 9 de outubro de 2013: `http://www.etsi.org/deliver/etsi_en/300700_300799/300744/01.06.01_60/en_300744v010601p.pdf`.

[7] ETSI. *Transmission system for handheld terminals*, 2004. ETSI EN 302 304 v1.1.1 (2004-11). Acessado em 9 de outubro de 2013: `http://www.etsi.org/deliver/etsi_en/302300_302399/302304/01.01.01_60/en_302304v010101p.pdf`.

[8] REIMERS, U. H. "DVB - The Family of International Standards for Digital Video Broadcasting", *Procceedings of the IEEE*, 2006.

[9] MORELLO, A., MIGNONE, V. "DVB-S2: The Second Generation Standard for Satellite Broad-band Services", *Proceedings of the IEEE*, 2006.

[10] *Proposed Text on $\pi/2$ BPSK and (G) MSK description*. IEEE, 09 2008. IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs). Acessado em 3 de outubro de 2013: `https://mentor.ieee.org/802.15/dcn/08/15-08-0674-01-003c-proposed-text-on-pi-2-bpsk-and-g-msk-description.pdf`.

[11] ETSI. *Digital Video Broadcasting(DVB); Interaction channel for satellite distribuition systems*, 2009. ETSI EN 301 790 v1.5.1 (2009-05). Acessado em 9 de outubro de 2013: `http://www.etsi.org/deliver/etsi_en/301700_301799/301790/01.05.01_60/en_301790v010501p.pdf`.

[12] ETSI. *Digital Video Broadcasting(DVB); User guidelines second generation system for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)*, 2005. ETSI TR 102 376 v1.1.1 (2005-02). Acessado em 18 de janeiro de 2011: `http://www.etsi.org/deliver/etsi_tr/102300_102399/102376/01.01.01_60/tr_102376v010101p.pdf`.

[13] BALDO, N. "Dynamic Modules in NS: patch documentation". Acessado em 04 de dezembro de 2013: `http://telecom.dei.unipd.it/ns/miracle/ns_dynamic_libraries/`.

[14] BALDO, N., GUERRA, F., MAGUOLO, F., et al. "NS-MIRACLE: Multi-Interface Cross-Layer Extension library for the Network Simulator". Acessado em 04 de dezembro de 2013: `http://telecom.dei.unipd.it/pages/read/58/`.

[15] ISSARIYAKUL, T., HOSSAIN, E. *Introduction to Network Simulation NS2*. Springer Science+Business Media, LLC, 2009.

[16] POSTEL, J. *DOD Standart Internet Protocol*. Relatório técnico, USC/Information Sciences Institute, Janeiro 1980. IEN128, RFC760. Acessado em 11 de dezembro de 2013: `http://tools.ietf.org/html/rfc760`.

[17] SCHMIDT, B. "DVB-S2 - Getting Signal Constellations and Bit Mappings". Acessado em 09 de agosto de 2013: `http://www.mathworks.com/matlabcentral/fileexchange/31039-dvb-s2-getting-signal-constellations-and-bit-mappings`, .

[18] SCHMIDT, B. "Calculation of LLR values with variable noise variance". Acessado em 21 de agosto de 2013: `http://www.mathworks.com/matlabcentral/fileexchange/30110-calculation-of-llr-values-with-variable-noise-variance`, .

[19] ANNAMALAI, M. “Octave Forge - reedmullerenc.m”. Acessado em 28 de outubro de 2013: http://sourceforge.net/p/octave/communications/ci/default/tree/inst/reedmullerenc.m, .

[20] ANNAMALAI, M. “Octave Forge - reedmullerdec.m”. Acessado em 28 de outubro de 2013: http://sourceforge.net/p/octave/communications/ci/default/tree/inst/reedmullerdec.m, .

[21] LIN, S., COSTELLO JR., D. J. “Error Control Coding”. 2 ed., cap. 4, Pearson Prentice Hall, 2003.

[22] AZARBAD, B., SALI, A. B. “DVB-S2 Model in Matlab: Issues and Impairments, MATLAB - A Fundamental Tool for Scientific Computing and Engineering Applications”. v. 2, cap. 10, Vasilios N. Katsikis, 2012. Acessado em 18 de Março de 2013: http://www.intechopen.com/books/.

[23] BOUSSEMART, V., BRANDT, H., BERIOLI, M. “Subset optimization of adaptative coding and modulation schemes for broadband satellite systems”, *IEEE ICC*, 2010.

[24] “GNU Automake”. Acessado em 10 de outubro de 2012: http://www.gnu.org/software/automake/manual/html_node/index.html.

[25] “Compiling Matlab mex files with GNU autotools”. Acessado em 19 de outubro de 2013: http://gnumex.sourceforge.net/autotools/#autotools.

[26] GONÇALVES, L. C., DE OLIVEIRA CORRÊA, M. E., SAADE, D. C. M. “Tutorial de NS2”, Notas de aula da disciplina de Comunicação de Dados da Universidade Federal Fluminense. Acessado em 20 de Abril de 2011: http://www.midiacom.uff.br/~debora/redes1/pdf/tutorial-ns2.pdf.

