Universidade de São Paulo

Escola de Engenharia de São Carlos

NATÁLIA SANTA BÁRBARA CAPELARI

TELEMETRIA AERONÁUTICA COM ENVIO DE DADOS METEOROLÓGICOS E INFORMAÇÕES PARA CORREÇÃO DIFERENCIAL DE SISTEMAS DE RADIOLOCALIZAÇÃO

São Carlos - SP

2012

Capelari, Natália Santa Bárbara
C238t Telemetria aeronáutica com envio de dados meteorológicos e informações para correção diferencial de sistemas de rádiolocalização / Natália Santa Bárbara Capelari; orientador Ivan Nunes da Silva. São Carlos, 2012.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2012.

1. Telemetria aeronáutica. 2. Desenvolvimento de sorftware. 3. Sistema embarcado. 4. Estação em terra. 5. Ethernet. 6. iNET-X. 7. Sistema de posicionamento global. I. Título.

FOLHA DE APROVAÇÃO

Nome: Natália Santa Bárbara Capelari

Título: "Telemetria Aeronáutica com Envio de Dados Meteorológicos e Informações para Correção Diferencial de Sistemas de Rádio-Localização"

Trabalho de Conclusão de Curso defendido e aprovado em 28 / 11 / 2012,

com NOTA 10,0 (dez, zero), *pela Comissão Julgadora:*

Prof. Associado Ivan Nunes da Silva (Orientador)
SEL/EESC/USP

Prof. Dr. Rogério Andrade Flauzino
SEL/EESC/USP

M.Sc. Eduardo Sylvestre Lopes de Oliveira
SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

NATÁLIA SANTA BÁRBARA CAPELARI

TELEMETRIA AERONÁUTICA COM ENVIO DE DADOS METEOROLÓGICOS E INFORMAÇÕES PARA CORREÇÃO DIFERENCIAL DE SISTEMAS DE RADIOLOCALIZAÇÃO

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos,
Universidade de São Paulo.

Curso de Engenharia Elétrica com
ênfase em Eletrônica

***Orientador**: Professor Dr. Ivan Nunes da Silva*

São Carlos - SP

2012

AGRADECIMENTOS

Agradeço ao Prof. Dr. Ivan Nunes da Silva, pela orientação e apoio durante o desenvolvimento deste trabalho.

A Domingos H. B. Rios, responsável pelo meu aprendizado e desenvolvimento dentro da Embraer.

A Embraer (Empresa Brasileira de Aeronáutica), pelo apoio durante o desenvolvimento deste trabalho e pela oportunidade de estágio, onde pude aprender e crescer como profissional e como pessoa.

Agradeço também a minha família e amigos, sempre presentes em minha vida e dispostos a me ajudar.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE TRECHOS DE CÓDIGO

RESUMO

Atualmente, o Sistema de Telemetria da Embraer (Empresa Brasileira de Aeronáutica) utiliza comunicação unidirecional entre a aeronave e a Estação em Terra. Nesse sistema, a aeronave envia dados gerados pelo sistema embarcado para a unidade de telemetria em terra, onde serão tratados e apresentados aos engenheiros de ensaios em voo, que podem acompanhar o ensaio da aeronave em tempo real. Não é possível, entretanto, a comunicação em sentido contrário, ou seja, a Estação em Terra não é capaz de enviar informações para a Aeronave. Para manter-se atualizada com as mais recentes tecnologias, a Embraer tem desenvolvido um projeto que visa aumentar a confiabilidade do enlace de telemetria aeronáutico, criando um enlace de rádio capaz de enviar informações nas direções ar-solo e solo-ar, possibilitando então a atuação de um protocolo de recuperação de dados. Tal projeto, inicialmente, pretende enviar para a aeronave dados da estação meteorológica de pista e do sistema de radio-localização (GPS), utilizando tecnologia Ethernet. Neste trabalho de conclusão de curso, apresenta-se o desenvolvimento de dois *softwares* de telemetria, um deles responsável pelo envio dos dados de GPS com padrão de comunicação iNET ((*Integrated Network Enhanced Telemetry*) e outro aplicativo desenvolvido com finalidade de detectar todos os pacotes iNET da nova rede de comunicação de telemetria e selecionar quais desses pacotes serão transmitidos para a aeronave ou para a Estação em Terra.

Palavras chave: Sistema de Telemetria, Desenvolvimento de *Software*, Sistema Embarcado, Estação em Terra, tecnologia Ethernet, iNET-X, GPS

ABSTRACT

Nowadays, the Embraer's Telemetry System uses a one-way communication between the aircraft and the Ground Station. In this system, the aircraft sends data generated by the Airborne System to the telemetry unit on Ground System, where it will be processed and displayed to the Flight Test Engineers, who can monitor the aircraft's flight test in real time. However, communication in the opposite direction it's not possible. The Ground Station is not able to send information to the aircraft. To keep up to date with the latest technologies, Embraer has developed a project that aims to increase the reliability of aeronautical telemetry link, creating a radio link capable of sending information in air-ground and ground-to-air directions, allowing the performance of a data recovery protocol. This project initially intends to send the Weather Station data and track radio-location system (GPS) using Ethernet technology. This work of completion presents the development of two software, one responsible for sending the GPS data with standard communication iNET (Integrated Network Enhanced Telemetry) and the other application developed with the purpose of detecting all iNET packages of the new telemetry network communication and select which of these packets will be transmitted to the aircraft or to the Ground Station.

Keywords: Telemetry System, Software Design, Airborne System, Ground Station, Ethernet Technology, iNET-X, GPS

Capítulo 1. O PROJETO

1.1 INTRODUÇÃO AO PROJETO

O projeto de telemetria aeronáutica com envio de dados meteorológicos e informações para correção diferencial de sistemas de radiolocalização faz parte de um projeto maior da Embraer que é destinado a aumentar a confiabilidade do enlace de telemetria aeronáutico da área de ensaios em voo, criando um enlace de rádio capaz de enviar informações nas direções ar-solo e solo-ar, possibilitando então a atuação de um protocolo de recuperação de dados para trazer à estação de terra os dados de ensaios perdidos devido às dificuldades de radiotransmissão. Aproveitando esse sistema, o projeto permitirá também o envio para a aeronave de informações importantes como dados meteorológicos de pista e informações para correção diferencial de sistemas de radiolocalização, como GPS (*Global Positioning System*).

Nesse projeto da Embraer, será preservado o sistema de telemetria que já existe, mantendo o enlace PCM-FM como o principal canal de comunicação para ensaio de aeronaves protótipos, porém prevê a possível substituição no futuro do enlace PCM por uma telemetria baseada inteiramente em pacotes de informações, utilizando a tecnologia Ethernet.

Na Figura 1.1, tem-se o diagrama de blocos representando de maneira simplificada o atual enlace de telemetria e na Figura 1.2 pode-se ver o diagrama de blocos simplificado do futuro enlace de telemetria, como foi planejado para esse projeto. Note que o enlace atual foi mantido.

As características do sistema de telemetria serão introduzidas no Capítulo 2.

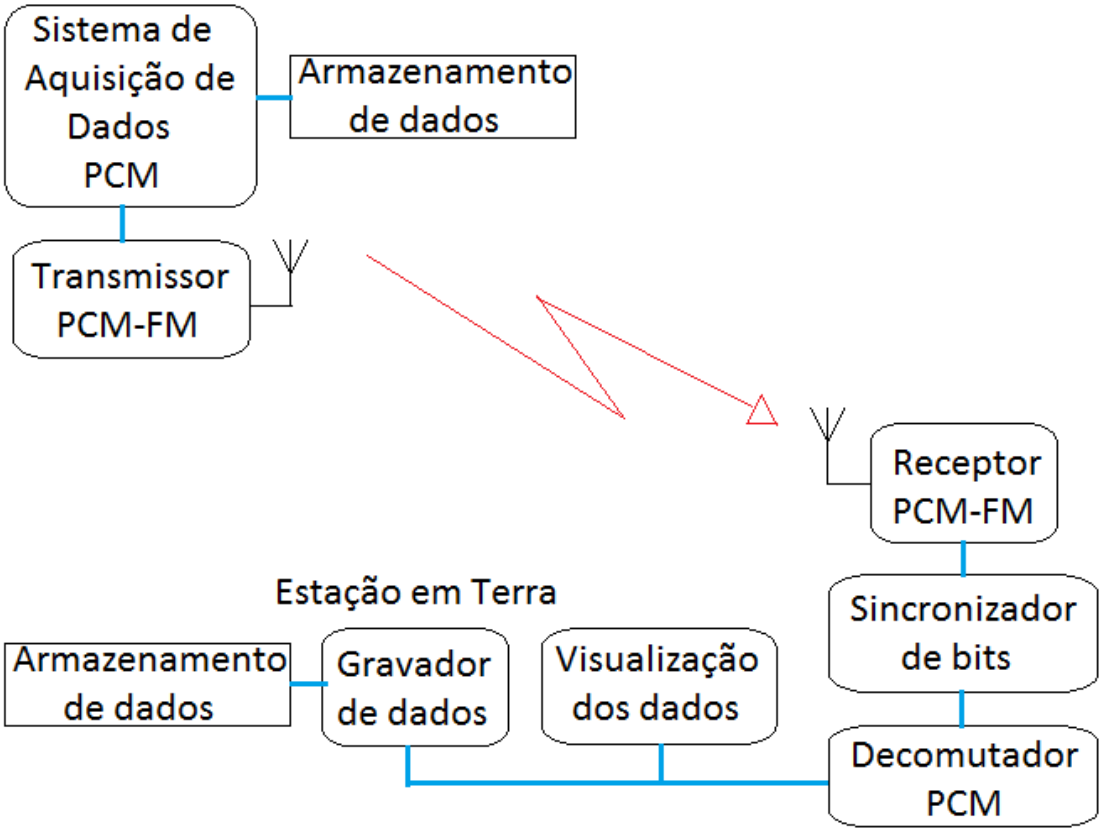

FIGURA 1.1 DIAGRAMA DE BLOCOS DO ENLACE ATUAL DE TELEMETRIA

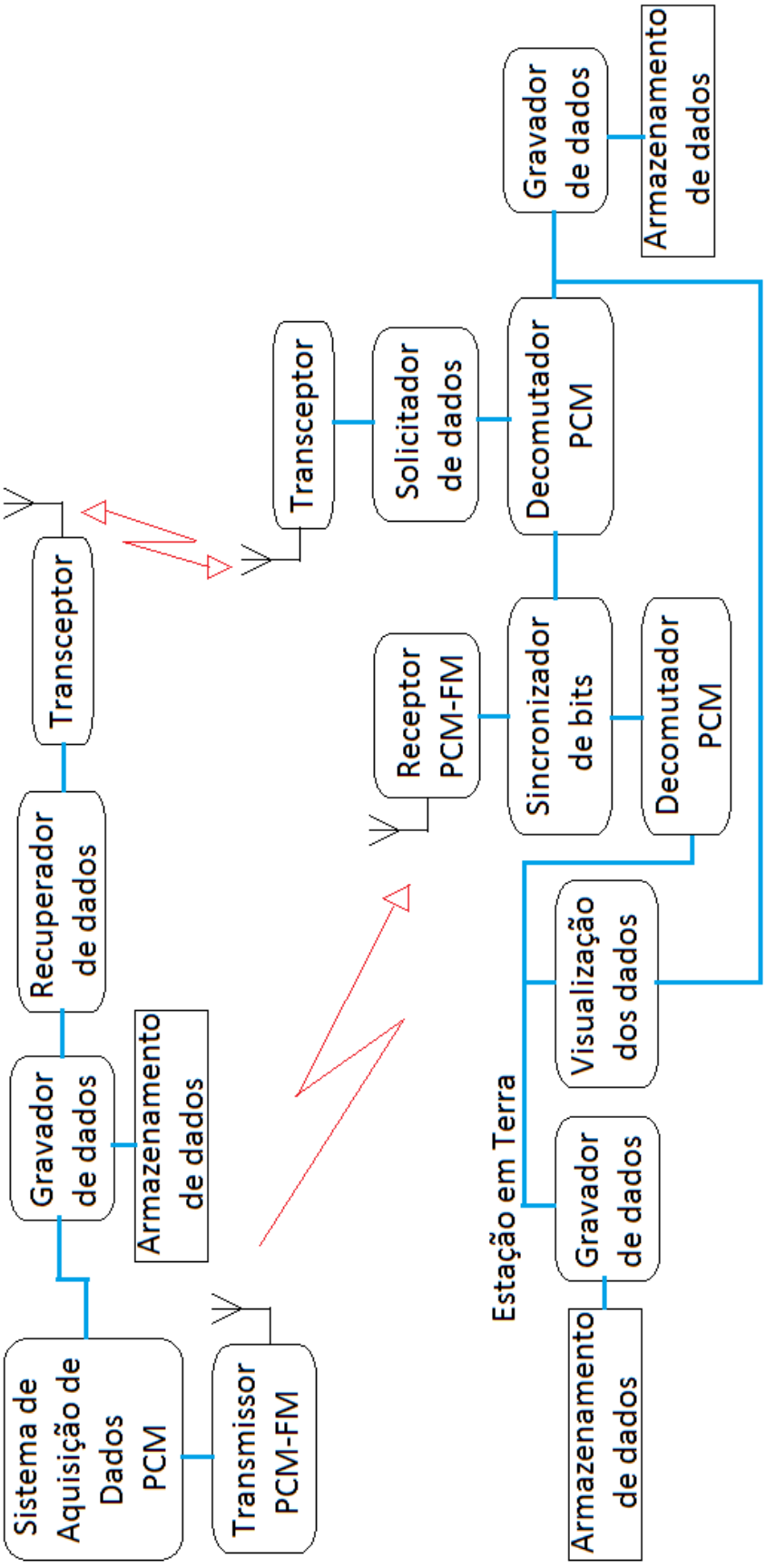

FIGURA 1.2 DIAGRAMA DE BLOCOS DO ENLACE FUTURO DE TELEMETRIA

O trabalho desenvolvido pode ser dividido em duas partes, ambas relacionadas ao envio de informações importantes para a aeronave.

Na primeira parte, foi desenvolvido um programa que recebe via comunicação serial informações de um GPS (de uma base fixa) para a correção do posicionamento fornecido pelo GPS embarcado. A informação a ser enviada para a aeronave se dá na forma de pacotes com formato iNET-X, como previsto no projeto da Embraer. Com as informações recebidas, serão feitos os cálculos de correção das informações recebidas do GPS embarcado, de modo a determinar com maior precisão o posicionamento do protótipo, em tempo real.

A segunda parte, ainda a ser desenvolvida, envolve o recebimento, tratamento e envio de informações meteorológicas da pista de pouso para a aeronave, também utilizando pacotes de informações. Desse modo, a comunicação entre o piloto da aeronave e a torre via rádio para obter informações das condições de pouso não serão mais necessárias, já que a aeronave poderá receber tais informações, com precisão, diretamente da estação meteorológica.

1.2 OBJETIVOS

O presente projeto tem como objetivo a criação dos programas, utilizando ferramentas como Microsoft Visual Studio, para envio das informações do GPS e da estação meteorológica para a aeronave via pacotes de informações. Além disso, o projeto destina-se também ao estudo e conhecimento detalhado dos equipamentos utilizados no projeto, como por exemplo, o conhecimento do Transceptor, dos equipamentos da estação meteorológica e do GPS utilizado. O projeto envolve conhecimentos referentes aos métodos de comunicação PCM, Ethernet e rádio.

Para o projeto, será também fundamental o desenvolvimento de um programa que auxilie a comunicação entre a aeronave e o sistema de telemetria em solo, capaz de selecionar as informações a serem enviadas ou recebidas em tempo real. Esse programa trabalhará em conjunto com o Transceptor, responsável por garantir a recuperação de pacotes perdidos durante a comunicação, pela requisição do reenvio destes.

1.3 FONTES DE RECURSOS

Os recursos utilizados, materiais e equipamentos são todos provenientes da Embraer.

Capítulo 2. INTRODUÇÃO À TELEMETRIA

Telemetria é o processo em que as características e informações referentes a um objeto são medidas e os resultados são transmitidos para uma estação onde serão gravados e analisados. A transmissão desses dados pode ser feita de diferentes maneiras, via satélites, cabos de comunicação, ondas de rádio, entre outros.

A telemetria permite que monitorar de um local seguro e conveniente o que está ocorrendo em um objeto de estudo, que pode estar em um local distante ou até mesmo perigoso.

Durante o desenvolvimento de uma aeronave, o sistema de telemetria tem importância vital. Na fase de ensaios em voo, são feitas diversas manobras a fim de se verificar, entre outras coisas, a estabilidade e segurança da aeronave em teste. Os dados coletados durante os testes são transmitidos em tempo real para a Estação em Terra (*Ground Station*) e podem ser analisados em questão de segundos.

A análise dos resultados em tempo real permite ao engenheiro de ensaio em voo decidir se será necessário repetir a manobra realizada ou aprová-la, seguindo então para a próxima manobra da lista, definida no cartão de voo. A verificação instantânea dos resultados de um ensaio garante que todos os dados desejados sejam adquiridos corretamente, evitando a necessidade de um novo e custoso ensaio.

Os dados dos ensaios em voo são também gravados em uma mídia de armazenamento, como, por exemplo, CDs ou DVDs, para posterior análise e arquivamento. Tais dados serão armazenados durante toda a vida da aeronave.

Atualmente, devido ao grande número de informações a serem transmitidas a respeito de uma aeronave, tornou-se muito custoso e impraticável a transmissão por canais separados para cada uma dessas informações. O processo de telemetria envolve, portanto, o agrupamento de todos os dados medidos em um formato comum, podendo ser transmitido como um único fluxo de dados. Uma vez recebidos, os dados são separados nas medições originais de cada componente e analisados.

2.1 PADRÕES DE TELEMETRIA: IRIG 106

Há normas e padrões a serem seguidos pelos sistemas de telemetria, a fim de fornecer critérios necessários no *design* e modificação de equipamentos. O propósito maior é garantir uma eficiente utilização de espectro na transmissão de dados, operações livres de interferência, interoperabilidade e compatibilidade de toda a gama de equipamentos.