Apêndice A

Bibliotecas Dinâmicas

A.1 Introdução

O projeto desenvolvido nesta dissertação requer a introdução de novos módulos ao programa NS-2 e a prática mais comum para isso é baixar uma versão oficial do programa, fazer as alterações necessárias no código fonte, como adição de novos arquivos na árvore de código existente e então compilar tudo isso em um novo executável do NS-2. Podemos perceber que isso significa que vamos alterar todo o código da distribuição do NS-2 com o qual estamos trabalhando, portanto esta deixará de ser uma versão oficial.

Mas há uma maneira de desenvolver novos módulos para o NS-2 sem precisar alterar o código fonte, esta solução é obtida utilizando bibliotecas dinamicamente carregáveis. Desenvolver novas extensões para o NS-2 é melhor desta maneira, pois temos várias vantangens, algumas delas são:

- Possibilidade de desenvolver novos módulos para o NS-2 (como novos agentes, tipos de pacotes e protocolos) sem precisar alterar o núcleo do simulador
- As bibliotecas dinâmicas podem ser carregadas durante a simulação, sem a necessidade de recompilar toda a distribuição do NS-2
- As modificações feitas com as bibliotecas dinâmicas tornam o conjunto do NS-2 mais os módulos desenvolvidos mais modular e escalável, assim sendo, adicionar novas características ao simulador será mais fácil e preservará a compatibilidade com versões anteriores.

Nas seções a seguir, o leitor encontrará informações sobre como criar e utilizar o pacote que permite o desenvolvimento de novos módulos para o NS-2 com as bibliotecas dinamicamente carregáveis. Informações mais completas (e na língua inglesa) sobre como criar o pacote com novos módulos podem ser encontradas no link informado na referência [13], neste apêndice estão escritas e traduzidas de forma sucinta as informações que possibilitam o uso do pacote.

Para baixar os arquivos necessários para usar bibliotecas dinâmicas, podemos baixar um exemplo de módulo criado com bibliotecas dinâmicas no site do departamento de telecomunicações da Universidade de Pádua (*Universitàdegli Studi di Padova*) que está citado na referência [14].

A.2 Como Construir Novos Módulos

Existem diversas maneiras de se compilar bibliotecas dinâmicas, porém a maneira apresentada neste trabalho será usando Autotools[1]. Vamos usar como exemplo a criação do módulo deste projeto chamado dvbs, caso o leitor queira criar seu próprio módulo, substitua o nome dvbs por aquele que desejar.

Primeiramente, devemos criar um diretório chamado dvbs, dentro desse diretório vamos criar dois outros, um chamado src que conterá os arquivos .cc e .h do novo módulo e um outro diretório chamado m4 que conterá o arquivo nsallinone.m4, arquivo este que podemos conseguir no pacote exemplo dei8021mr-1.1.4, dentro do diretório m4, que podemos conseguir no site da referência [14]. No diretório raíz, ou seja no diretório dvbs, criamos um executável denominado autogen.sh com o seguinte conteúdo:

```
#!/bin/sh

aclocal -I m4 --force && libtoolize --force && automake -- foreing
--add-missing && autoconf
```

[1]Autotools é um conjunto de ferramentas que ajudam a compilar e a criar de novos programas. Duas ferramentas que fazem parte desse conjunto é o Autoconf e o Automake. Elas ajudam a configurar e a compilar de maneira fácil arquivos de *Makefile* para um programa. Informações mais detalhadas dessas ferramentas podem ser encontradas em [24]

Depois de criar este arquivo e fazer com que ele se torne executável usando o comando `chmod`, criamos um outro com o nome *configure.ac*. Este arquivo contém instruções Autoconf, detalhadas por macros, para criar um script *configure*. No nosso caso o *configure.ac* deve conter o seguinte:

```
AC_INIT(dvbs,1.0)
AM_INIT_AUTOMAKE
AC_PROG_CXX
AC_PROG_MAKE_SET

AC_DISABLE_STATIC
AC_LIBTOOL_WIN32_DLL
AC_PROG_LIBTOOL

AC_PATH_NS_ALLINONE

AC_DEFINE(CPP_NAMESPACE,std)

AC_CONFIG_FILES([
Makefile
src/Makefile
m4/Makefile
])

AC_OUTPUT
```

O próximo arquivo criado no diretório raíz é o *Makefile.am* e, assim como o *cofigure.ac* utiliza macros para criar o arquivo *configure*, este novo arquivo que escrevemos também possui algumas macros que definem alguns elementos que as instruções do Automake necessitam para criar o arquivo *Makefile.ins*. No caso do nosso pacote, o *Makefile.am* deve conter o seguinte código:

```
SUBDIRS = src m4
EXTRA_DIST = autogen.sh
ACLOCAL_AMFLAGS = -I m4
DISTCHECK_CONFIGURE_FLAGS = @NS_ALLINONE_DISTCHECK_CONFIGURE_FLAGS@
```

Agora no diretório m4 criamos um arquivo Makefile.am que possui a linha de código abaixo:

```
EXTRA_DIST = nsallinone.m4
```

Neste momento devemos criar mais um arquivo Makefile.am, mas agora no diretório src, este arquivo irá conter a parte principal da configuração:

```
lib_LTLIBRARIES = libdvbs.la

libdvbs_la_SOURCES = dvbs.cc dvbs.h initlib.cc
libdvbs_la_CPPFLAGS = @NS_CPPFLAGS@
libdvbs_la_LDFLAGS = @NS_LDFLAGS@
libdvbs_la_LIBADD = @NS_LIBADD@

nodist_libdvbs_la_SOURCES = embeddedtcl.cc
BUILT_SOURCES = embeddedtcl.cc
CLEANFILES = embeddedtcl.cc

TCL_FILES = dvbs-init.tcl

embeddedtcl.cc: Makefile $(TCL_FILES)
cat $(TCL_FILES) | @TCL2CPP@ DvbsTclCode > embeddedtcl.cc

EXTRA_DIST = $(TCL_FILES)
```

Como podemos observar neste código, aqui dizemos quais são os arquivos fonte em `libdvbs_la_SOURCES`, os *compilers flags* e os *linkers flags* em `libdvbs_la_CPPFLAGS` e `libdvbs_la_LDFLAGS` respectivamente e devemos adicionar as bibliotecas criadas em `libdvbs_la_LIBADD`.

Existem mais dois arquivos especiais no diretório src que tem objetivos específicos, são eles o initlib.cc e dvbsinit.tcl. O primeiro arquivo contém uma função de inicialização denominada Dvbs_Init, que será chamada quando o novo módulo criado neste projeto for carregado. Este arquivo deve possuir o seguinte código:

```
#include <tclcl.h>

extern EmbeddedTcl DvbsTclCode;

extern "C" int Dvbs_Init() {
DvbsTclCode.load();
return 0;
}
```

No arquivo dvbs-init.tcl podemos incluir inicializações adicionais, caso o desenvolvedor queira.

Se todos os passos descritos até aqui foram executados corretamente, ao chamar o executável *autogen.sh* no diretório raíz do nosso novo módulo, ele deve gerar os arquivos necessários para compilar e instalar o pacote com nosso novo módulo. Somente na primeira vez precisamos rodar este arquivo, somente sendo necessário executálo novamente se incluirmos novos arquivos no projeto. Depois de chamar esse *autogen.sh*, só precisamos executar alguns códigos na linha de comando como mostrado a seguir:

```
$> ./autogen.sh
$> ./configure --with-ns-allinone=path/to/ns-allinone-2.34
$> make
$> sudo make install
```

Apêndice B

Matlab Engine e Biblioteca Matrix

B.1 Matlab Engine

O Matlab Engine é uma biblioteca que contém rotinas que permitem que o usuário possa chamar o Matlab a partir de seus próprios programas implementados em C, C++ ou Fortran.

Os programas da biblioteca Matlab Engine são *standalones* escritos em C/C++ e Fortran e se comunicam com um processo Matlab separado via *pipes*, se o usuário estiver programando em um ambiente UNIX.

Para a utilização do Matlab Engine, é necessário possuir uma versão instalada do Matlab na máquina. Se esta possuir apenas o Matlab Compiler Run, a biblioteca não funcionará.

B.2 Biblioteca Matrix

Quando utilizamos funções do Matlab através de nossos próprios programas escritos em C++, como é o caso desta dissertação, precisamos fazer uso das funções da biblioteca Matrix C/C++ para poder manipular as variáveis com as quais trabalhamos.

Ao escrever um programa em C/C++, trabalhamos com tipos de dados bem conhecidos por programadores, como por exemplo char, int, float, entre outros. Porém, ao trabalhar com a linguagem do Matlab, nos deparamos com a estrutura de dados utilizada por este *software*, o *mxArray*. Então a biblioteca Matrix é importante, pois suas funções nos ajudam a identificar dados do mxArray, assim como ler estes dados, verificar o tamanho, transformar

em tipos de dados conhecidos e fazer outras manipulações.

As funções desta biblioteca estão divididas em cinco grupos de acordo com seus objetivos: “Tipos de dados”, “Criar ou Deletar Vetores”, “Validar Dados”, “Acessar Dados” e “Converter Tipos de Dados”.

B.3 Gerando Bibliotecas Dinâmicas que usam Matlab Engine

Para fazer com que o NS-2 e o Matlab funcionem conjuntamente utilizando o Matlab Engine e a biblioteca Matrix, precisamos compilar as bibliotecas dinâmicas que criam o simulador DVB-S2 utilizando essas bibliotecas do Matlab. Para isso temos que alterar alguns arquivos criados para as bibliotecas dinâmicas que foram apresentados no Apêndice A.

Primeiramente, devemos baixar o arquivo “matlab.m4” no site referênciado em [25] e salvá-lo na pasta “m4”. Neste arquivo, precisamos fazer algumas alterações. O arquivo depende da arquitetura da plataforma linux em que as bibliotecas vão ser instaladas. Se o a arquitetura for de 32 bits, não é necessário fazer alterações, mas se a arquitetura for de 64 bits, temos que substituir no arquivo todos os “glnx86” por “glnx64”. A segunda e última alteração no arquivo é nos locais onde está escrito “MATLAB_LIB”, onde devemos adicionar ao final “-leng”. Dessa maneira adicionamos as bibliotecas do Matlab Engine para compilar as bibliotecas dinâmicas.