A reunião dessas normas forma o IRIG-106 (*Inter Range Instrumentation Group*), desenvolvida e mantida pelo *Telemetry Group*, do *Range Commanders Council* (RCC).

O IRIG-106 inclui desde normas para utilização de nomenclaturas, numeração de bits, faixas de frequência, estampas de tempo, entre outras. Ele é atualizado a cada dois anos, sendo a versão mais recente IRIG-106 2011.

2.2 SISTEMAS DE TELEMETRIA

Um sistema de telemetria é unicamente configurado e construído de acordo com as especificações e requisitos de cada aplicação. O sistema de telemetria aeronáutica é constituído por duas partes: o sistema embarcado (*Airborne System*) e o sistema em terra (*Ground System*). A aquisição de dados é iniciada com a medição de atributos físicos através de sensores (transdutores). Tais medições são transformadas em valores chamados Unidade de Engenharia (*Engineering Unit - EU*).

Na Figura 2.1, pode-se ver um diagrama de blocos mostrando de forma simplificada o sistema de telemetria. Cada uma das etapas do processo será explicada nos tópicos a seguir.

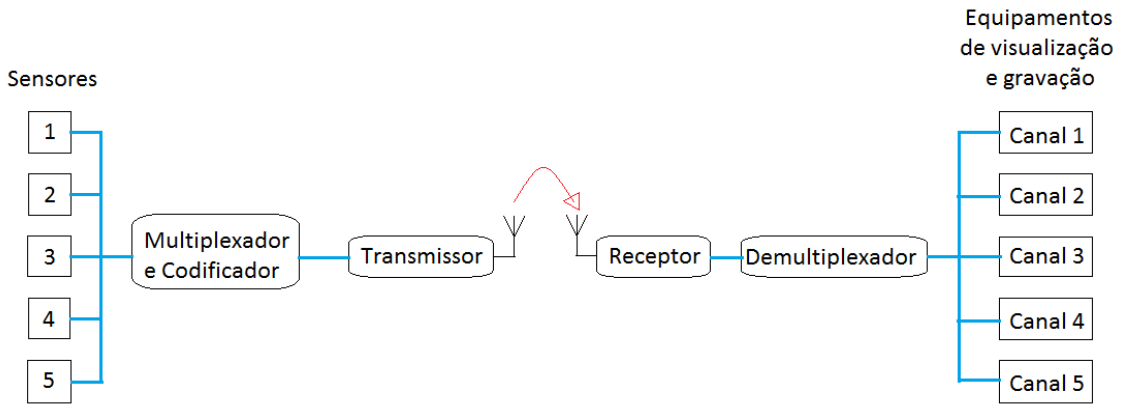

FIGURA 2.1 DIAGRAMA DE BLOCOS SIMPLIFICADO DE UM SISTEMA DE TELEMETRIA (ADAPTAÇÃO DA FONTE: [1])

2.2.1 SENSORES E TRANSDUTORES

Propriedade de transdução é a característica de certo evento físico capaz de representar um parâmetro e de ser transformado em sinal elétrico por meio de algum dispositivo ou processo. Transdução, por consequência, é o processo de converter a propriedade de transdução em sinal elétrico, que por sua vez poderá ser usado como entrada para um instrumento.

Segundo [2], um transdutor é o dispositivo que converte uma forma de energia ou quantidade física em outra.

De acordo com a definição em [3], os transdutores dividem-se em dois subconjuntos: os sensores, que fornecem informações de entradas em nosso sistema a partir do mundo externo e os atuadores, que executam ações de saída para o mundo externo. Utilizando esta definição, sensores são transdutores cuja ação é dar entradas do mundo externo para o sistema.

Ogata [4] estende um pouco mais a discussão sobre os transdutores, classificando-os como:

- Transdutor analógico: são transdutores nos quais os sinais de entrada e de saída são funções contínuas do tempo. As amplitudes dos sinais podem assumir quaisquer valores dentro das limitações físicas do sistema.
- Transdutor a dados amostrados: são transdutores nos quais os sinais e saída ocorrem apenas em instantes discretos de tempo, normalmente periódicos. As amplitudes do sinal são não quantificáveis.
- Transdutor digital: são aqueles nos quais os sinais de entrada e saída são discretos e a amplitude dos mesmos é quantificável, ou seja, pode assumir apenas certos valores discretos.
- Transdutor analógico-digital: são transdutores nos quais o sinal de entrada é uma função contínua do tempo e o sinal de saída é um sinal quantificável, podendo assumir apenas certos valores discretos.
- Transdutor digital-analógico: são aqueles nos quais o sinal de entrada é um sinal quantizado e o sinal de saída é uma função contínua do tempo.

Sensores podem ser também classificados quanto à forma de operação, direta ou indireta e a necessidade de alimentação externa, chamado de sensores passivos ou ativos:

- Operação direta: convertem uma forma de energia em outra sem que suas características sofram alterações;
- Operação indireta: são sensores que tem suas propriedades alteradas devido à ação de uma grandeza externa;
- Sensor ativo: um sensor ativo é aquele que requer uma fonte elétrica externa para alimentar o sensor;
- Sensor passivo: é aquele que provê sua própria energia, ou a deriva do próprio fenômeno que está sendo medido.

2.2.2 SISTEMA EMBARCADO *(AIRBORNE SYSTEM)*

2.2.2.1 Aquisição de dados

Na aquisição de dados, a saída de cada sensor deve ser transformada, filtrada ou então modificada para ser compatível ao tipo de sinal de entrada do próximo estágio do sistema. A relação entre o valor medido e o sinal de saída do sensor pode variar de acordo com o tipo de medição. Para auxiliar na definição dessas relações, usam-se condicionadores de sinal incorporados aos sistemas de calibração.

Durante a fase de teste da aeronave, geralmente são conhecidas as características físicas do que se pretende medir para definir e verificar a relação entre o sinal e a saída do sensor. Por exemplo, os *flaps* da aeronave, quando em solo, podem estar em certo ângulo conhecido, enquanto são feitas medidas nos sensores. Define-se, então,

a relação entre o ângulo de disposição dos *flaps* e o sinal de saída, que posteriormente será usado pelo sistema em terra.

2.2.2.2 Multiplexação

Um Multiplexador é utilizado para medir serialmente cada um dos sinais analógicos dos sensores, criando como sinal de saída um único fluxo de pulsos, cada um com uma tensão relativa ao respectivo canal de medição. O multiplexador tem também como papel a simplificação dos circuitos de aquisição de sinais, pois é extremamente custosa e incômoda a utilização de caminhos e cabos diferentes para cada um dos sinais provenientes dos sensores.

Como os sinais de diferentes sensores são combinados no processo de multiplexação, estes precisam ser transmitidos de forma a possibilitar a separação dos sinais depois do recebimento. Para isso, uma das soluções é a Multiplexação por Divisão de Tempo (*Time Division Multiplexing* – TDM), onde cada canal tem acesso ao link de transmissão apenas durante um curto período de tempo.

Nesse método, cada canal é amostrado pelo multiplexador, seguindo uma sequência predefinida. Assim que todos os canais forem amostrados, o processo é reiniciado a partir do primeiro canal. Desse modo, as amostras de cada canal estão sempre intercaladas por amostras dos demais, sempre seguindo a mesma ordem. Como todos os canais são constantemente monitorados, sua amostragem deve ter frequência suficiente para que o sinal não apresente mudanças bruscas entre duas amostras.

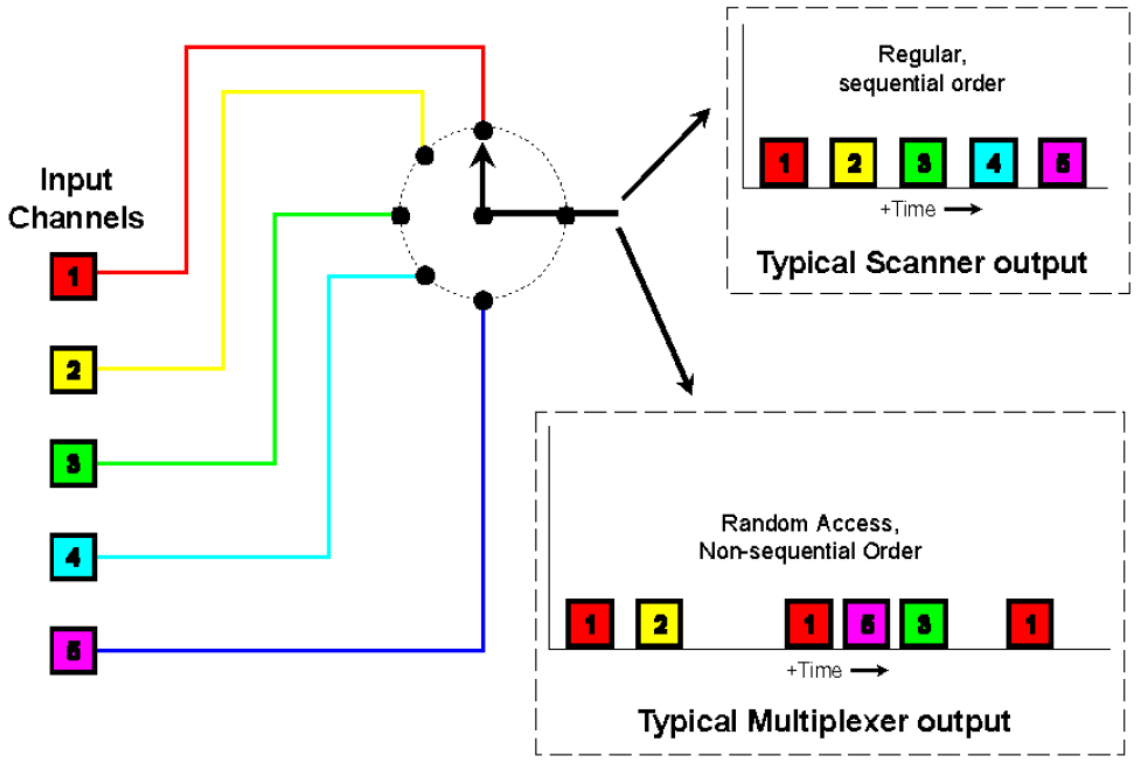

FIGURA 2.2 TÉCNICA DE MULTIPLEXAÇÃO POR DIVISÃO DE TEMPO (FONTE: [1])

Na Figura 2.2, tem-se um exemplo simples de TDM, com amostragem regular, onde todos os sensores são amostrados uma vez a cada ciclo e amostragem aleatória, onde a informações de um sensor é amostrada mais de uma vez por ciclo, dependendo apenas da frequência de disponibilidade do dado e frequência de amostragem necessária.

Um escaneamento completo feito pelo multiplexador (equivalente a uma revolução do Comutador) produz um módulo (*frame)*. Cada módulo produzido contém uma sequência de palavras (*words*) equivalentes o valor de cada medida. A cada revolução do comutador, é produzida a mesma sequência de palavras, de modo que só é possível saber a origem de cada uma dessas palavras se a ordem do escaneamento for conhecida. Além das *words* contidas no *frame*, é adicionada ao fim de cada *frame* uma palavra única, chamada Sincronizador de *Frame*, utilizada como referência para o processo de decomutação dos dados, isso é, o processo de separação dos dados.

Em um comutador simples, as informações medidas são amostradas apenas uma vez por revolução. No entanto, como a taxa de variação das medidas podem variar muito, a frequência de amostragem deve se acomodar a isso. Um exemplo é a medição de vibração, que requer um número muito maior de amostras por segundo do que a de temperatura.

De acordo com o Teorema de Nyquist [5], a taxa de amostragem de um dado deve ter ao menos o dobro da frequência máxima de variação do sinal para obter um modelo adequado desses dados. Porém, um sistema onde todos os dados fossem amostrados de acordo com a maior frequência entre os sensores seria inviável, principalmente pelo gasto desnecessário de banda para transmissão de tantos dados sem real necessidade, além do desperdício no espectro de frequência da portadora e de energia. A melhor solução é o uso de técnicas como Supercomutação e Subcomutação.

Supercomutação (veja a Figura 2.3) é o nome que se dá quando um dado é amostrado várias vezes durante uma revolução do comutador. Geralmente, o número de *words* entre duas medições supercomutadas é igual, para facilitar o processo de leitura do sinal.

Ao contrário da supercomutação, a subcomutação, na Figura 2.4, representa sinais que são amostrados apenas poucas vezes por tempo, pois não variam com frequência. Um bom exemplo de sinal que pouco varia é a posição do trem de pouso da aeronave, que está acionado durante a decolagem ou pouso, ou não acionado durante o voo.

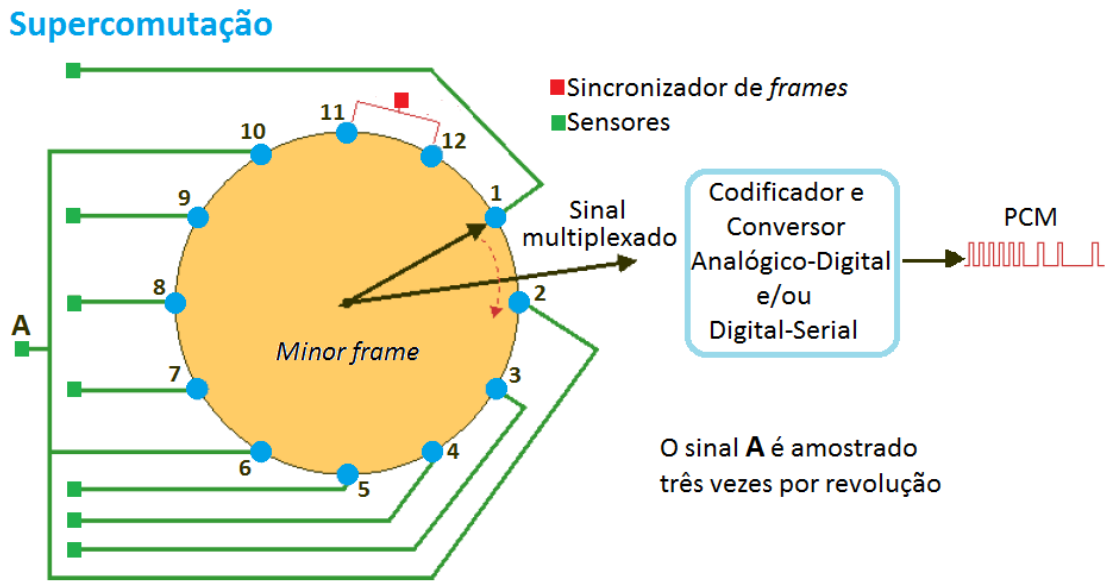

FIGURA 2.3 SUPERCOMUTAÇÃO DO PARÂMETRO A (ADAPTAÇÃO DA FONTE: [6])

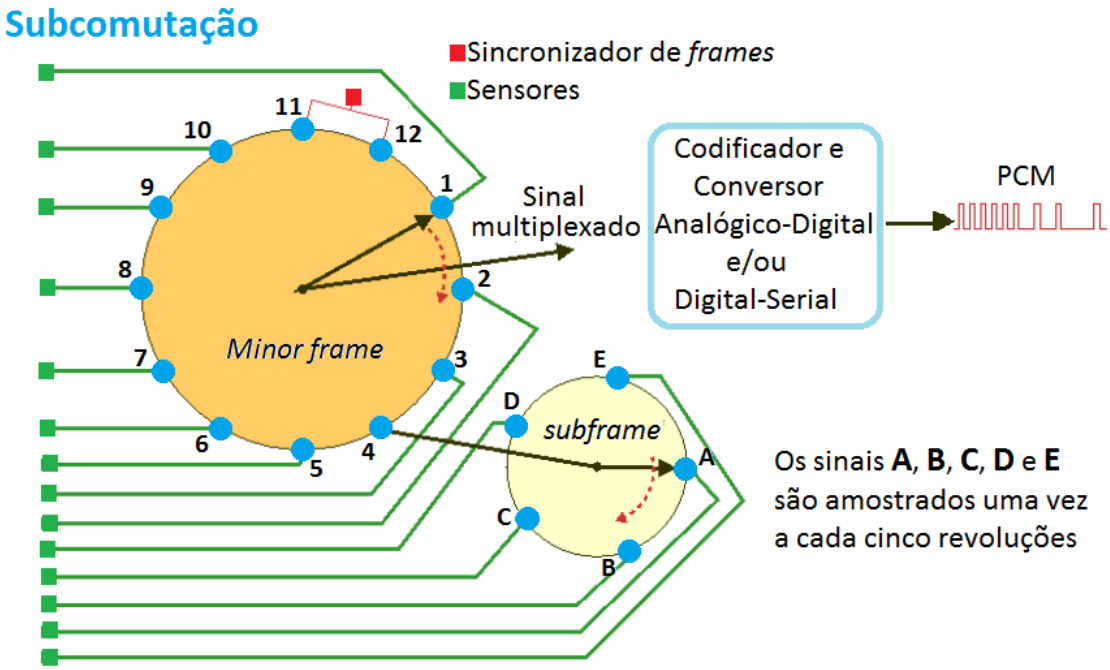

FIGURA 2.4 SUBCOMUTAÇÃO DOS PARÂMETROS A, B, C, D E E (ADAPTAÇÃO DA FONTE: [6])

2.2.2.3 Modulação por Código de Pulso (PCM)

O esquema onde a amplitude do pulso de TDM é proporcional ao valor medido pelo sensor, é chamado Modulação por Amplitude de Pulso (*Pulse Amplitude Modulation* - PAM). Esse tipo de modulação possui várias limitações, entre elas a sensibilidade a ruídos, pois pulsos distorcidos não mais representam uma medição real. Outra desvantagem é a limitação de precisão do sinal. É necessário um valor mínimo de diferença de amplitude entre dois pulsos para que estes sejam considerados de fato valores diferentes.

Outra forma de modulação é a Modulação por Duração de Pulso (*Pulse Duration Modulation* – PDM), onde a medição é representada por pulsos de mesma amplitude e comprimento diferentes. Semelhante ao PAM, esse tipo de modulação sofre limitações de precisão e sensibilidade a ruídos.