Feito essas alterações, o próximo arquivo a ser alterado é o “configure.ac”. Nesse arquivo vamos adicionar algumas linhas para configurar o Make com as bibliotecas usadas pelo Matlab Engine. O arquivo agora terá o seguinte conteúdo:

```
AC_INIT(dvbs,1.0)
AM_INIT_AUTOMAKE

AC_PROG_CXX
AC_PROG_MAKE_SET

AC_CONFIG_SRCDIR([matlab.m4])

AC_PROG_LN_S
```

```
AC_PROG_INSTALL
AC_PROG_MAKE_SET

AC_CONFIG_MACRO_DIR([m4])

AC_DISABLE_STATIC
AC_LIBTOOL_WIN32_DLL
AC_PROG_LIBTOOL

AC_PATH_NS_ALLINONE

AC_DEFINE(CPP_NAMESPACE,std)

CHECK_MATLAB

if test "$have_matlab" = "no"
then
    AC_MSG_ERROR([Must specify matlab directory using --with-matlab])
fi

AC_CONFIG_FILES([
Makefile
src/Makefile
m4/Makefile
])

AC_OUTPUT
```

Depois dessas alterações terem sido realizadas, agora iremos executar os códigos na linha de comando como mostrado na Seção A.2. A única modificação nesse passo é no comando "./configure" que agora deve conter o caminho para onde o Matlab está instalado

```
$> ./autogen.sh
$> ./configure --with-ns-allinone=path/to/ns-allinone-2.34 --with-matlab=/
path/to/MATLAB
$> make
$> sudo make install
```

O último detalhe para as bibliotecas com os nóvos módulos do NS-2 possam funcionar com o Matlab Engine, é a instalação na máquina do cshell, que é um shell modelado segundo a linguagem C.

Apêndice C

Network Simulator

C.1 Introdução

O Network Simulator 2 é um simulador *open-source* construído para pesquisas na área de comunicação de rede de computadores. Uma característica que devemos ressaltar é que o NS2 utiliza uma contagem discreta do tempo para simular as redes reais, sendo assim determinamos ocorrências de uma rede como eventos e estes podem ser programados para acontecer em certos momentos da simulação.

A arquitetura deste simulador é construída utilizando duas linguagens de programação, a linguagem C++, que é responsável pelo mecanismo interno de simulação dos objetos por ser uma linguagem mais robusta, e a linguagem OTcl (*Object-oriented Tool Comand Language*), que é usada para configurar a simulação, estabelecendo as relações entre seus objetos e agendando os eventos, ou seja, criamos todo o ambiente de simulação e os agentes que atuam nele. Isto é mais eficaz se feito com o OTcl, pois esta é uma linguagem interpretável e então é mais rápido para fazer modificações na simulação usando essa linguagem, pois não temos que recompilar todo o código.

C.2 Utilização do NS-2

C.2.1 Instalação

O NS2 foi desenvolvido em UNIX e portanto sua instalação deve ser feita nesta plataforma, porém há possibilidades de se instalar em um outro sistema operacional utilizando um emulador UNIX, como o Cygwin para Windows.

Existem duas maneiras de instalar o NS-2, uma maneira é utilizando *all-in-one suite* e a outra usando *component-wise*. O primeiro modo de instalação é o recomendado para usuários iniciantes, o pacote *all-in-one* possui os componentes necessários para o funcionamento do simulador além de alguns outros opcionais. Este pacote possui um arquivo com nome `install` que é um script que configura o ambiente do NS-2 e também cria seu executável. No pacote encontramos componentes principais para o funcionamento do simulador como o NS, o OTcl e o Tcl, e também encontramos alguns componentes opcionais que nos auxiliam na visualização das redes simuladas e no tratamento dos resultados, como o *Network AniMator* (NAM), o Zlib e o Xgraph.

Neste relatório será explicado apenas os passos para instalação utilizando o pacote *all-in-one suite*. Primeiro devemos instalar os pacotes `build-essential`, `autoconf`, `automake` e `libxmu-dev`. Já com estes pacotes instalados, devemos executar os comandos de instalar e validar da seguinte maneira:

```
shell>./install
shell>./validate
```

O primeiro comando instala o simulador e cria seu executável usando um arquivo `make` e o segundo comando "valida" o NS2 rodando alguns scripts que testam as funcionalidades básicas dos componentes instalados.

C.2.2 Criando Uma Simulação

Como vimos na seção C.1, a linguagem que utilizamos para criar o ambiente de simulação e seus agentes é a OTcl, então escrevemos um script OTcl onde criamos a simulação. Este script será lido por um interpretador e será gerada saídas específicas segundo a preferência do usuário. Em [26] os autores mostram um roteiro que facilita a criação do script, nele encontramos as principais etapas de construção de uma simulação. O roteiro é composto por sete passos que serão apresentados e detalhados a seguir:

- Criação do objeto simulador (escalonador de eventos);
- Abertura de arquivos para análise posterior (trace);
- Criação da topologia da rede (nós e enlace);

- Criação dos agentes da camada de transporte e conexão com os nós;
- Criação dos geradores de tráfego e conexão com os agentes da camada de transporte;
- Programação dos eventos da simulação (dinâmica);
- Fechamento da simulação, animação (NAM) e geração de estatísticas.

Nas subseções a seguir aprenderemos mais sobre cada fase da criação de uma simulação e também sobre a sintaxe utilizada pelo NS-2. Serão apresentadas algumas das principais classes usadas em grande parte das simulações.