Hoje em dia, adota-se como formato padrão na telemetria sinais do tipo Modulação por Código de Pulso (*Pulse Code Modulation* – PCM), que será mais bem detalhado no tópico 2.2.3.1.

Na Figura 2.5, encontram-se exemplos dos três tipos de modulações de pulso citados, para uma onda senoidal.

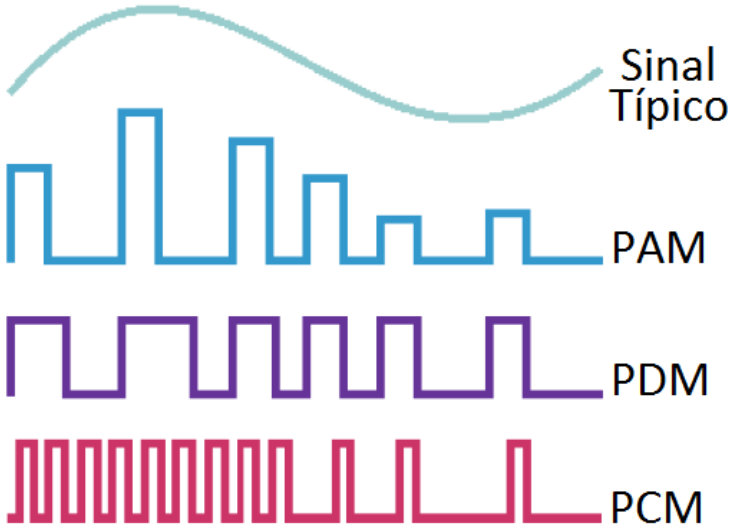

FIGURA 2.5 EXEMPLO DE MODULAÇÕES PARA UMA ONDA SENOIDAL (ADAPTAÇÃO DA FONTE: [6])

Na Modulação por Código de Pulso, a amplitude instantânea do sinal amostrado é representada por um código de vários dígitos binários, ou bits, resultando em uma série de pulsos e espaços. Todos os pulsos possuem a mesma altura e mesmo comprimento.

Um pulso distorcido não degrada o sinal, contanto que o pulso ainda possa ser reconhecido. Desse modo, é necessário apenas um equipamento receptor capaz de detectar a ausência ou presença de um pulso.

Sinais do tipo PCM são pouco suscetíveis a ruídos, bastante acurados e possuem resolução alta, limitada apenas pela resolução do Conversor Analógico-Digital (CAD). Outra vantagem é o grande número de canais que podem ser transmitidos, ideal para a telemetria aeronáutica. Em sistemas binários, o número de níveis de quantização possíveis em qualquer código é 2^n, onde n é o número de dígitos no código. Por exemplo, um código com palavras de 8 bits (1 Byte), frequentemente utilizado em telemetria, possui 256 níveis discretos de amplitudes acima de zero.

Depois de gerados os pulsos PCM de cada canal, estes são serializados por um Formatador, transformando o fluxo de dados paralelos em uma linha de pulsos binários. O sinal é então filtrado e transmitido.

As formas de transmissão são muitas. As informações podem ser enviadas via ondas rádio, cabos coaxiais, ethernet, ou gravados para posterior análise. Em muitos casos, os dados PCM não são apenas transmitidos, mas também gravados e armazenados.

Sistemas de telemetria que utilizam ondas de rádio ou outros métodos de transmissão sem fio (*wireless*) oferecem algumas vantagens diante dos demais. Entre elas, podemos citar:

- O sistema não utiliza linhas de transmissão ou fios que podem ser interrompidos ou danificados;
- Menor tempo de transmissão dos dados;
- Uso em regiões remotas onde a utilização de fios ou cabos não é pratica ou não é possível;
- Possibilidade de utilização de estações de recebimento móveis.

2.2.3 Sistema de Recebimento de Dados (Ground System)

Os dados recebidos são amplificados e reconstruídos por sincronizadores. Esse passo é necessário, pois o sinal sofre interferência de ruídos e pode ser distorcido durante a transmissão. Em seguida, um decomutador reconhece o padrão de sincronização, decodificando o fluxo de dados em seus canais originais, formando dados paralelos. O decomutador é responsável também por separar os dados de PCM nas medidas originais (chamados de dados primários).

2.2.3.1 PCM *Stream*

Como visto anteriormente, a sequência de pulsos que representa a amostra de um único canal é chamada *word*. Um ciclo completo de amostras, que inclui várias *words*, é chamado *minor frame*. Cada pulso ou espaço em uma *word* equivale a um dígito binário (*binary digit* – bit). O número total de bits em cada palavra determina a resolução da amostra, ou seja, o número de diferentes níveis discretos do sinal que podem ser identificados e codificados.

Cada *minor frame* possui, além das *words* representando os sinais medidos dos sensores e das palavras de sincronismo, sequências de palavras de Sincronização de *Frames* (*Frame Synchronization* – FS). Eles marcam o fim de um *frame*, para que a reconstrução do dado original na Estação de Solo seja possível.

Na Figura 2.6, tem-se a ilustração de um *frame* com parâmetros subcomutados. Observe que neste *frame*, tem-se a presença de *words* de sincronismo de *frames* e *words* de sincronismo de *minor frames*.

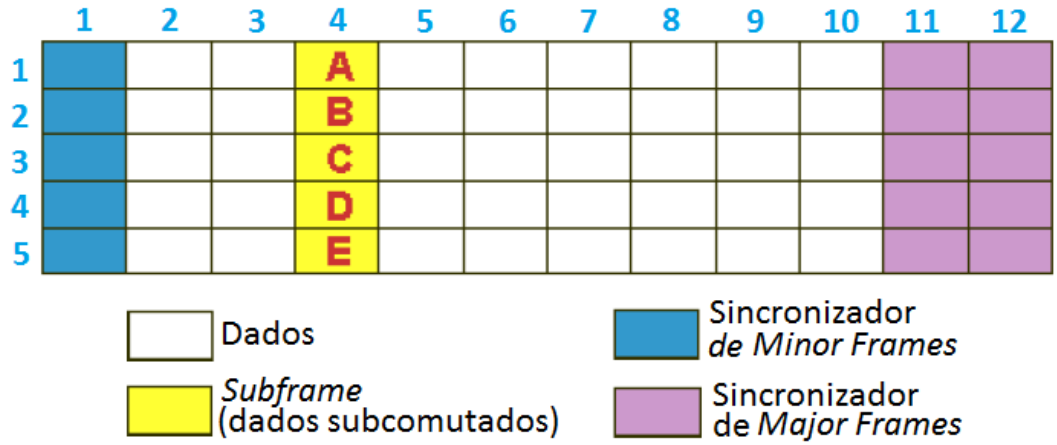

FIGURA 2.6 EXEMPLO DE UM *FRAME* COM DADOS SUBCOMUTADOS (ADAPTAÇÃO DA FONTE: [6])

Data words são medições, cálculos, contadores, comandos, funções, ou qualquer outra informação inserida dentro de um *frame*. É comum que, antes de serem preenchidos por data *words*, os *frames* contenham inicialmente *common words*. Formadas por uma palavra binária fixa ou então por valores ou funções predeterminadas, elas preenchem *words* não utilizadas, indicando que ali não há dados ou medidas úteis. Apenas durante o processo de escrita dos *frames*, essas common *words* serão substituídas por palavras de sincronismo ou *data words*.

Devido à supercomutação de alguns canais e/ou a subcomutação de outros, o grupo de *minor frames* que contém ao menos uma amostra de cada canal forma o chamado *major frame*. Consequentemente, cada *frame* dentro de um *major frame* deve conter palavras de identificação, também conhecidas como *Subframe Identification* (SFID). Estas palavras identificam e numeram os *minor frames*, auxiliando também na identificação e separação entre dois *minor frames* de *major frames* diferentes.

2.2.3.2 Reconstrução do PCM *Stream*

Antes de serem transmitidos, os pulsos retangulares são filtrados de modo a arredondar suas bordas, reduzindo a largura de banda requerida na transmissão. Depois de transmitido, o sinal chega à *Ground Station*, distorcido e com ruídos. É necessária a reconstrução desse sinal antes de utilizá-lo.

A primeira função de processamento de sinal reconstrói o sinal de modo a obter um número mínimo de erros e então o tempo de sincronização da informação é estabelecido através de uma série de pulos de *clock*, síncronos ao sinal recebido. Em seguida, o valor de cada bit é classificado e interpretado como zero ou um. Esse processo é chamado Sincronização de Bits. Os dados PCM só poderão ser corretamente decodificados se a sincronização dos bits for realizada com exatamente o mesmo tempo de *clock* do sinal no momento da codificação. Isso constitui uma das desvantagens da utilização desse método, pois um pequeno erro de sincronização impede a decodificação correta do sinal.

Depois da Sincronização de Bits, ocorre a Sincronização de *Frames*. A sequência de pulsos é convertida em palavras e as palavras de sincronização de frames são identificadas e isoladas. Só depois de isolados, os *minor frames* passam por um decomutador, que volta a separar e identificar as *words* em seus canais originais de acordo com a posição de cada *word* dentro do *frame*.

2.2.3.3 Decomutação

Cada *word*, depois de separada, passa por processos diferentes de acordo com a informação que possuem. Algumas passam por funções matemáticas e são transformadas em Unidade de Engenharia, outras são processadas e sofrem manipulações em seus bits, outras ainda, são mantidas como estão. Geralmente, palavras referentes a uma mesma medida ou informação são reorganizadas em uma nova sequência e enviadas para outros decomutadores ou outros hardwares dedicados.

Muitos decomutadores suportam *words* de diferentes comprimentos, de acordo com as características das medidas desejadas (*words* maiores implicam em dados de valores mais precisos). Isso significa que o conteúdo de todo o *frame* é alterado significativamente. Para reconhecer uma *word* corretamente, o decomutador deve alterar seu formato a cada palavra ou *frame*. Para obter uma resposta rápida, o decomutador deve manter em sua memória todas as possíveis definições dos frames. Em geral, o formato padrão utilizado em telemetria possui palavras de 16 bits no máximo.

2.2.3.4 Simulação e Codificação

Muitas vezes, um sistema de aquisição de dados ou um gravador analógico de instrumentação não estão sempre à disposição da estação de telemetria para produzir dados PCM para verificação dos sistemas ou treinamento de operadores. Para isso, utilizam-se simulações idênticas aos dados PCM gerados pelo sistema de aquisição.

Simuladores podem produzir *frames* estáticos, com taxas fixas, ou então *frames* mais complexos de acordo com a capacidade do decomutador. Podem ser gerados *frames* e *major frames*, com subcomutação, supercomutação, dados assíncronos, entre outros.

Existem também simuladores dinâmicos, que usam fontes externas de dados em tempo real. Estes nada mais são do que Codificadores PCM. Um novo sinal PCM é produzido extraindo as palavras de outro sinal PCM de entrada ou de outras fontes de dados.

2.2.3.5 Processamento em tempo real

O resultado da decomutação é a reconstrução das medidas dos sensores, dados ou *words* de computadores. Tais informações podem ser observadas em formatos amigáveis, como Unidades de Engenharia (unidades de distância, temperatura,

tempo). O processamento em tempo real requer que os dados brutos sejam convertidos e manipulados rapidamente, satisfazendo a necessidade de avaliação imediata de tais valores, garantindo a tomada de decisões em relação à segurança, continuação dos testes e outros controles.

Além da conversão para Unidades de Engenharia, processadores servem também para outras funções, como por exemplo:

- Verificação de valores: valores das medidas são verificados continuamente, para garantir que estes estejam dentro dos limites desejados, predizer problemas, alertar riscos, entre outros.
- Manipulação de bits: *frames* de telemetria nem sempre possuem uma medida por *word*. É relativamente comum que bits não utilizados sejam combinados de algumas *words* para formar um novo parâmetro. É papel do processamento em tempo real reunir tais medidas.
- Derivação de Parâmetros: muitas vezes, um atributo é resultado de múltiplas medidas e informações. Cabe ao processamento desses dados organizá-los e fazer os cálculos necessários para obter o resultado desejado.
- Compressão de dados: Dados com muitas amostras são geralmente comprimidos e simplificados utilizando algoritmos de amostragem ou média.

2.2.3.6 Exibição em tempo real

Monitores, televisores e outros displays são utilizados para mostrar os dados já processados, em seus valores alfanuméricos, ou gráficos. Tais dados podem também serem enviados via redes de comunicação locais, ou internet, expandindo a visualização para além da tela do operador de telemetria. Cada usuário pode ver os dados que mais lhe interessam, independentemente dos demais.

A visualização dos dados de telemetria por mais de uma pessoa é comum, pois a quantidade de informações fornecida é grande demais para uma única pessoa ser capaz de interpretar. Muitos dos dados gerados sequer passam pelos olhos de um engenheiro, ficando apenas como dados a serem verificados por *softwares* especialmente desenvolvidos para tais tarefas.

2.2.3.7 Arquivamento

Todos os dados capturados pela *Ground Station* devem ser arquivados para posterior análise e para satisfazer requisitos legais, garantindo que a aeronave em teste está propriamente qualificada.

Tipicamente, o armazenamento de dados é feito na *Ground Station*, tão logo sejam recebidos. Equipamentos caros de gravação armazenam os dados PCM logo depois da sincronização de bits. Também são armazenados os dados obtidos depois da decomutação e/ou do processamento e feitas cópias de segurança.

É comum também a gravação de parâmetros selecionados, ou *frames* específicos, de acordo com a necessidade.

Cópias desses dados são armazenadas em cofres de segurança, enquanto outras versões podem ser distribuídas para testes, análises posteriores, simulações, ou mesmo para outras empresas ou clientes interessados no produto.

Capítulo 3. DESENVOLVIMENTO DE *SOFTWARES*

Hoje em dia, a maioria dos sistemas incorporados a uma aeronave moderna conta com controles eletrônicos digitais, onde estão arquivados algoritmos de controle necessários, ditos *softwares*. Ou seja, a resposta (*output*) desejada de um sistema para um dado conjunto de condições (*input*) é obtida via um dispositivo eletrônico (computador) controlado por um uma sequência de instruções (programa), ao invés de meios mecânicos ou elétricos.

A principal razão de se adotar tais sistemas controlados por *software* é que eles podem ser prontamente programados para aceitar e processar dados nas quantidades, taxas e com um grau de complexidade que não pode ser alcançado por nenhum outro tipo de sistema de controle.

Esses sistemas são fisicamente compactos, leves, com baixo requerimento de energia e resfriamento. Sistemas individuais podem ser integrados com relativa facilidade e, conforme os requisitos do sistema se tornem individualmente e coletivamente mais complexos, o uso e significância dos *softwares* aumenta. Sistemas controlados por *software* são, e continuarão a ser, a tecnologia escolhida para muitas funções de controle no futuro.

Software não existe em sentido físico (embora seja componente do sistema eletrônico que controla) e por isso não pode ser diretamente testado por meios empíricos, embora, com algumas exceções, seu comportamento possa ser acessado monitorando seu desempenho no sistema que controla. Por ser empregado extensivamente na maioria das aeronaves modernas e provavelmente tornar-se-á ainda mais dominante no futuro, é importante que o Engenheiro de Ensaios em Voo (*Flight Test Engineer* - FTE) possa apreciar a natureza do *software* e os problemas relacionados ao seu *design*, desenvolvimento e testes (que dependem principalmente de técnicas não físicas).

3.1 INTRODUÇÃO AO SOFTWARE

O *Software* é representado pela documentação que o descreve e define o propósito das leis de controle envolvidas, além de seu desenvolvimento desde os conceitos iniciais, as mais detalhadas especificações, até o código executável, usado pelo computador. O *Software* é preciso, versátil, adaptável e facilmente modificado. É também imune a falhas aleatórias, podendo apenas causar problemas se contém erros gerados por um *design* defeituoso, ou seja, falhas de desenvolvimento, erros de código, erros de processamento de código, defeitos de implementação, ou se a própria documentação é errônea.

Se houver um erro no *design* do *software*, as mesmas respostas indesejadas serão provocadas toda vez que certa condição ocorrer. A probabilidade de se encontrar estes resultados indesejados, entretanto, depende somente da frequência com que

aquelas condições ocorrem. Consequentemente, é impossível predizer a quantidade de falhas de design que serão encontradas.

É importante salientar que as qualidades de adaptabilidade e fácil modificação de um *software* também têm desvantagens, pois quaisquer modificações, incluindo correções, podem introduzir novos erros. Por isso, qualquer modificação deve estar sujeita aos mesmos níveis de rigor de *design*, desenvolvimento e testes daqueles aplicados às modificações anteriores, além de disciplina rigorosa para se alcançar os resultados desejados.

3.2 DESIGN E DESENVOLVIMENTO DE SOFTWARES

O desenvolvimento de *software* é um processo progressivo, no qual as características de controle desejadas são expressas inicialmente em termos gerais e refinadas enquanto a arquitetura do sistema é desenvolvida. Um programa típico é construído em módulos e cada um deles deve ser produzido, testado e documentado.

3.2.1 TÉCNICAS DE TESTE

3.2.1.1 Análise do código

Esse processo não requer que o programa a ser testado seja executado. Ele consiste simplesmente na leitura do código escrito, procurando anomalias, erros de escrita ou de lógica de programação.

A maioria dos códigos-fonte é escrito em Linguagem de Alto Nível (*High Level Language* – HLL), pois são códigos mais fáceis de ler e escrever, menos suscetíveis a erros. Programas utilizados na implementação de um *software* possuem ferramentas de desenvolvimento que auxiliam o programador na tarefa de localização de erros de escrita e até apontam possíveis erros de lógica (como por exemplo, utilização indevida de constantes do tipo ponto flutuante no lugar de números inteiros).

Uma consideração a ser feita, entretanto, na escolha da utilização ou não de códigos em linguagem de alto nível é a transformação, ou compilação, necessária para converter o código fonte em um código executável (linguagem de baixo nível, ou linguagem de máquina). Essa transformação, feita por um compilador, nunca deve ser considerada totalmente confiável, pois também é passiva de erros. Em *softwares* com aplicações críticas, o código executável deve sempre ser conferido junto ao código fonte.

O uso de linguagem de alto nível requer precauções, sendo recomendáveis algumas condutas:

- Procurar minimizar a complexidade do código;
- Usar preferencialmente conjuntos e subconjuntos de instruções próprias da HLL utilizada;
- Ter disciplina na criação de códigos, utilizando padrões de notação de codificação;

- Utilizar ferramentas de verificação de fidelidade ao conjunto de instruções e padrões de codificação da linguagem.

3.2.1.2 Testes dinâmicos

Testes dinâmicos são conduzidos ao se executar o *software* no computador. Esses testes devem ser concebidos junto aos requisitos e especificações dos resultados desejados. Deve-se buscar conduzir tais testes em condições semelhantes àquelas que serão enfrentadas pelo *software* no ambiente real. Esse tipo de teste também verifica a integração *software-hardware*.

É importante buscar testar cada caminho ou possibilidade de execução, englobando a maior parte de ocorrências possíveis.

3.2.1.3 Erros comuns

- Erros em ferramentas: É possível que qualquer ferramenta, incluindo a ferramenta utilizada para auxiliar a criação do *software*, contenha erros. A indicação falsa de erros no código diminui a velocidade do processo de certificação do *software*, mas pode ser resolvida com relativa facilidade. Os problemas mais sérios e difíceis de resolver são os erros não detectados, geralmente uma forma bastante sutil de um problema genuíno. Eles ocorrem quando um erro na ferramenta coincide com um erro no código analisado, de modo que tal erro seja negligenciado;
- *Overload*: um *overload* ocorre quando a quantidade de dados em um processo excede a quantidade que o processo está preparado para lidar, causando a sobrecarga do barramento ou dos *buffers* de memória;
- Dados corrompidos: Todo *software* deve ser escrito de forma a lidar com a possibilidade de dados corrompidos (seja por ruídos, ou falha de um sensor, por exemplo). Para isso, deve tomar algumas precauções antes de utilizar o dado recebido dentro do programa: Garantir que se o dado esteja dentro da faixa esperada de valores e verificar a razoabilidade do dado (por exemplo, comparando seu valor com outro dado anteriormente recebido, levando em consideração o comportamento esperado da informação).

3.2.2 O papel do Engenheiro de Ensaios em Voo (FTE)

A validação de um *software* é geralmente conduzida por um especialista como parte integrante do desenvolvimento do sistema, e não faz parte do trabalho do FTE. Entretanto, para ser capaz de avaliar o desempenho do *software*, o FTE deve estar apto a entender os relatórios do especialista, cujo conteúdo, entre outras coisas, mostra se o código e especificações do *software* são congruentes e livres de erros.

Além disso, o FTE deve solicitar evidências de que os conceitos do *software* foram desenvolvidos corretamente, que a documentação das especificações não tenha ambiguidades e que durante os testes sejam realizados somente o que foi requerido para o *software*. Essas exigências só poderão ser alcançadas se houver um perfeito

entendimento entre o desenvolvedor e o usuário do *software*. Enquanto os procedimentos empregados devam limitar as oportunidades de o *software* ser utilizado erroneamente, o FTE deve considerar aspectos decorrentes de erro humano.

O FTE deve também ter uma visão geral de todo o sistema, conhecendo as possíveis consequências decorrentes dos defeitos que o *software* possa ter dentro do sistema e vice-versa.

É essencial que o FTE esteja suficientemente envolvido na avaliação do *software* de maneira que ele esteja confortável com as características de todo o sistema, especialmente os métodos (e as limitações desses métodos) usados na validação do *software*. Por fim, o FTE deve ter provas oferecidas pelos especialistas para julgar, no contexto de todos os fatores relevantes, que o *software* desenvolvido está adequado para realizar seu papel.

Capítulo 4. PADRÃO DE TRANSMISSÃO ETHERNET

A tecnologia Ethernet foi desenvolvida em meios da década de 70, inicialmente baseada nos conceitos da rede ALOHA, criada na Universidade do Havaí, na mesma década. A rede ALOHA utilizava transmissão via rádio para conectar vários computadores em ilhas diferentes. Resumidamente, seu protocolo baseava-se na seguinte ideia: todos os computadores têm direito de transmissão a qualquer momento. Depois de transmitir a informação, o computador aguarda durante um período de tempo por uma resposta do receptor, confirmando o recebimento dos dados. Caso essa resposta não ocorra nesse tempo, o computador assume que outra estação transmitiu seus dados simultaneamente, causando a colisão desses dados e impossibilitando a estação de recepção de compreender a informação. O computador, então, retransmite seus pacotes e novamente espera pelo sinal de recebimento. Esse sistema, entretanto, possuía um defeito: com o aumento do tráfego de informações, ocorria também o aumento de colisões, que por sua vez fazia as estações repetirem suas transmissões, aumentando ainda mais o tráfego de dados.

A Ethernet nasceu com o propósito de criar uma rede de comunicação entre várias estações, com o sistema ALOHA melhorado, tornando arbitrário o acesso ao canal de comunicação comum. Além de detectar ocorrências de colisões, o sistema também ouve o canal a procura de outras transmissões acontecendo antes de transmitir, além de suportar o acesso de múltiplas estações ao canal de comunicação. Por essas razões, o protocolo de acesso Ethernet é chamado CSMA/CD (*Carrier Sense Multiple Access with Collision Detect*).

Posteriormente, com o aumento da utilização do padrão Ethernet, o comitê do Instituto de Engenheiros Eletricistas Eletrônicos (IEEE) criou a norma IEEE802.3 para utilização dos padrões de Ethernet em equipamentos eletrônicos.

Existem diferentes alternativas de tecnologia Ethernet, de acordo com as características dos cabos de transmissão utilizados, assim como o alcance de transmissão e velocidades.

Em resumo, as tecnologias podem ser vistas na Tabela 4.1:

TABELA 4.1 TECNOLOGIAS ETHERNET				
Sigla	Denominação	Cabo	Velocidade	Alcance
10Base2	Ethernet fino (*thin* Ethernet)	Cabo coaxial (50 Ohms) de pequeno diâmetro	10 Mb/s	185m
10Base5	Ethernet espesso (*thick* Ethernet)	Cabo coaxial de diâmetro grosso	10 Mb/s	500m
10Base-T	Ethernet *standard*	Par trançado (*Twisted pair*)	10 Mb/s	100m
100Base-TX	Ethernet rápido (*Fast* Ethernet)	Duplo par trançado	100 Mb/s	100m
100Base-FX	Ethernet rápido (*Fast* Ethernet)	Fibra óptica multimodo do tipo (62.5/125)	100 Mb/s	2 km
1000Base-T	Ethernet Gigabit	Duplo igual entrançado	1000 Mb/s	100m
1000Base-LX	Ethernet Gigabit	Fibra óptica monomodo ou multimodo (*Long*)	1000 Mb/s	550m
1000Base-SX	Ethernet Gigabit	Fibra óptica multimodo (*Short*)	1000 Mbit/s	550m
10GBase-SR	Ethernet 10Gigabit	Fibra óptica multimodo	10 Gbit/s	500m
10GBase-LX4	Ethernet 10Gigabit	Fibra óptica multimodo	10 Gbit/s	500m

4.1 IEEE 802.3

O Instituto de Engenheiros Eletricistas Eletrônicos (IEEE) é uma organização profissional formada com o objetivo principal de estabelecer padrões para formatos de computadores e dispositivos. Dentre suas muitas publicações, o padrão IEEE 802 define as camadas físicas (as tecnologias de transmissão dos hardwares de rede, os modos de transmissão, o formato dos pacotes, as características de transmissão, entre muitos outros) e a subcamada MAC (*Media Access Control*) de uma rede Ethernet.

Dentre os sub-padrões do IEEE 802, está o IEEE 802.3, que especifica as camadas de ligação dos dados do Modelo OSI (Interconexão de Sistemas Abertos), visando facilitar a interconectividade entre máquinas de diferentes fabricantes para a administração de redes locais.

4.2 INET-X

Recentemente, soluções para redes de Instrumentação de Ensaios em Voo (*Flight Test Instrumentation* – FTI) utilizam cada vez mais sistemas com tecnologia Ethernet. A iniciativa iNET (*Integrated Network Enhanced Telemetry*) busca atender aos requisitos de FTI em relação a administração de sistemas, sincronização de tempo, transmissão de dados, entre outros. Tal padronização também contribui para o aumento da flexibilidade no desenvolvimento de sistemas, além a maior compatibilidade e interoperabilidade de equipamentos de diferentes empresas.

O iNET-X é uma expansão dos padrões iNET, garantindo o aumento de performance, coerência entre redes e facilidade de administração e instalação.

Algumas características importantes do padrão iNET-X:

- Todos os equipamentos e interfaces que utilizam padrões iNET-X devem estar de acordo com os padrões de Ethernet IEEE 802.3-2005;
- É necessária também a utilização da interface MAC (*Medium Access Control*), e que cada equipamento de rede tenha um único endereço MAC;
- Os *frames* de Ethernet devem ter tamanho entre 64 Bytes e 1581 Bytes;
- Uso de *Internet Protocol* (IP) versão 4;
- *User Datagram Protocol* (UDP) deve ser utilizado para transmissão em tempo real;
- Todos os pacotes devem ser transmitidos em modo *multicast*, usando protocolo UDP, quando existir mais de um consumidor para o dado enviado;
- Para os equipamentos, deve-se escolher o número de Porta de Destino de modo a garantir a não utilização de portas reservadas;
- Para serviços de rede e de sincronização, deve-se utilizar o protocolo IEEE 1588 *Precision Time Protocol* (PTP), para sincronizar todas as informações de tempo de todos os pacotes gerados em equipamentos diferentes;
- Para garantir consistência e interoperabilidade, todas as estampas de tempo devem usar o padrão de formato PTP;
- Pacotes iNET-X são encapsulados com cabeçalho IENA ou iNET e transmitidos como um pacote UDP/IP.

Para a aquisição de dados:

- Os dados não podem sofrer atrasos de mais de 50ns na fonte;
- Devem ser transmitidos em *multicast* utilizando formato de *Streams*. (contínuos)
- Cada *Stream* deve ter um Identificador de *Stream* único.
- Os dados do cabeçalho dos pacotes são recebidos em formato de Bytes de rede, ou seja, formato *big-endian* (os bytes mais significativos são armazenados nos menores endereços, de modo decrescente em relação ao seu peso numérico).

4.2.1 FORMATOS DE ARQUIVOS PCAP

Arquivos PCAP (*Packet Capture*) são arquivos de captura de tráfego de rede. Eles possuem características especiais que permitem a leitura e localização de cada pacote de rede através de aplicativos especializados, como o Wireshark, por exemplo. A Microsoft Windows possui uma biblioteca, chamada WinPcap, utilizada para programação de aplicativos que transmitem ou recebem pacotes de rede.

Os pacotes iNET-X, são capturados durante sua transmissão em arquivos no formato PCAP. O dado desejado, encontra-se na região de dados do iNET-X, que por sua vez possui encapsulamentos dos protocolos de transmissão necessários.

A estrutura desse pacote PCAP está definida na Figura 4.1.

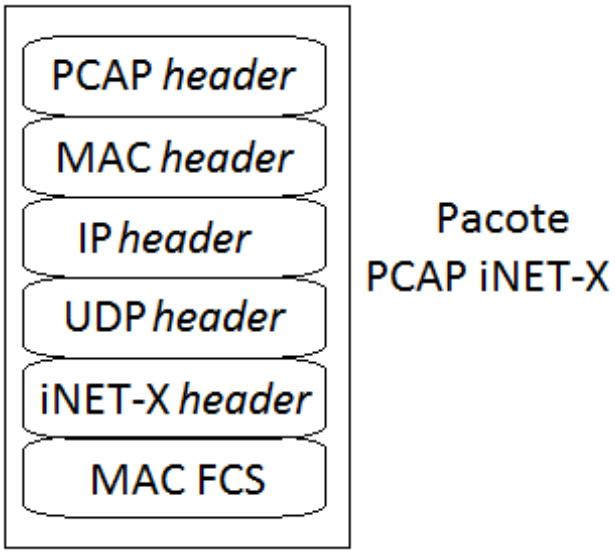

FIGURA 4.1 ESTRUTURA DE UM PACOTE iNET-X EM UM ARQUIVO PCAP

4.2.1.1 PCAP *Header* (16 Bytes)

O cabeçalho PCAP (veja Figura 4.2) inclui de informações de estampa de tempo do pacote (sincronizada ao PTP) e de tamanho do pacote.

4.2.1.2 MAC *Header* (14 Bytes)

Este cabeçalho traz informações do endereço MAC dos equipamentos de destino e fonte do pacote (veja Figura 4.2). Como previsto na norma, esses endereços são únicos para cada equipamento de rede.

Aqui, tem-se também a informação que indica o protocolo de encapsulamento dentro do *frame* Ethernet. Para esse trabalho, todos os pacotes possuíam protocolo IP versão 4, indicado pelo valor 0x0800 (em hexadecimal) do Ether Type.

Dentro do *payload* MAC, tem-se o pacote de fato enviado pelo equipamento. Esse *payload* MAC deve ter ao menos 46 Bytes. Caso contrário, o *frame* é complementado de modo a ter tamanho mínimo de 64 Bytes, requerido para o *frame* Ethernet.

Ao fim dos dados do pacote, existem 4 Bytes CRC (*Cyclic Redundancy Check*) calculados sobre o *frame* inteiro, para verificação de pacotes corrompidos, formando o *Frame Check Sequence* (FCS).

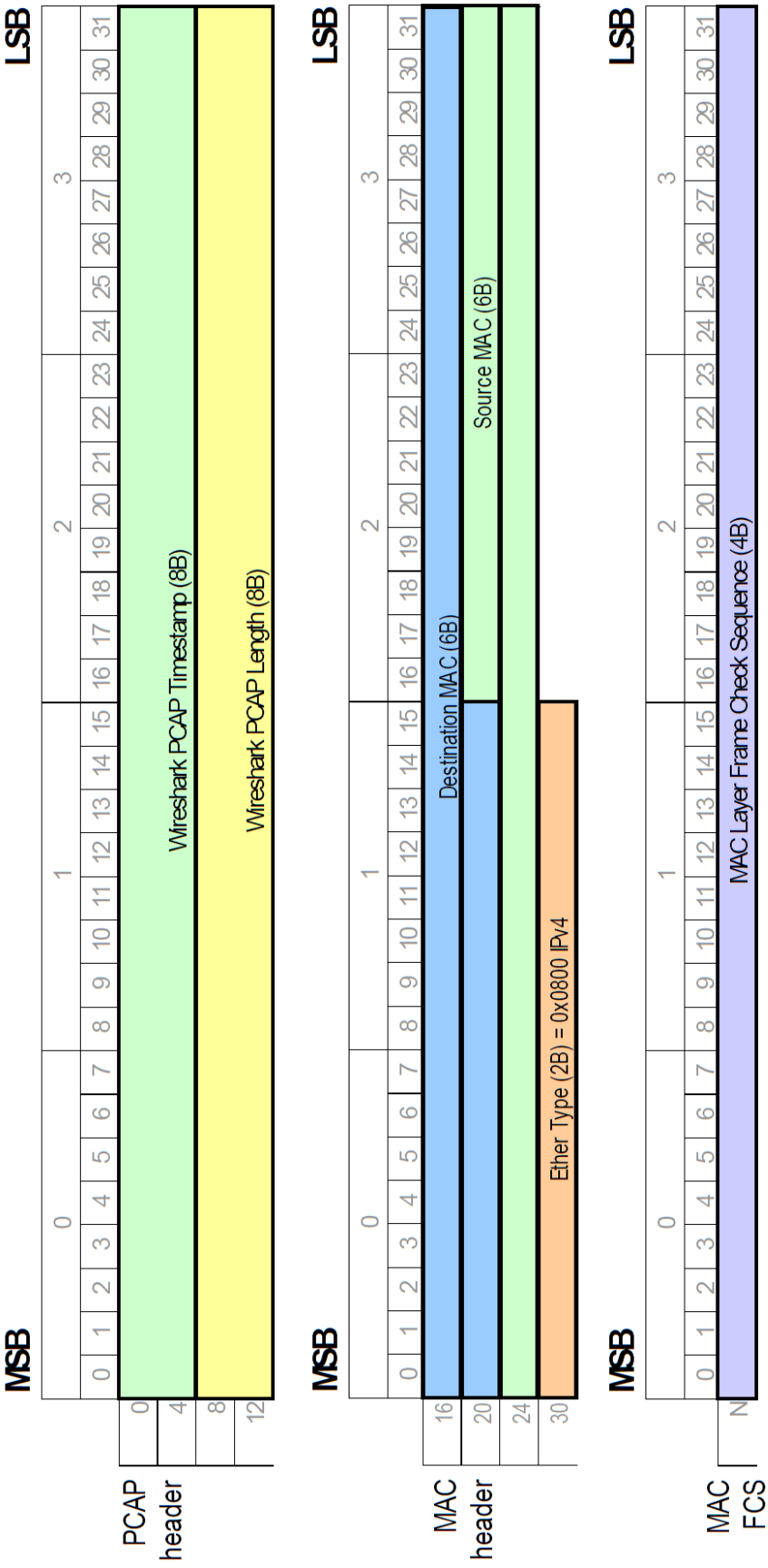

FIGURA 4.2 ESTRUTURAS DO PCAP HEADER, MAC HEADER E MAC FCS (FONTE: [7])