Criação do Objeto Simulador

Na primeira linha de comando do arquivo onde escreveremos o script que conterá a simulação, devemos ajustar a variável que identifica o início da simulação da seguinte maneira:

```
set ns [new Simulator]
```

Esta linha cria o objeto simulador, associa-o à variável `ns` e realiza as seguintes tarefas:

- Inicializa o formato dos pacotes;
- Cria um escalonador de eventos;
- Seleciona o formato padrão de endereçamento.

Abertura de Arquivos para Análise Posterior

Após executar a simulação, vamos querer analisar os resultados e podemos até mesmo mostrar a simulação de uma forma gráfica. Para fazer isto, precisamos antes criar arquivos que guardarão algumas informações sobre a simulação, estes são chamados arquivos *trace*.

Na primeira linha do script OTcl criamos o objeto simulador, nas linhas seguintes vamos criar nossos arquivos *trace*. Se quisermos gravar informações sobre cada evento que ocorre nas simulações, gravamos um arquivo *trace* da seguinte maneira:

```
set tf [open out.tr w]
$ns trace-all $tf
```

Inicialmente devemos abrir um arquivo *trace* ou criar, caso ele não exista, com o comando `open` e, depois do nome do arquivo, que no exemplo é `out.tr`, escrevemos `w` para indicar que o programa irá escrever (em inglês ***W**rite*) neste arquivo. Este comando associa o arquivo *trace* aberto ou criado à variável `tf`. Na linha seguinte, o comando `trace-all` grava informações da simulação no arquivo *trace* em formato geral e assim poderemos analisar os resultados posteriormente.

Podemos também querer criar uma animação gráfica da simulação e para isto também precisamos abrir ou criar um arquivo *trace*. O comando utilizado também é o `open`, mas o arquivo deve possuir a extensão ".nam" para ser lido pelo *Network Animator* (NAM), que é a aplicação capaz de gerar as animações gráficas das simulações. Outra diferença em relação ao código que vimos é o comando para gravar as informações no arquivo *trace*, em vez de gravarmos em formato geral, agora queremos gravar no formato de entrada do NAM e para isto precisamos utilizar o comando `namtrace-all` em vez de `trace-all`. Então, para gravar os dados base para animação gráfica da simulação, devemos digitar no script:

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

Criação da Topologia da Rede

Criamos o objeto simulador e os arquivos para análise posterior da simulação, o próximo passo é criar a topologia da rede que desejamos simular. Nesta fase montamos a rede, definindo os nós e o enlace entre eles. Devemos ser cuidadosos na criação dos nós e dos enlaces, a topologia final da rede a ser simulada deve ser o mais próxima possível do real.

Primeiramente, devemos criar os nós. Para o NS-2 estes podem funcionar como hosts, enviando ou recebendo pacotes, ou como roteadores, apenas repassando pacotes para outros nós.

Os nós podem ser construidos como *unicast* ou como *multicast*, sendo que o padrão é o primeiro modo. Se quisermos nós *multicast* temos que especificar esta opção quando criamos o objeto simulador, escrevendo o seguinte comando:

```
set ns [new Simulator -multicast on]
```

Para criar os nós devemos digitar no nosso script as seguintes linhas:

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
```

Cada linha destas define um nó que é apontado pelas variáveis `n0`, `n1` e `n2`. Ao longo do script, se quisermos nos referir ao nó `n0`, por exemplo, devemos escrever `$n0`.

Se quisermos criar muito nós, podemos fazer um laço de repetição para que não tenhamos que digitar diversas linhas. Poderíamos criar os três nós mostrados anteriormente atribuindo a `numNode` o valor 3 e digitando as linhas

```
for {set i 0} {$i < numNode} {incr i} {
 set n($i) [$ns node]
}
```

Após criar os nós necessários para construir a rede desejada, temos que criar os enlaces que ligam estes nós, para que assim eles possam trocar dados entre eles. Dependendo da tecnologia de rede que foi empregada, a sintaxe dos enlaces pode variar. Como exemplo, utilizamos um enlace *full-duplex* entre dois nós:

```
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
```

Esta linha, que possui nosso exemplo de enlace, está criando um enlace bi-direcional entre os nós `n0` e `n1` que tem a capacidade de enviar 1 Mbits/s com um retardo de 10 ms. Se quisermos criar um enlace uni-direcional, devemos definir um `simplex-link` em vez de `duplex-link`. O último parâmetro utilizado define o tipo de fila, no exemplo temos um "DropTail" que representa um algoritmo FIFO (*Firts In, First Out*).

Após criar a topologia da rede, com todos os seus nós e enlaces, devemos determinar quais nós irão receber ou enviar dados ou mesmo realizar as duas tarefas e para isto devemos associar um agente ao nó. Cada agente roda uma aplicação que determina que tipo de tráfego é simulado. Nas próximas duas subseções, mostraremos como funciona a geração de tráfego no NS-2 e será explicado como funcionam as duas classes de objetos nas quais a geração de trafégo está baseada, a classe *Agente* e a classe *Aplicação*.

Criação dos Agentes da Camada de Transporte e Conexão com os Nós

Existem dois protocolos muito usados nas redes, o *User Datagram Protocol* e o *Transmission Control Protocol*, mais conhecidos por suas siglas UDP e TCP respectivamente. Para

fazer a implementação destes protocolos no NS-2, é necessário que criemos agentes, que são componentes da arquitetura do simulador que geram a simulação destes protocolos. Estes agentes estabelecem um canal de comunicação entre os nós transmissor e receptor, por onde irá trafegar as informações geradas pelas aplicações associadas a cada agente.

Os protocolos UDP e TCP possuem agentes diferentes, agora vamos começar pelo protocolo UDP. Para criar um agente, utilizamos o comando `new Agent/UDP` que cria o agente UDP e o comando `attach-agent` que associa o agente ao nó, que agora passará a ser um nó emissor. As linhas de código a seguir mostram a utilização destes comandos.