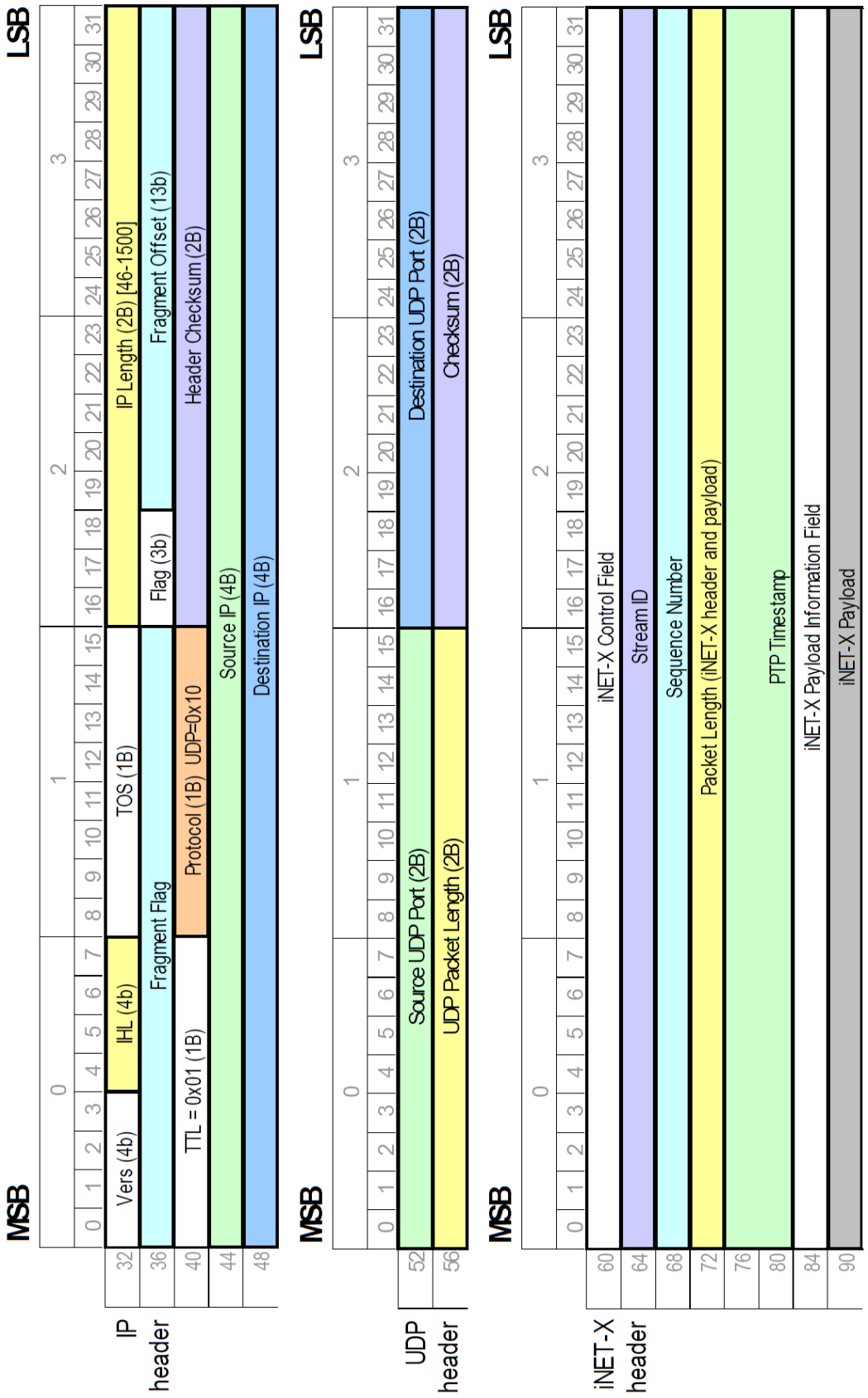

FIGURA 4.3 ESTRUTURAS DO IP HEADER, UDP HEADER E INET-X HEADER (FONTE: [7])

4.2.1.3 IP *Header* (20 Bytes)

O cabeçalho IP (Figura 4.3) possui as seguintes informações:

- Versão (4bits): Versão do IP. Para IPv4, esse valor deve ser 4;
- *Internet Header Lenght* (IHL) (4 bits): indica o número de palavras de 32 bits contidas no cabeçalho, indicando o tamanho deste. O valor mínimo desse campo é 5 (5x32 = 160 bits = 20 Bytes) e o máximo é 15;
- *Type of Service* (TOS) (8 bits): indica a qualidade de serviço de acordo com os bits iguais a um e aqueles em zero (0-2: *Precedence*, 3: *Latency Critical*, 4: *High Throughput*, 5: *Reliability Critical*, 6: *Best-Effort* e 7: *Never defined*);
- *Total Length* (16 bits): define o tamanho do pacote IP, incluindo cabeçalho e dados. O valor mínimo deve ser 20 Bytes (20 Bytes *header* + 0 Bytes dados) e o máximo é limitado pelo tamanho da palavra de 16 bits (65 535 em decimal);
- *Fragment Identifier* (16 bits): identifica fragmentos de um pacote IP original;
- *Fragment Flag* (3 bits): controla e identifica fragmentos, informando se é necessária a desfragmentação do pacote.
- *Fragment Offset* (13 bits): indica o número do fragmento;
- *Time to Live* (8 bits): limita o tempo de vida de um pacote, impedindo que este seja retransmitido mais de uma vez, em círculos. Caso o roteador pegue um pacote com o tempo 'expirado', este será descartado ou enviado de volta ao transmissor de onde veio;
- *Protocol* (8 bits): define o protocolo utilizado no campo de dados do IP (chamado *datagram*). Para protocolo UDP, esse valor é 0x10;
- *Header Checksum* (16 bits): O tamanho do *header* deve ser comparado ao valor desde campo para verificação de pacotes corrompidos;
- *Source IP Address* (32 bits): endereço IP do equipamento que transmitiu o pacote. Note que esse pode não representar o verdadeiro gerador do pacote;
- *Destination IP Address* (32 bits): endereço IP do equipamento que recebeu o pacote.

Depois do cabeçalho IP tem-se a região de seus dados, *datagram*. Nesta região, encontra-se o pacote UDP.

4.2.1.4 UDP *Header* (8 Bytes)

Na camada de transporte, tem-se informações de como o dado é transmitido. O TCP (*Transport Control Protocol*) e o UDP (*User Datagram Protocol*) são dois métodos de transmissão, sendo o segundo mais leve e simples.

O protocolo UDP (Figura 4.3) não requer nenhuma conexão estabelecida para que o dado seja transmitido, isto é, não há garantias de seu recebimento. Entretanto, este é o protocolo utilizado em telemetria, pois a garantia de recebimento deve ser realizada pelos equipamentos receptores, que devem ser confiáveis.

O cabeçalho UDP possui as seguintes informações:

- *Source/Destination UDP Port* (16 bits cada): número da porta de transmissão e recebimento dos pacotes. Tais números devem ser bem conhecidos e definidos, de modo a não se utilizar portas reservadas para outros sistemas e evitar erros;
- *Length* (16 bits): tamanho de todo o *datagram* UDP, incluindo cabeçalho e dados. O tamanho mínimo é 8 Bytes (8 Bytes *header* + 0 Bytes dados). Note que, embora o limite teórico desse valor seja 65 535 (8 Bytes UDP *header* + 65 527 Bytes dados). Deve-se levar em consideração o limite do pacote IP. Logo, esse valor é na verdade 65 515 (onde 65 535 é o tamanho máximo do pacote IP = 20 Bytes IP *header* + 8 Bytes UDP *header* + 65 507 Bytes dados);
- *Checksum* (16 bits): utilizado para verificação de erros de cabeçalho e dados. Esse campo é opcional para IPv4, devendo ser deixado com valor nulo se não for utilizado.

No *payload* do pacote UDP, encontra-se finalmente o pacote iNET-X.

4.2.1.5 iNET-X *Header* (28 Bytes)

O pacote de informações construído no sistema de telemetria é encapsulado no pacote iNET-X e preparado para o envio. Os valores dos campos do *header* iNET-X são controlados pelo programa que os envia, enquanto os demais cabeçalhos exteriores são controlados pelo sistema operacional.

A estrutura desse cabeçalho, como pode ser vista na Figura 4.3 e na Figura 4.4 em maiores detalhes é a seguinte:

- *iNET control field* (32 bits): os primeiros 32bits do cabeçalho são para controle do iNET, indicando:
    - Versão (4 bits): versão da aplicação iNET. Esse campo é utilizado com valor 1;
    - *Option Word Count* (4 bits): valor fixado em 0x1, pois o iNET-X utiliza uma única palavra de 32 bits definida como iNET-X *Payload Information Field*;
    - *Reserved* (8 bits): não utilizado, reservado para uso futuro;
    - *Message Flags* (32 bits): contém informações de fragmentação, sincronização de tempo, etc;
- *Stream ID* (32 bits): valor do *Stream ID* do pacote, identificando de maneira única o tipo de pacote e das informações que carrega. Essa será uma das mais importantes informações contidas no iNET-X *header*, pois através dela identifica-se a fonte do pacote, se ele traz informações de vídeo, PCM, GPS, entre outros. Para cada tipo de arquivo e tipo de equipamento, o sistema de Telemetria definiu *Stream IDs* específicos, com as seguintes características:

    - Dígito 1: tipo do pacote (Arinc-429, PCM, TTP, RS-232, etc);
    - Digito 2: sistema de aquisição (identificação do gravador);
    - Digito 3 e 4: número do chassi do equipamento;
    - Dígito 5: *slot* do equipamento;
    - Dígito 6 e 7: canal.
- *Sequence Number* (32 bits): número de sequência do pacote iNET-X. Seu valor é inicialmente zero e incrementa de um em um a cada pacote gerado e transmitido. Esse número será utilizado para verificação de perda de pacotes no programa.
- *Packet Length* (64 bits): tamanho total do pacote iNET-X, incluindo o cabeçalho.
- *PTP timestamp* (64 bits): estampa de tempo associada ao tempo do mais antigo dado existente no *payload*. Essa informação é dividida em duas:
    - *Time* (*seconds*): segundos contados desde 1º de Janeiro de 1970;
    - *Time* (*nanoseconds*): nano segundos, contados desde o início da contagem dos segundos;
- *iNET-X Payload Information Field* (32 bits): contém informações dos dados contidos no *payload*:
    - *Error Bit* (1bit): indica a existência ou não de erros associados ao dado, definindo a integridade do *payload*;
    - *Lost count* (4bits): numero de pacotes perdidos devido à saturação do *buffer* de transmissão do pacote. Essa perda de pacotes ocorre quando a velocidade de geração dos pacotes é maior que a velocidade em que estes são transmitidos.
    - *TimeOut* (TO) (1bit): indica se o pacote iNET-X é gerado e transmitido qualdo ocorre um evento de timeout (1) ou como resultado de um *buffer* FIFO (*First In First Out*) (0);
    - Os demais bits dessa estrutura são reservados para utilizações futuras.

A região de dados do iNET-X possui as informações geradas e transmitidas pelo sistema embarcado da aeronave. Para cada tipo de informação, o pacote iNET-X pode assumir diferentes formatos de *payload*, que podem ter seus próprios cabeçalhos. Alguns dados, por exemplo, utilizam vários *minor frames* dentro de um único payload, e estes devem ter informações de tempo para cada *minor frame*, tamanho desses *frames*, quantidade de *minor frames* por pacote.

Embora existam outras tantas características dos pacotes iNET-X, não é o objetivo desses trabalho abordá-las. Para maiores informações, é recomendável a leitura dos arquivos disponibilizados pela *Curtiss Wright Controls*, empresa de equipamentos eletrônicos para sistemas de telemetria e aeronaves.

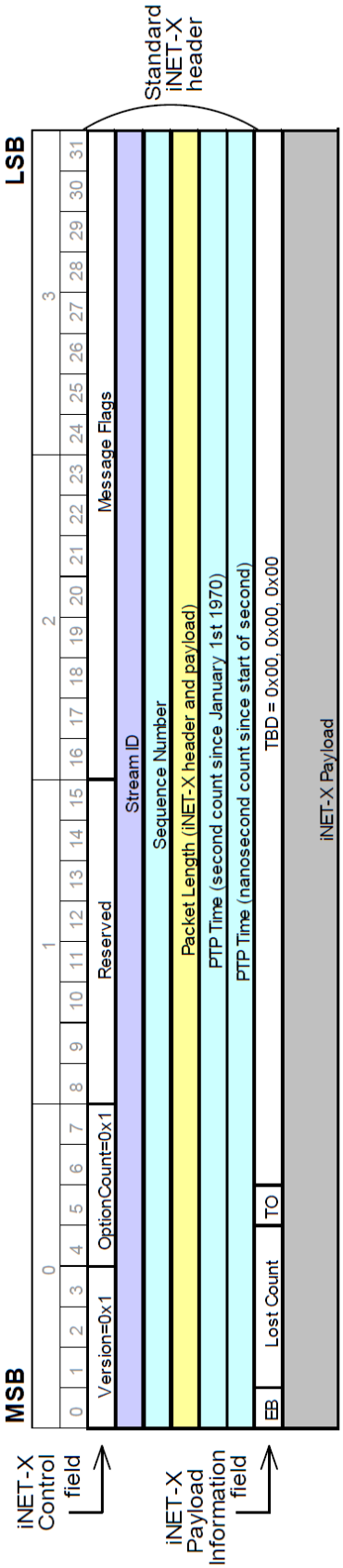

FIGURA 4.4 ESTRUTURA DO CABEÇALHO iNET-X EM MAIORES DETALHES (FONTE: [7])

4.2.1.6 iNET-X Parser Aligned

Dentre os muitos formatos iNET-X, foi utilizado para transmissão dos pacotes de informações de GPS o formato iNET-X *Parser Aligned (*Figura 4.5*)*. Este formato permite que, dentro do bloco de dados do pacote, existam vários blocos de informações parciais, ditas *Parser Blocks*.

Cada *Parser Block* possui seu próprio cabeçalho, com informações sobre número e tamanho de mensagem ou códigos de erros, como pode ser visto a seguir:

- *Parser Information Word* (4 Bytes): contém informações sobre o status da mensagem:
  - *Er* (1 bit): indica que um erro ocorreu
  - *Error Code* (5 bits): código do erro ocorrido
  - *Quad Bytes* (9 bits): número de *quad bytes*, relacionado ao tamanho de toda informação transmitida, incluindo *Elapsed Time*, e a própria *Parser Information Word*. Equivale ao número de *Words* (16 bits) contidas no bloco.
  - *Message Count* (8 bits): contador de mensagens. Esse valor começa em zero, incrementando de um em um a cada bloco e tem valor máximo 255, quando recomeça a contagem. Esse número conta a quantidade de blocos totais e não de quantos existem por pacote.
  - *TBD* (4 bits): reservado para usos futuros.
  - *Bus ID* (4 bits): indica o número do canal de transmissão. Para o sistema de telemetria, esse valor equivale ao último número do *Stream ID* do pacote.
- *Elapsed Time* (4 Bytes): estampa de tempo, em nano segundos, adicionada ao tempo PTP existente no cabeçalho iNET-X. Esse tempo sempre é zero para o primeiro bloco do pacote e incrementa à medida que os blocos são criados.
- *Message Data* (N x 4 Bytes): mensagem a ser enviada. Para alguns tipos de pacotes, deve ter tamanho múltiplo de 4 Bytes, sendo complementada até atingir o tamanho desejado.

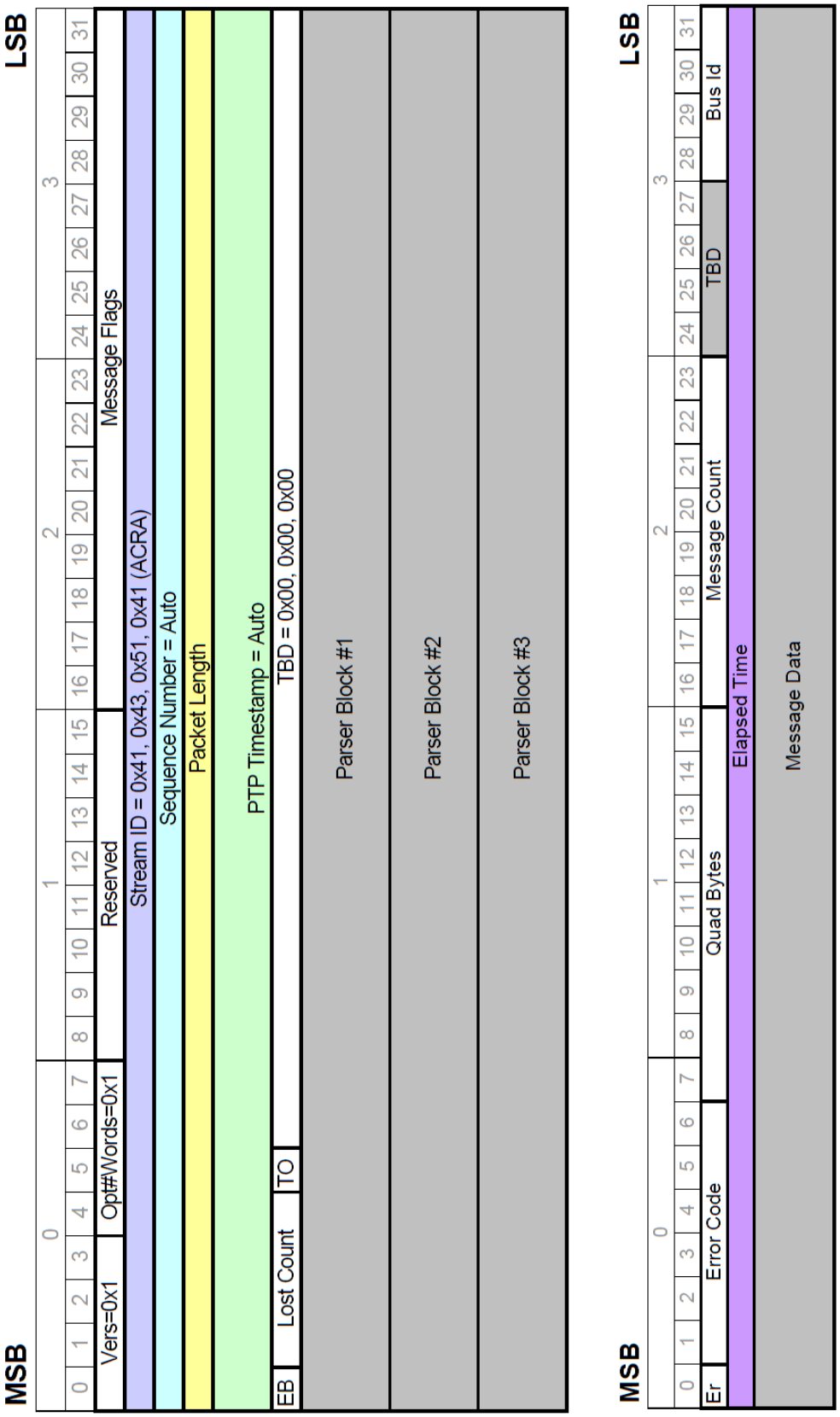

FIGURA 4.5 EXEMPLO DE ESTRUTURA DA MENSAGEM INET-X PARSER ALIGNED COM TRÊS BLOCOS E CABEÇALHO DE UM PARSER BLOCK (FONTE: [7])

Capítulo 5. *GLOBAL POSITIONING SYSTEM* (GPS)

5.1 INTRODUÇÃO HISTÓRICA

O Sistema de Posicionamento Global, ou GPS, é um sistema de navegação baseado na utilização de satélites, desenvolvido pelo Departamento de Defesa dos Estados Unidos no início dos anos 70. O GPS foi desenvolvido inicialmente para utilização militar e só mais tarde foi disponibilizado para uso civil.

O GPS fornece informações contínuas de tempo e posicionamento em qualquer lugar do mundo, sob quaisquer condições climáticas.

5.2 O SISTEMA

O GPS pode ser subdividido em três seguimentos: o seguimento espacial, seguimento de controle e, por fim, o seguimento de utilização.

5.2.1 SEGUIMENTO ESPACIAL

O seguimento espacial do GPS consiste basicamente em uma constelação de 24 satélites em órbita praticamente circular ao redor da Terra (veja a Figura 5.1). Tais satélites estão estrategicamente posicionados em cada um dos seis planos orbitais. Essa disposição garante que, de qualquer lugar da terra, um receptor de GPS esteja visível por quatro a dez satélites. Para esse receptor ter sua posição definida, ele deve ter contato com quatro satélites, no mínimo, como será explicado no tópico 5.2.3.

Cada satélite transmite um sinal, utilizado principalmente para calcular a distância do receptor de GPS ao satélite. Esse sinal também contém as coordenadas do satélite em função do tempo. Os sinais transmitidos são controlados por relógios atômicos extremamente precisos.

FIGURA 5.1 EXEMPLIFICAÇÃO DO POSICIONAMENTO DOS SATÉLITES DE GPS NOS PLANOS ORBITAIS DA TERRA

5.2.2 SEGUIMENTO DE CONTROLE

Este consiste em uma rede mundial de estações de controle, cuja principal (*Master Control Station* - MCS) está localizada em Colorado, nos Estados Unidos. A função do sistema de controle operacional de determinar e predizer a localização de cada satélite, a integridade do sistema, entre outros. Essas informações são posteriormente enviadas para os satélites.

5.2.3 SEGUIMENTO DE UTILIZAÇÃO

O terceiro seguimento consiste na utilização militar e civil do GPS. Por ser utilizado por um número ilimitado de civis e também para finalidades militares, o sistema apenas fornece informações para os usuários, sem a possibilidade de comunicação no sentido oposto. Com um receptor GPS conectado a uma antena GPS, o usuário recebe sinais de GPS, que podem ser usados para determinar sua posição em qualquer lugar do planeta.

Para localizar o receptor, o sistema utiliza o sinal de três satélites GPS. A teoria do sistema é baseada no método de triangulação. Cada satélite envia um sinal para o receptor. No sinal recebido, existe, entre outras, a informação do tempo exato em que esse sinal foi enviado. O receptor, então, calcula o intervalo de tempo para informação chegar até ele, pois conhece o instante em que recebeu o sinal.

Sabendo que velocidade com que esse sinal viaja, é a velocidade da luz, pois o sinal recebido é uma onda eletromagnética, é possível então calcular a distância entre o satélite de GPS e o receptor, ao multiplicar o intervalo de tempo pela velocidade da luz (aproximadamente 300 000 km/s).

Na Figura 5.2, o satélite 1 envia os sinais para o receptor (P) e esse, por sua vez, calcula a distância (R1) que se encontram o satélite e o receptor. Assim, o Ponto P poderia ser qualquer ponto da circunferência, sendo, por isso, necessário, outro satélite. O satélite 2 vai repetir o processo, encontrando a distância (R2).

Estas duas circunferências se encontram em dois pontos, o que faz com que seja necessária a presença de um terceiro satélite. Com o satélite 3, é descoberta a distância (R3), o que faz com que estas três circunferências se interceptem em apenas um ponto (P). Esse ponto é onde o receptor está localizado.

Esse método só tem validade se o intervalo de tempo calculado estiver correto. Para isso, os relógios atômicos presentes nos satélites têm que estar sincronizados. Assim, na prática, é necessária a presença de um quarto satélite que serve de referência aos outros satélites. É o chamado satélite de referência.

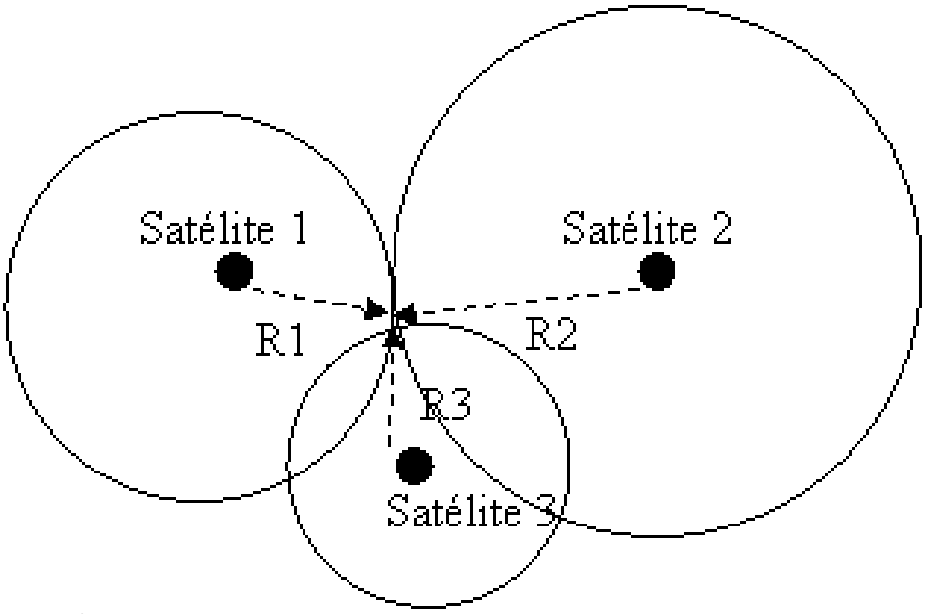

FIGURA 5.2 MÉTODO DE TRIANGULAÇÃO UTILIZADO PARA LOCALIZAÇÃO DO RECEPTOR DE GPS

5.3 DEFINIÇÃO DE MENSAGENS DE GPS

O receptor de GPS Novatel será utilizado no projeto. Dentre suas muitas funções, há três principais tipos de mensagens que ele transmite via comunicação serial, todas em ASCII e com as informações separadas por vírgulas.

5.3.1 ESTRUTURA DO CABEÇALHO ASCII

Todas as mensagens possuem um cabeçalho ASCII. Essa estrutura possui informações a respeito do tipo de mensagem, *status* do receptor, informação de tempo em segundos, entre outros.

A Tabela 5.1 mostra a estrutura desse cabeçalho.

TABELA 5.1 ESTRUTURA DO CABEÇALHO DAS MENSAGENS DE GPS (ADAPTADO DA FONTE: [8])

Field #	Field Name	Field Type	Description
1	Sync	Char	Sync character. The ASCII message is always preceded by a single '#' symbol.
2	Message	Char	This is the ASCII name of the
3	Port	Char	This is the name of the port from which the log was generated. The string is made up of the port name followed by an _x where x is a number from 1 to 31 noting the virtual address of the port. If no virtual address is indicated, it is assumed to be address 0
4	Sequence #	Long	This is used for multiple related logs. It is a number that counts down from N-1 to 0 where 0 means it is the last one of the set. Most logs only come out one at a time in which case this number is 0.
5	% Idle Time	Float	The minimum percentage of time that the processor is idle between successive logs with the same Message ID.

6	GPS Time Status	Enum	This value indicates the quality of the GPS time
7	Week	Ulong	GPS week number.
8	Seconds	GPSec	Seconds from the beginning of the GPS week accurate to the millisecond level.
9	Receiver Status	Ulong	This is an eight digit hexadecimal number representing the status of various hardware and software components of the receiver between successive logs with the same Message ID
10	Reserved	Ulong	Reserved for internal use.
11	Receiver s/w Version	Ulong	This is a value (0 - 65535) that represents the receiver software build number.
12	;	Char	This character indicates the end of the header.

5.3.2 BEST AVAILABLE VELOCITY DATA (BESTVEL)

Esse *log* contém a melhor informação de velocidade disponível computada pelo receptor. Em adição, é reportado o status da informação fornecida, indicando se o dado recebido é válido. A Tabela 5.2 mostra as características dessa mensagem.

TABELA 5.2 ESTRUTURA DA MENSAGEM BESTVEL DE GPS (ADAPTADO DA FONTE: [8])

Field #	Field type	Data Description	Format	Binary Bytes	Binary Offset
1	header	Log header		H	0
2	sol status	Solution status	Enum	4	H
3	vel type	Position type	Enum	4	H+4
4	latency	A measure of the latency in the velocity time tag in seconds. It should be subtracted from the time to give improved results. seconds. It should be subtracted from the time to give improved results.	Float	4	H+8
5	age	Differential age in seconds	Float	4	H+12
6	hor spd	Horizontal speed over ground, in meters per second	Double	8	H+16
7	trk gnd	Actual direction of motion over ground (track over ground) with respect to True North, in degrees	Double	8	H+24
8	vert spd	Vertical speed, in meters per second, where positive values indicate increasing altitude (up) and negative values indicate decreasing altitude (down)	Double	8	H+32
9	Reserved		Float	4	H+40
10	xxxx	32-bit CRC (ASCII and Binary only)	Hex	4	H+44
11	[CR][LF]	Sentence terminator (ASCII only)	-	-	-

Um exemplo da mensagem BESTVEL pode ser visto na Figura 5.3:

```
#BESTVELA,COM1,0,59.5,FINESTEERING,1643,221232.200,00000000,827b,4807;
SOL_COMPUTED,DOPPLER_VELOCITY,0.150,0.000,0.0031,302.568110,-0.0112,
0.0*6a9363e6
```

FIGURA 5.3 EXEMPLO DA MENSAGEM BESTVEL

A letra "A" ao final do nome do tipo da mensagem (em "BESTVELA"), indica que a mensagem está em ASCII.

5.3.3 BEST AVAILABLE CARTESIAN POSITION AND VELOCITY (BESTXYZ)

Essa mensagem contém a melhor informação de posição e velocidade calculadas pelo receptor.

A seguir, a Tabela 5.3 com as características dessa mensagem:

TABELA 5.3 ESTRUTURA DA MENSAGEM BESTXYZ DE GPS (ADAPTADO DA FONTE: [8])

Field #	Field type	Data Description	Format	Binary Bytes	Binary Offset
1	header	Log header		H	0
2	P-sol status	Solution status	Enum	4	H
3	pos type	Position type	Enum	4	H+4
4	P-X	Position X-coordinate (m)	Double	8	H+8
5	P-Y	Position Y-coordinate (m)	Double	8	H+16
6	P-Z	Position Z-coordinate (m)	Double	8	H+24
7	P-X σ	Standard deviation of P-X (m)	Float	4	H+32
8	P-Y σ	Standard deviation of P-Y (m)	Float	4	H+36
9	P-Z σ	Standard deviation of P-Z (m)	Float	4	H+40
10	V-sol status	Solution status	Enum	4	H+44
11	vel type	Velocity type	Enum	4	H+48
12	V-X	Velocity vector along X-axis (m/s)	Double	8	H+52
13	V-Y	Velocity vector along Y-axis (m/s)	Double	8	H+60
14	V-Z	Velocity vector along Z-axis (m/s)	Double	8	H+68
15	V-X σ	Standard deviation of V-X (m/s)	Float	4	H+76
16	V-Y σ	Standard deviation of V-Y (m/s)	Float	4	H+80
17	V-Z σ	Standard deviation of V-Z (m/s)	Float	4	H+84
18	stn ID	Base station identification	Char[4]	4	H+88
19	V-latency	A measure of the latency in the velocity time tag in seconds. It should be subtracted from the time to give improved results.	Float	4	H+92
20	diff_age	Differential age in seconds	Float	4	H+96
21	sol_age	Solution age in seconds	Float	4	H+100
22	#obs	Number of observations tracked	Uchar	1	H+104
23	#GPSL1	Number of GPS L1 ranges used in computation	Uchar	1	H+105
24	#L1	Number of GPS L1 ranges above the RTK mask angle	Uchar	1	H+106
25	#L2	Number of GPS L2 ranges above the RTK mask angle	Uchar	1	H+107
26	Reserved	Reservado para uso futuro	Char	1	H+108

27	Reserved	Reservado para uso futuro	Char	1	H+109
28	Reserved	Reservado para uso futuro	Char	1	H+110
29	Reserved	Reservado para uso futuro	Char	1	H+111
30	xxxx	32-bit CRC (ASCII and Binary only)	Hex	4	H+112
31	[CR][LF]	Sentence terminator (ASCII only)	-	-	-

Um exemplo da mensagem BESTXYZ pode ser visto na Figura 5.4:

```
#BESTXYZA,COM1,0,59.5,FINESTEERING,1643,221232.200,00000000,d821,4807;
SOL_COMPUTED,SINGLE,3935097.5886,-4432052.2202,-2350136.4080, 2.5321, 2.6056,
2.0889, SOL_COMPUTED,DOPPLER_VELOCITY, -0.0084,0.0056,0.0057,0.3026,0.3114,
0.2497 ,"",0.150,0.000,0.000,7,7,0,0,0,06,0,03*5bf60a84
```

FIGURA 5.4 EXEMPLO DA MENSAGEM BESTXYZ

5.3.4 OMNISTAR HP/XP POSITION (OMNIHPPOS)

Mensagem de informação de posição Extra Performance (XP) or High Performance (HP). A Tabela 5.4 mostra as características dessa mensagem:

TABELA 5.4 ESTRUTURA DA MENSAGEM OMNIHPPOS DE GPS

Field #	Field type	Data Description	Format	Binary Bytes	Binary Offset
1	header	Log header		H	0
2	sol status	Solution status	Enum	4	H
3	pos type	Position type	Enum	4	H+4
4	lat	Latitude	Double	8	H+8
5	lon	Longitude	Double	8	H+16
6	hgt	Height above mean sea level	Double	8	H+24
7	undulation	Undulation - the relationship between the geoid and the WGS84 ellipsoid (m)	Float	4	H+32
8	datum id#	Datum ID number	Enum	4	H+36
9	lat σ	Latitude standard deviation	Float	4	H+40
10	lon σ	Longitude standard deviation	Float	4	H+44
11	hgt σ	Height standard deviation	Float	4	H+48
12	stn id	Base station ID	Char[4]	4	H+52
13	diff_age	Differential age in seconds	Float	4	H+56
14	sol_age	Solution age in seconds	Float	4	H+60
15	#obs	Number of observations tracked	Uchar	1	H+64
16	#GPSL1	Number of GPS L1 ranges used in computation	Uchar	1	H+65
17	#L1	Number of GPS L1 ranges above the RTK mask angle	Uchar	1	H+66
18	#L2	Number of GPS L2 ranges above the RTK mask angle	Uchar	1	H+67
19	Reserved	Reservado para uso futuro	Uchar	1	H+68
20	Reserved		Uchar	1	H+69
21	Reserved		Uchar	1	H+70
22	Reserved		Uchar	1	H+71
23	xxxx	32-bit CRC (ASCII and Binary only)	Hex	4	H+72
24	[CR][LF]	Sentence terminator (ASCII only)	-	-	-

Um exemplo da mensagem OMNIHPPOS pode ser visto na Figura 5.5:

```
#OMNIHPPOSA,COM1,0,59.5,FINESTEERING,1643,221232.300,00000000,808d,4807;
INSUFFICIENT_OBS,NONE,-21.76152389064,-48.39900828691,615.2072,-6.7000,WGS84,
0.2033,0.6777,0.5813,"0", 0.000,0.000,7,0,0,0,0,00,0,00*55aed9b2
```

FIGURA 5.5 EXEMPLO DA MENSAGEM OMNIHPPOS

5.3.5 NOVO PADRÃO: OMNISTAR

Uma das funções do programa criado é a formação do novo tipo de mensagem, com padrão OmniSTAR, contendo informações predeterminadas dos três tipos de mensagens anteriores.

Esse novo padrão possui uma série de especificações a serem seguidos. Por exemplo, quando ocorre algum erro e faltam informações, devem-se identificar as mensagens perdidas por uma sequência de seis caracteres “x” no lugar em que a informação deveria estar. Essa nova mensagem não possui cabeçalho. Todas as colunas devem ser separadas por vírgulas e o caractere de fim de linha é o ponto e vírgula. Cada linha da mensagem corresponde a um instante de tempo. A coluna dos três arquivos corresponde ao tempo, deve ser feita uma correção para que fique menor do que 86400 segundos, descontando os múltiplos de 86400 do mesmo. Por exemplo: 178400 = 5600 (descontando duas vezes 86400).

Uma mensagem OmniSTAR é formada por um bloco contendo as mensagens BESTVEL, BESTXYZ e OMNIHPPOS, todas de um mesmo tempo.

Na Tabela 5.5 estão definidos os campos de informações dessa mensagem, assim como os mnemônicos utilizados para nomear os campos e a unidade (no Sistema Internacional – SI) da informação. Também está indicada a proveniência de tal informação.