```
set udp0 [new Agent/UDP]
$ns attach-agent $ns0 $udp0
```

Tendo criado o agente UDP e anexado-o ao nó que será o emissor, precisamos criar o agente que irá receber os dados gerados e anexá-lo ao nó que servirá como receptor. O agente receptor do protocolo UDP é o *Null* e sua função é apenas receber os pacotes enviados a ele. Para criarmos o agente *Null* utilizamos o comando `new Agent/Null`. Nota-se que o comando é muito parecido com o de criação do agente UDP, só mudamos o nome do agente colocando o que queremos criar. Para associá-lo ao nó receptor usamos o mesmo comando `attach-agent` indicando o nó que queremos que seja o receptor. A criação de um agente *Null* e sua associação a um nó pode ser observada nas linhas a seguir. No exemplo associamos o agente ao nó 1.

```
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
```

Finalmente, tendo os agentes emissor e receptor, criamos o canal de comunicação, a nível de transporte, entre eles. Este canal, conhecido como seção, é criado pelo comando `connect` indicando os agentes emissor e receptor respectivamente. As linhas que seguem mostram a criação de uma seção entre agentes UDP e *Null*.

```
$ns connect $udp0 $null0
```

Para o protocolo TCP precisamos agentes diferentes. Primeiro devemos criar o agente emissor TCP, fazemos isso de forma bem parecida do agente emissor UDP, e depois devemos associá-lo a um nó. Usando os mesmo comandos já utilizados anteriormente, podemos fazer a criação do agente e a sua associação ao nó, como podemos observar.

```
set tcp0 [new Agent/TCP]
$ns attach-agent $n2 $tcp0
```

Da mesma maneira que criamos o agente receptor *Null* para o protocolo UDP, assim também o protocolo TCP tem um agente receptor específico, o agente *Sink*. Utilizando mais uma vez os comandos conhecidos para criação de agente e de associação ao nó, faremos isso para o agente *Sink* e vamos associá-lo ao nó que será o receptor do TCP.

```
set sink0 [new Agent/TCPSink]
$ns attach-agent $ns3 $sink0
```

Agora já sabemos como criar os agentes dos dois principais protocolos utilizados em redes IP, temos que associar a estes as aplicações responsáveis pela criação de dados, ou seja, pela geração das informações que trafegarão no canal criado entre os nós nos quais estão anexados os agentes emissor e receptor. A próxima subseção explica melhor como criamos estas aplicações.

Criação dos Geradores de Tráfego e Conexão com os Agentes da Camada de Transporte

Agora que já temos nosso canal de comunicação criado, podemos utilizar uma determinada aplicação que irá gerar dados que irão trafegar por este canal. Cada protocolo pode rodar determinadas aplicações, mas neste momento mostremos apenas um exemplo para cada protocolo apresentado na subseção anterior.

Como antes começamos mostrando como criar um agente do protocolo UDP, agora vamos começar explicando como criar uma aplicação usada por este protocolo. As primeiras linhas de código que serão apresentadas nesta subseção irão exemplificar a geração de um tráfego CBR(*Constant Bit Rate*), que comumente é usado para *streaming* de áudio e/ou vídeo.

```
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
```

Observando os comandos para a criação do tráfego CBR, a primeira linha cria a aplicação tráfego CBR, nas linhas subsequentes acontece o ajuste de dois parâmetros, o tamanho do pacote em *bytes* e o intervalo de transmissão em segundos. A última linha do código apresentado, associa a aplicação a um agente UDP. Observamos que apesar de usar o mesmo comando `attach-agent` que usamos para associar os agentes aos nós, existem algumas diferenças no uso das duas, como por exemplo onde é executado o comando. Para associar o agente ao nó, executamos o comando no domínio do `$ns` e, para associar a aplicação ao agente, fazemos isto no domínio da aplicação `$cbr0`, devemos tomar cuidado para não se confundir.

Agora, como exemplo de aplicação para um agente TCP, vamos mostrar a criação da aplicação FTP (*File Transfer Protocol*), nas linhas de código a seguir vemos como fazemos para criar tal aplicação e em seguida como associá-la ao agente responsável por executá-la. Podemos observar que o procedimento é bem parecido com o que foi apresentado para o tráfego CBR.