TABELA 5.5 ESTRUTURA DA MENSAGEM OMNISTAR DE GPS

Campo	Mnemônico	Descrição	Fonte do dado (campo)	Unidade
1	PTIME	Tempo	Cabeçalho do bloco das três mensagens (8)	s
2	X	Position X-coordinate in ECEF system	BESTXYZ (4)	m
3	Y	Position Y-coordinate in ECEF system	BESTXYZ (5)	m
4	Z	Position Z-coordinate in ECEF system	BESTXYZ (6)	m
5	sig_X	Standard deviation of X	BESTXYZ (7)	m
6	sig_Y	Standard deviation of Y	BESTXYZ (8)	m
7	sig_Z	Standard deviation of Z	BESTXYZ (9)	m
8	VX	Velocity vector along X-axis	BESTXYZ (12)	m/s
9	VY	Velocity vector along Y-axis	BESTXYZ (13)	m/s
10	VZ	Velocity vector along Z-axis	BESTXYZ (14)	m/s
11	sig_VX	Standard deviation of VX	BESTXYZ (15)	m/s
12	sig_VY	Standard deviation of VY	BESTXYZ (16)	m/s

13	sig_VZ	Standard deviation of VZ	BESTXYZ (17)	m/s
14	lat	Latitude	OMNIHPPOS (4)	°
15	lon	Longitude	OMNIHPPOS (5)	°
16	hgt	Height	OMNIHPPOS (6)	m
17	sig_lat	Latitude standard deviation	OMNIHPPOS (9)	°
18	sig_lon	Longitude standard deviation	OMNIHPPOS (10)	°
19	sig_hgt	Height standard deviation	OMNIHPPOS (11)	m
20	hor spd	Horizontal speed over ground	BESTVEL (6)	m/s
21	trk gnd	Actual direction of motion over ground (track overground) with respect to True North	BESTVEL (7)	°
22	vert spd	Vertical speed, where positive values indicate increasing altitude (up) and negative values indicate decreasing altitude (down)	BESTVEL (8)	m/s

Um exemplo da mensagem OmniSTAR pode ser visto na Figura 5.6:

```
68882.4,3934819.3197,-4432242.5336,-2350220.5777,0.2670,0.1501,0.0914,-0.0086,
0.0080,0.0023 ,0.1194,0.0671,0.0409,-21.76237601748,-48.40223915569,610.6439,
0.0468,0.2591,0.1812,0.0025,207.100548,-0.0118
```

FIGURA 5.6 EXEMPLO DA MENSAGEM OMNISTAR DE GPS

Capítulo 6. SOFTWARES E FERRAMENTAS UTILIZADAS

6.1 INTRODUÇÃO AO MICROSOFT VISUAL STUDIO

Microsoft Visual Studio [9] é uma solução integrada que permite ao desenvolvedor criar aplicações e *softwares* para qualquer dispositivo que utilize o sistema operacional Microsoft Windows. Essa ferramenta utiliza programação em linguagem C, possui interface de fácil aprendizagem e ferramentas que facilitam o processo de criação de programas.

6.1.1 ÁREA DE DESENVOLVIMENTO DO CÓDIGO

O Microsoft Visual Studio possui uma grande quantidade de funções e menus para auxiliar o processo de criação do *software*. Todos os menus são facilmente customizáveis de acordo com a necessidade do desenvolvedor. É possível organizar as janelas do aplicativo de modo a deixar à vista as informações que se deseja verificar.

Na Figura 6.1, está um exemplo dessa janela, com um trecho do código em desenvolvimento (no centro). A janela lateral esquerda contém uma relação de todos os arquivos do projeto, código fonte, cabeçalhos, recursos, arquivos externos (que fazem parte da biblioteca do próprio Windows). E uma janela inferior, disposta desse modo para acompanhar também o desenvolvimento do projeto. Nessa figura, a janela inferior mostra todas as ocorrências de uma dada variável em todo o programa.

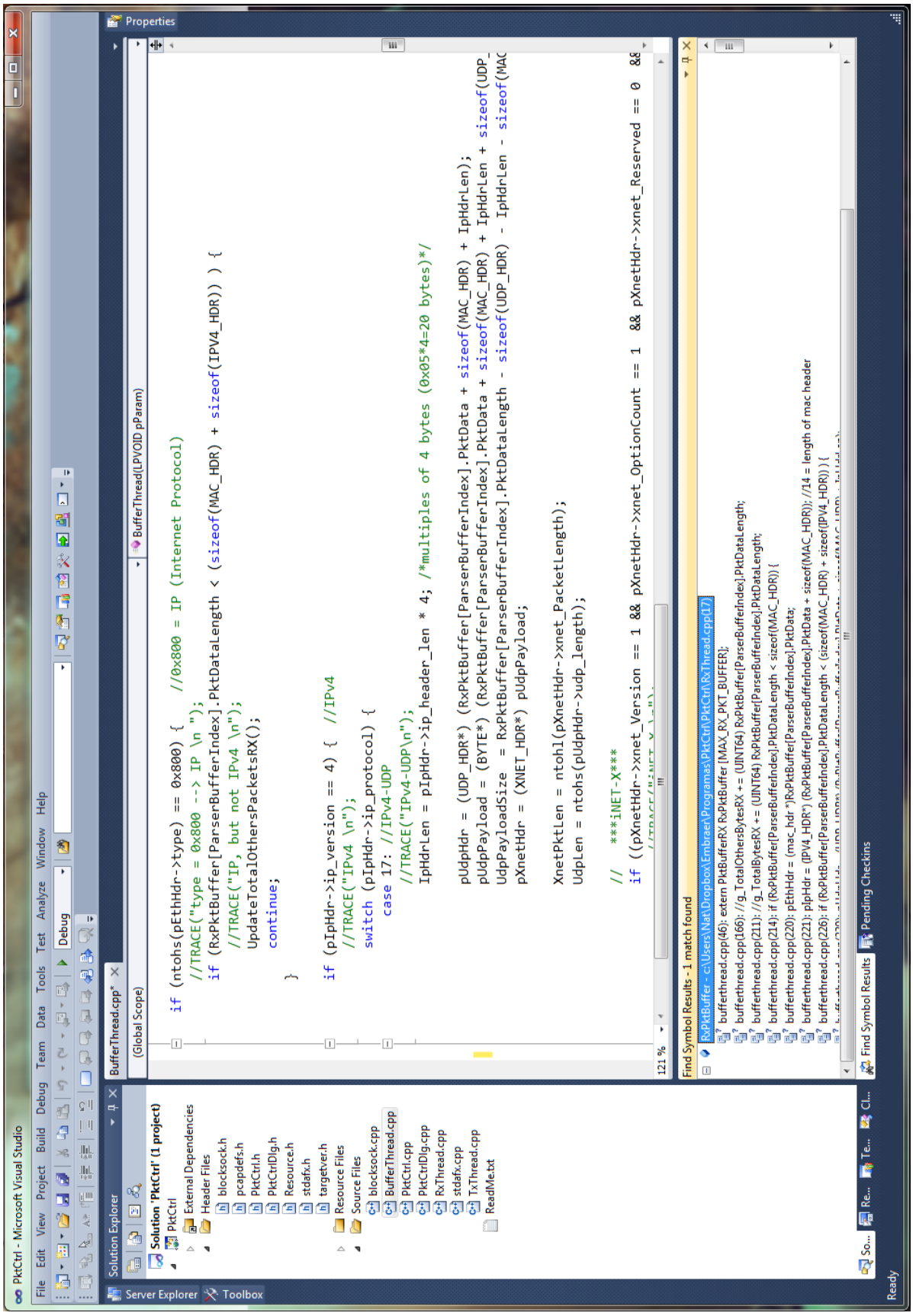

FIGURA 6.1 ÁREA DE TRABALHO DO VISUAL STUDIO

6.1.2 *EXECUÇÃO E ACOMPANHAMENTO DO CÓDIGO*

O Visual Studio também permite a execução controlada do programa, permitindo que o usuário acompanhe passo a passo cada uma das instruções executadas e também oferece a possibilidade do controle de execução através da utilização de *breakpoints* em pontos estratégicos de acordo com a necessidade do programador.

Na Figura 6.2 é possível ver um exemplo da utilização dessa importante ferramenta.

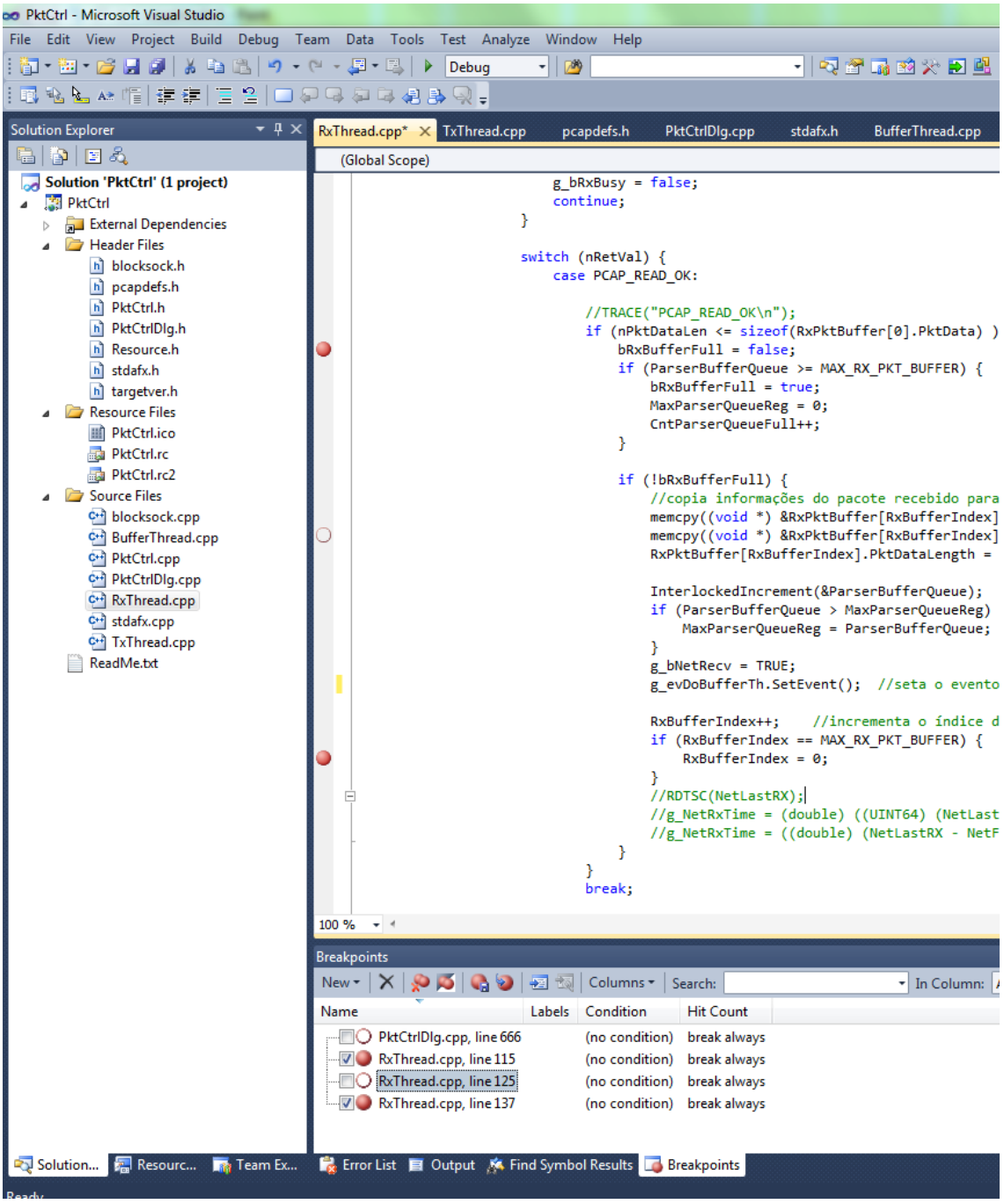

FIGURA 6.2 ÁREA DE TRABALHO DO VISUAL STUDIO, DURANTE A EXECUÇÃO DO PROGRAMA, COM UTILIZAÇÃO DO RECURSO DE BREAKPOINTS

Além da possibilidade de acompanhar a execução das instruções do programa, o desenvolvedor pode também utilizar o recurso *watch*, para visualizar valores de variáveis e *buffers* a sua escolha, durante o processamento. Graças a esses recursos, é possível localizar facilmente o momento em que uma variável é corrompida, por exemplo, ou verificar valores não desejados, auxiliando a correção de erros.

A Figura 6.3 mostra a utilização desse recurso.

6.1.3 CRIAÇÃO DO DIÁLOGO

Com essa poderosa ferramenta, também é possível a criação da janela de diálogo do aplicativo desenvolvido, onde o desenvolvedor tem acesso às ferramentas necessárias na criação e controle do diálogo. Aqui, é possível inserir botões, caixas de texto, listas, entre outros, além de realizar o controle de todo o comportamento do objeto, como este irá responder aos comandos do usuário e as variáveis relacionadas a ele.

A área de desenvolvimento dos recursos visuais também permite a criação do ícone do arquivo executável do aplicativo e controle da janela "*About*", que fornece informações da versão do aplicativo e dos direitos de cópia.

Todas as ferramentas seguem os padrões e comportamento daquelas disponíveis no sistema operacional Windows.

Na Figura 6.4, pode-se ver a janela de criação de diálogo do programa GpsCap.

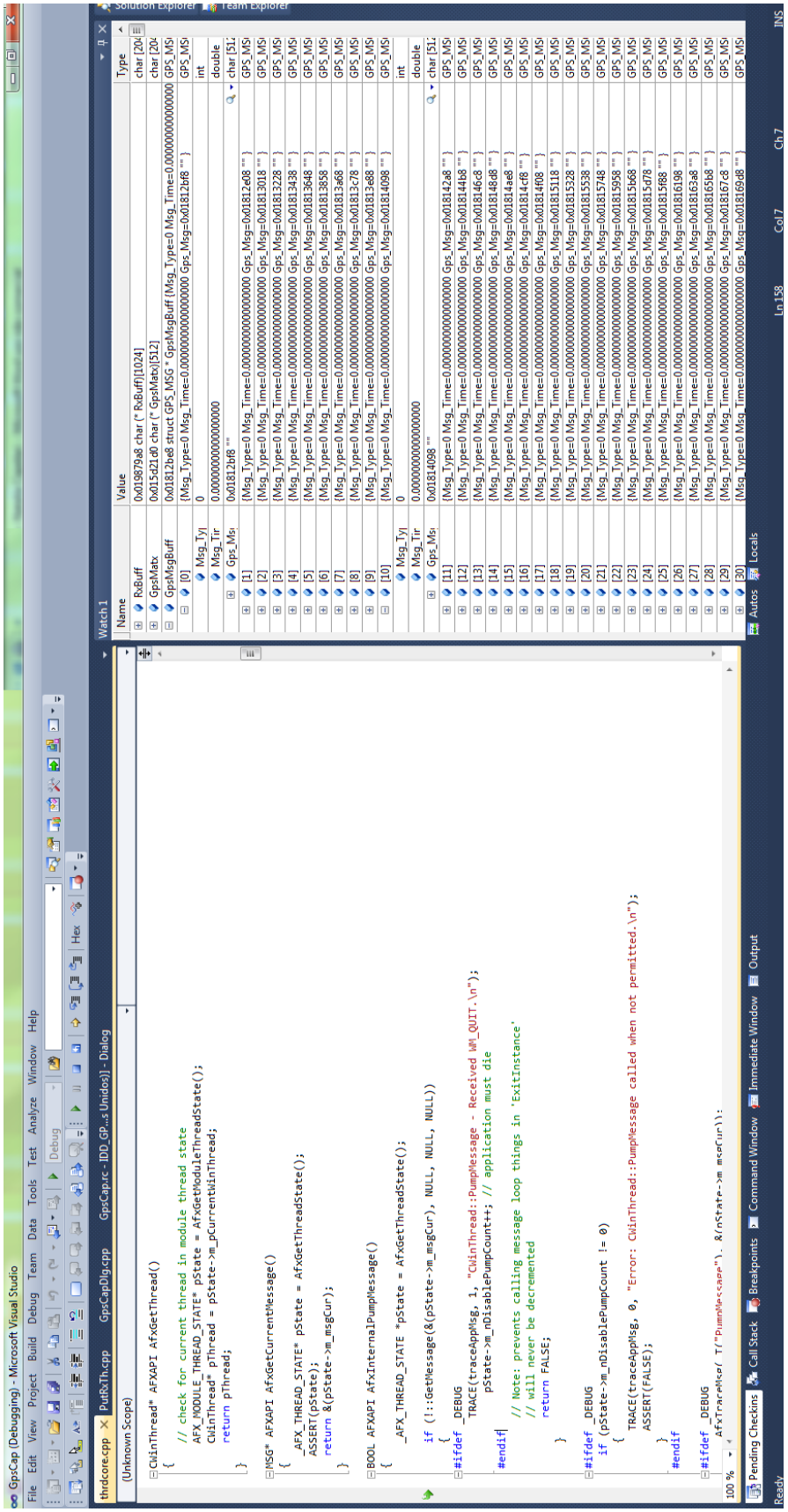

FIGURA 6.3 ÁREA DE TRABALHO DO VISUAL STUDIO, DURANTE A EXECUÇÃO DO PROGRAMA, COM UTILIZAÇÃO DO RECURSO DE WATCH PARA ACOMPANHAR O ESTADO DE UM *BUFFER*