```
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
```

Programação dos Eventos da Simulação

Neste momento já temos a rede que desejamos simular montada, com nós, enlaces, agente e aplicações, temos apenas que indicar quando cada aplicação deve funcionar e por quanto tempo ela será executada. A dinâmica da simulação é feita com facilidade, uma vez que o NS-2 é um simulador orientado a eventos.

Para programar os eventos, primeiramente devemos estipular um tempo total de simulação e, durante este período determinado, podemos chamar alguns eventos específicos, como por exemplo início e finalização de tráfegos, quedas ou perdas de dados nos enlaces, entre outros.

A seguir temos um pequeno código que mostra como esta programação é feita. Neste exemplo, temos uma aplicação CBR que é iniciada no instante 0.5s e finalizada em 4.5s. Após 5s, chamamos o procedimento *finish*, que será mostrado na próxima subseção, e a simulação é encerrada.

```
$ns at 0.5 "cbr0 start"
```

```
$ns at 4.5 "cbr0 stop"
$ns at 5.0 "finish"
```

Fechamento da Simulação, Animação (NAM) e Geração de estatísticas

Já temos tudo o que precisamos para realizar a simulação, exceto alguns detalhes, o encerramento da simulação, a geração de estatísticas para análise posterior e a execução da animação, esta última é opcional. Para finalizar, criamos um procedimento que realizará estas tarefas, este é responsável também por fechar e limpar todos os arquivos trace já existentes com o mesmo nome, e abrir o programa NAM, caso este seja utilizado na simulação. A última linha do código apresentado a seguir possui o comando `run` que executa a simulação e inicia o escalonador de eventos do NS-2. Este comando deve vir sempre na última linha do script que contém a simulação.

```
proc finish{} {
  global ns nf
  $ns flush-trace
  close $nf
  exec nam out.nam &
  exit 0
  }

$ns run
```

Depois de seguir todos estes passos e criar assim um script que irá executar a simulação da rede que queremos, basta entrarmos em um terminal e digitar o seguinte comando para executar a nossa simulação.

```
$ns nome_do_script.tcl
```

C.3 Criando Uma Simulação com o Sistema DVB-S2

Para criar uma simulação no NS-2 utilizando o sistema DVB-S2, basta seguir as orientações básicas apresentadas na Seção C.2.2 e adicionar mais alguns parâmetros relacionados com o simulador do DVB-S2.

C.3.1 Carregando a Biblioteca do Simulador DVB-S2

A primeira linha de comando do arquivo onde escrevemos nossa simulação deve ser a criação do objeto simulador. Logo depois dessa linha, escrevemos o comando que carrega a biblioteca que implementa o simulador DVB-S2. Portanto as primeiras linhas do script ficam assim:

```
set ns [new Simulator]


load "/usr/local/lib/libdvbs.so"
```

O comando `load` carrega a biblioteca que possui o sistema desejado. Nesse caso a biblioteca `libdvb.so` possui as classes que implementam o DVB-S2.

C.3.2 Determinando os Parâmetros do Simulador DVB-S2

O DVB-S2 é um sistema de transmissão de televisão digital usando satélite, portanto, para criar as simulações no NS-2, vamos utilizar o pacote que configura redes criadas por satélite. Umas das partes que modificamos é o enlace do satélite, no qual adicionamos alguns parâmetros utilizados na simulação feita no Matlab. Mas antes de criar os enlace e os nós do satélite e dos terminais, devemos determinar as configurações do simulador de DVB-S2.

Logo após escrever o comando que carrega a biblioteca do DVB-S2, determinamos alguns parâmetros da simulação que são usados para a criação dos nós e dos enlaces. As linhas a seguir configuram algumas variáveis que carregam essas informações.

```
global opt
set opt(chan)           Channel/Dvbs
set opt(bw_up)          25Mb
set opt(bw_down)        25Mb
set opt(phy)            Phy/Sat
set opt(mac)            Mac/Sat/Dvbs
set opt(ifq)            Queue/DropTail
set opt(qlim)           50
set opt(ll)             LL/Sat
```

```
set opt(wiredRouting)   OFF
```

Analisando essas variáveis em ordem temos o `chan` onde indicamos que iremos utilizar o canal do simulador DVB-S2. Este parâmetro indica que o simulador NS-2 deve usar a classe com a qual chamamos o canal implementado no Matlab, como explicado na Seção 3.3.3.

Os parâmetros `bw_up` e `bw_down` determinam a banda-passante para os enlaces de *uplink* e *downlink* respectivamente. Os parâmetros `phy`, `mac`, `ifq`, `qlim`, `ll` e `wireRouting` indicam, em ordem, as classes e os valores para camadas física e MAC, o tipo e o tamanho da fila utilizada na simulação, a camda de enlace e o uso ou não de roteamento como em redes cabeadas.

C.3.3 Criando os Nós do Satélite e dos Terminais

Já tendo determinado os valores que vamos utilizar nas simulações, é o momento de criar os nós. Como estamos trabalhando com uma rede estabelecida por satélite, vamos ter dois tipos diferentes de nós, um para o satélite e outro para os terminais.

Para criar o nó de satélite, temos três opções como escolha, os satélites 'geo' (para geoestacionários), 'geo-repeater' (para satélites *bent-pipes*) e o 'polar'. Para as simulações com o NS-2, devemos utilizar o a opção 'geo'. Para criar o nó, configuramos o mesmo e depois criamos o nó propriamente dito com os comandos:

```
$ns node-config -satNodeType geo \
                -macType $opt(mac) \
                -phyType $opt(phy) \
                -llType $opt(ll) \
                -ifqType $opt(ifq) \
                -ifqLen $opt(qlim) \
                -channelType $opt(chan) \
                -downlinkBW $opt(bw_down) \
                -wiredRouting $opt(wiredRouting)

set n0 [$ns node]
$n0 set-position $lon
```

O parâmetro `satNodeType` indica o tipo de nó para satélite que vamos utilizar na simulação. Com o comando `set-position` indicamos a longitude `$lon`, em graus, na qual o satélite está localizado.

Para criar os nós dos terminais, utilizamos os mesmos comandos, mas mudamos os parâmetros utilizados. Em `satNodeType`, usamos a opção `terminal` e para a posição dos terminais indicamos sua longitude e também sua latitude, indicados no exemplos a seguir com `$lon` e `$lat`.

```
$ns node-config  -satNodeType terminal \
                 -llType $opt(ll) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(qlim) \
                 -macType $opt(mac) \
                 -phyType $opt(phy) \
                 -channelType $opt(chan) \
                 -downlinkBW $opt(bw_down) \
                 -wiredRouting $opt(wiredRouting)

set n1 [$ns node]
$n1 set-position $lat $lon;
```

C.3.4 Configurando os Enlaces de Satélite

Depois de criar os nós que vamos utilizar na simulação, o próximo passo é estabelecer o enlace entre eles. O comando usado para os nós de satélite e terminais é diferente do que o mostrado na Seção C.2.2. as linhas a seguir mostram os comandos para conectar um nó terminal a um nó satélite.

```
 $n1 add-gsl geo $opt(ll) $opt(ifq) $opt(qlim) $opt(mac) $opt(bw_up) \
    $opt(phy) [$n0 set downlink_] [$n0 set uplink_]
```

C.3.5 Determinando o Modelo de Erro

As simulações no NS-2 não carregam dados reais no pacote, apenas informações nos seus cabeçalhos. Para criar o simulador DVB-S2 no NS-2 utilizando o Matlab, tivemos que

alterar na criação do pacote do simulador de redes para que ele gerasse dados e os guardasse no pacote. Com isso, tivemos que alterar a classe que determina se um pacote foi recebido com sucesso ou não. Essa classe é a *Sat/Error* e as modifcações feitas nela estõa mais detalhadas na Seção 3.3.4.

Para utilizar a classe que verifica erros nos pacotes UDP transmitidos pelo DVB-S2, devemos escrever no script da simulação os comandos

```
set em_ [new ErrorModel/Dvbs]
$em_ unit pkt
$n2 interface-errormodel $em_
```

O `ErrorModel/Dvbs` indica que estamos utilizando a classe criada para a verificação de erros dos pacotes transmitidos através do sistema DVB-S2. O nó `$n2` no qual a classe de erro é conectada pelo comando `interface-errormodel` é o nó definido como terminal receptor.

C.3.6 Criando os Geradores de Tráfego Segundo os Parâmetros do DVB-S2

Por fim, a última modificação feita no simulador DVB-S2 é na geração dos dados da simulação. A forma como anexamos o agente UDP no nó terminal é igual a mostrada anteriormente. No exemplo a seguir, adicionamos um agente UDP `$udp0` ao nó `$n1`. O tamanho máximo do pacote a ser transmitido é determinado por `packetSize_` em bits.

```
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
$udp0 set packetSize_ 8000
```

Em seguida, determinamos o tipo de tráfego que vamos utilizar na simulação. Para o simulador DVB-S2, utilizamos um tráfego CBR modificado. Como mencionado na Seção C.3.6, o NS-2 não gera dados e preenche seus pacotes com eles, porém no simulador DVB-S2 os pacotes carregam dados e não apenas informações no cabeçalho. Para criar este tráfego, usamos os comandos a seguir.

```
set cbr0 [new Application/Traffic/CBR/RandomData]
```

```
$cbr0 attach-agent $udp0
$cbr0 set packetSize_ 4793
$cbr0 set interval_ 2
$cbr0 set modulation_ 2
$cbr0 set esn0_ 6.5
$cbr0 set pilots_ 0
```

O tráfego criado por `Application/Traffic/CBR/RandomData` é composto de bits aleatórios e cada pacote carrega o `packetSize_` bytes determinado nesse comando.

Um novo parâmetro adicionado ao tráfego original CBR é a modulação determinado por `modulation_`. Este parâmetro pode receber valores de 1 a 4, onde 1 corresponde a modulação QPSK, 2 a 8PSK, 3 a 16APSK e 4 a 32APSK. O valor padrão é 1. Mais outros dois parâmetros foram criados, o `esn0_` e `pilots_`. Em `esn0_` determinamos o valor de Es/N0 em decibéis no qual queremos fazer a simulação, o valor padrão para este campo é 20dB. O parâmetro `pilots_` configura o uso de símbolos pilot, se o valor for 0, a simulação não usa pilotos, mas se for escolhido 1, é porque na simulação serão transmitidos símbolos piloto. O valor padrão desse parâmetro é 0.

Para concluir a simulação, basta agora programar os seus eventos, fechar a simulação, realizar a simulação e analisar os resultados que estarão nos arquivos trace.