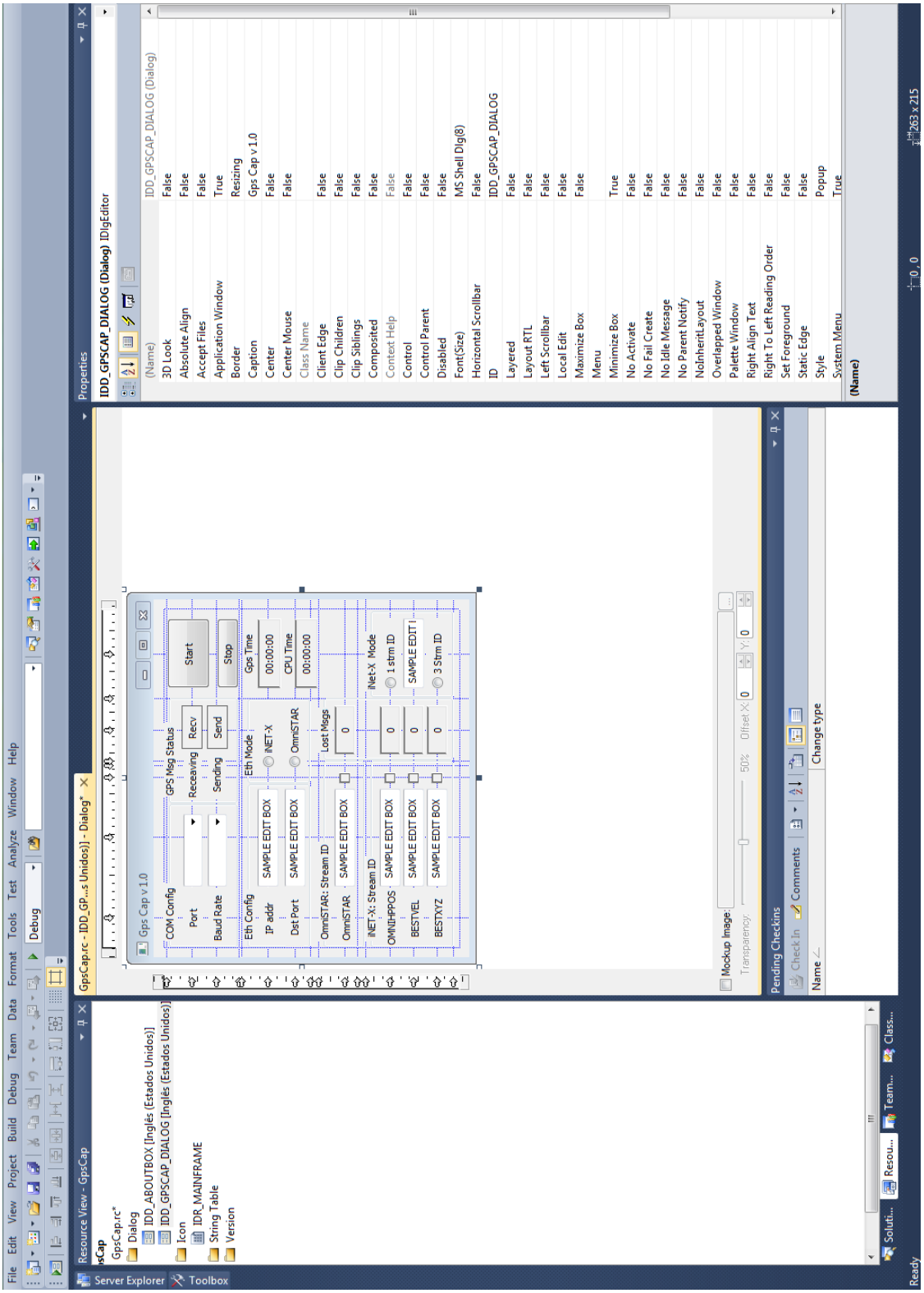

FIGURA 6.4 ÁREA DE DESENVOLVIMENTO DO DIÁLOGO DO APLICATIVO

6.2 WIRESHARK

Wireshark [10] é um analisador de protocolo de rede, que permite a captura e visualização do tráfego de uma rede de computador.

Esse programa fui utilizado inúmeras vezes durante o desenvolvimento dos aplicativos para receber e verificar os dados enviados pelo GpsCap e pelo PktControl.

O Wireshark possui várias extensões que podem ser instaladas no programa, permitindo o reconhecimento de outros protocolos de comunicação, como por exemplo, o reconhecimento de arquivos iNET-X. Desse modo, é possível visualizar em tempo real as informações contidas em um pacote iNET-X e os valores contidos em seu cabeçalho.

Esse aplicativo também permite filtrar os pacotes de rede por certo endereço IP, ou qualquer outra informação que possa ser relevante ao usuário, além de gravar o trafego em arquivos PCAP, permitindo posterior análise deste.

6.3 PCHECK (EMBRAER)

O programa PCheck é utilizado na verificação de arquivos PCAP de dados gravados durante os ensaios da aeronave. Com esse aplicativo é possível verificar a integridade dos arquivos, verificar perda de pacotes e também extrair pacotes PCM, GPS, CAN, entre outros, simplesmente conhecendo o *Stream ID* do dado desejado.

Um programa muito útil e veloz, que também foi atualizado para acompanhar os avanços do sistema de telemetria desse projeto, como será mais bem explicado no Capítulo 9.

Sua janela é vista na Figura 6.5.

6.4 NETMON (EMBRAER)

Outro programa criado na Embraer, o NetMon (Figura 6.6) é um aplicativo poderoso que monitora em tempo real o tráfego de rede do computador, indica todos os tipos de pacotes recebidos e fornece informações de números de pacotes recebidos, velocidade de comunicação, perda de pacotes, *resets*, entre outros.

O programa também é capaz de simular pacotes iNET-X para testes em receptores ou processar arquivos PCAP gravados a escolha do usuário.

Dentre as muitas funções disponíveis do NetMon, apenas algumas poucas foram utilizadas, principalmente para testes de transmissão e recebimento de pacotes iNET-X pelos programas desenvolvidos no projeto.

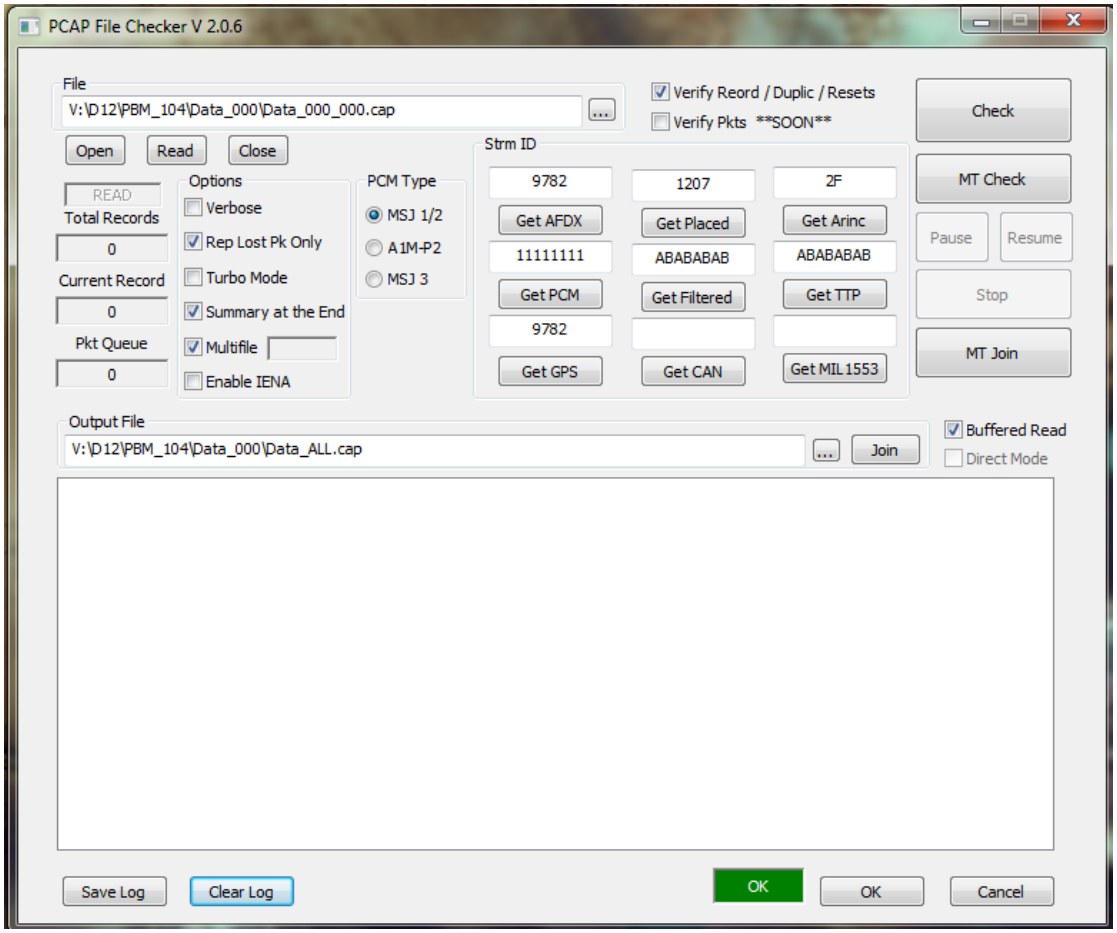

FIGURA 6.5 JANELA DO PROGRAMA PCHECK, PRÓPRIO DA EMBRAER

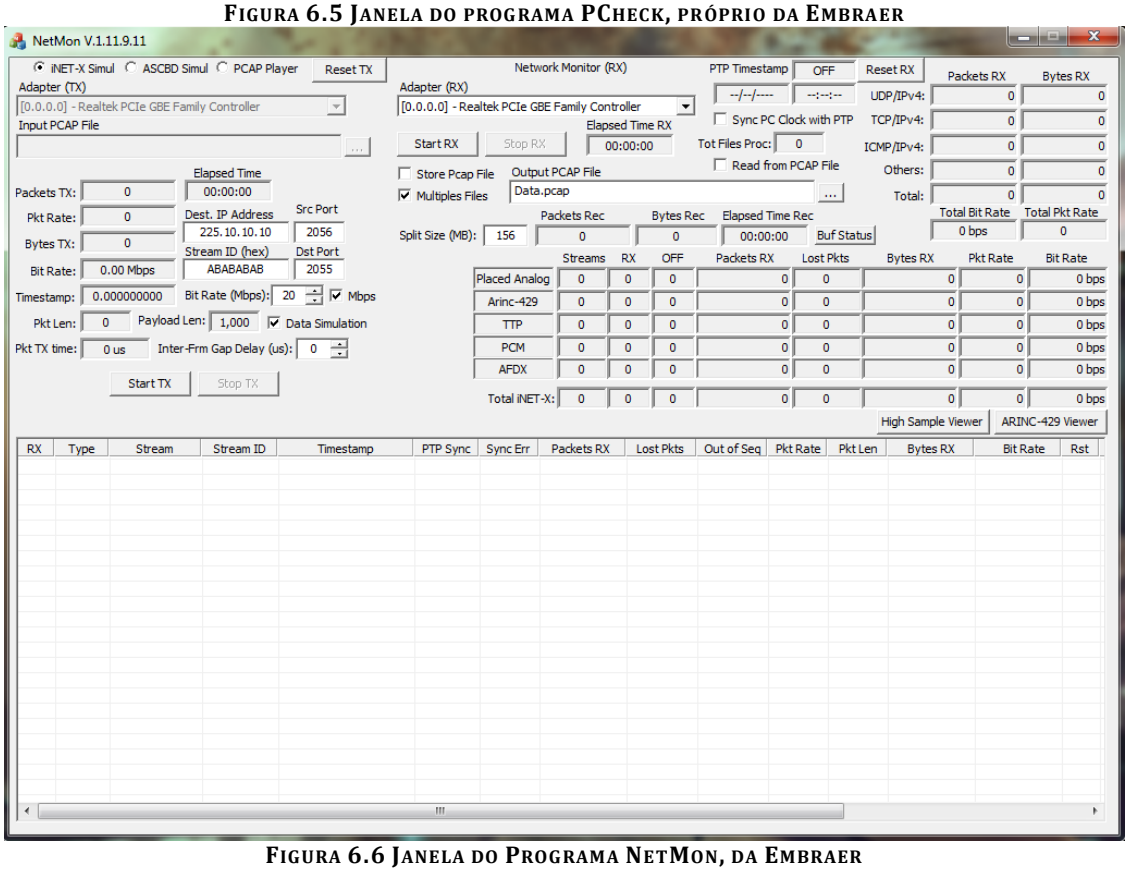

FIGURA 6.6 JANELA DO PROGRAMA NETMON, DA EMBRAER

Capítulo 7. PROGRAMA: GPSCAP

7.1 MOTIVAÇÃO

Atualmente, podem-se encontrar GPS modernos que transmitem suas informações via Ethernet. É impossível, entretanto, trocar todos os sistemas de localização de aeronaves já produzidas, onde as informações são enviadas via uma porta serial do tipo RS-232.

Visando atualizar o método de envio de mensagens de GPS, a ideia desse programa nasceu. Seu propósito inicial é receber via RS-232 as mensagens de GPS, verificar erros nessas mensagens e então enviá-las via Ethernet.

Em adição, o programa apresenta três possibilidades de envio:

- Envio dos três tipos de mensagem (BESTVEL, BESTXYZ e OMNIHPPOS) com um único *Stream ID*;
- Envio dos três tipos de mensagem separadamente, com *Stream IDs* diferentes;
- Envio de uma única mensagem de GPS, utilizando o padrão OmniSTAR.

No padrão OmniSTAR, a mensagem é enviada sem utilização de cabeçalho. Nos outros dois, utiliza-se o formato *iNET-X Parser Aligned* (ver o tópico 4.2.1.6).

7.2 DESENVOLVIMENTO DO PROGRAMA

O recebimento e envio de mensagens tem prioridade crítica nesse programa. A fonte das mensagens, o GPS, não possui sistema de confirmação de recebimento (*comunicação assíncrona*), portanto não é possível diminuir o número de mensagens enviadas por segundo, ou requisitar o reenvio de mensagens perdidas.

Para evitar a perda de informação o programa deve receber, tratar e enviar as mensagens de forma rápida. Para isso, o programa foi construído em quatro partes, chamadas *threads*.

Thread é uma sequência de instruções do programa que pode ser conduzida de forma independente do programa. Um programa pode conter múltiplas *threads*. Caso o computador tenha um único processador, este executará as *threads* semelhante a um comutador, por multiplexação de divisão de tempo (TDM). Essa troca entre tarefas geralmente acontece rápida o suficiente de modo que as *threads* sejam executadas quase simultaneamente. No caso de computadores com mais de um processador (incluindo sistemas *multi-core*), a execução das tarefas ocorre simultaneamente, onde cada processador executa uma ou mais *threads* separadamente.

7.2.1 RECEBIMENTO DE MENSAGENS VIA PORTA SERIAL

Buscando aumentar ao máximo a velocidade de recebimento das mensagens, optou-se pela criação de uma porta *I/O Completion*.

Segundo o site de referência de Desenvolvedores Windows (MSDN) [11]: Porta *I/O Completion* é um modelo eficiente para se processar múltiplas requisições assíncronas de recebimento em um sistema de multiprocessadores. Quando um processo cria uma porta *I/O Completion*, ele cria também uma lista de objetos relacionados a solicitações, cuja única função é servir a essas solicitações. Isso permite lidar rapidamente com as solicitações, no momento em que elas ocorrem.

A *Thread* responsável pela comunicação serial processa constantemente as mensagens recebidas na porta especificada, com a frequência especificada, e disponibiliza tais mensagens em um *buffer* de memória, que é acessado pela *Thread* seguinte. Quando esse *buffer* está completo, o processo de escrita recomeça.

7.2.2 LEITURA E ORGANIZAÇÃO DAS MENSAGENS RECEBIDAS

A próxima *Thread* é acionada no momento em que a primeira mensagem de GPS é recebida, e continua seu processo durante todo a comunicação.

Essa *Thread* deve agir de modo eficiente, acompanhando a velocidade da anterior. Caso a primeira *Thread*, de recebimento, seja muito mais veloz, o *buffer* será reescrito antes que todas as mensagens sejam lidas e organizadas, acarretando na perda de mensagens.

A principal função desse processo de leitura é separar as mensagens recebidas e organizá-las uma por linha. Como a comunicação serial é contínua, todas as mensagens chegam seguidas umas das outras, sem separação de tempo entre elas.

Sabe-se que todas as mensagens recebidas possuem um comum o caractere “#” que indica o início da mensagem (caractere de sincronismo) e um caractere “\n”, que representa o fim de linha (ver o tópico 5.3). O processo de organização é feito localizando um caractere “#” e copiando para uma linha de um novo *buffer* de memória todos os caracteres seguintes, até se encontrar o “\n”. Então, uma nova linha é indicada e o processo recomeça, procurando pelo próximo “#”.

Semelhante à *Thread* anterior, esse novo *buffer* de memória possui tamanho fixo, e quando todas as linhas são preenchidas, o processo de escrita recomeça.

7.2.3 TRATAMENTO DAS MENSAGENS DE GPS E LOCALIZAÇÃO DE ERROS

A Terceira *Thread* também deve estar em sincronismo perfeito com a anterior. Assim que uma mensagem de GPS é identificada e escrita no *buffer* de leitura, essa *Thread* é acionada e continua seu processo enquanto houver mensagens novas a serem lidas nesse *buffer*.

Conhecendo o formato das mensagens de GPS, sabe-se que os três tipos de mensagens possuem um cabeçalho com formato comum, onde as informações de Tipo de mensagem, Porta de comunicação, Tempo, entre outros (ver o tópico 5.3).

Para a seleção dos parâmetros desejados, utiliza-se uma função criada especialmente para extrair as informações entre vírgulas, chamada *GetFieldFn*. Nessa função, deve-se conhecer a localização da informação desejada, inserindo como parâmetro de entrada o número de vírgulas que precedem a informação. Novamente, deve-se ter em mente o formato da mensagem de GPS recebida.

Depois de identificado o tipo de mensagem, localiza-se a informação de Tempo (em segundos), presente no cabeçalho de todas as mensagens. Esse valor é armazenado e posteriormente compara-se o valor do tempo de uma mensagem com o valor obtido na mensagem anterior.

O GPS utilizado envia um trio de mensagens a cada 0,1 segundo. Ou seja, a cada segundo, o GPS é atualizado dez vezes. O programa, então, deve comparar duas informações de tempo consecutivas (do mesmo tipo de mensagem), *time1* e *time2*. Da comparação entre os dois valores, os seguintes resultados são possíveis:

- (*time2* - *time1*) ≥ 0,11: É esperado que o tempo da segunda mensagem fosse maior que o primeiro, porém apenas 0,1 segundo. Caso a diferença seja maior, é considerada uma situação de Mensagens Perdidas.
- *time1* > *time2*: Uma mensagem de tempo menor que a anterior também é um situação de Mensagem Perdidas.

O programa tem função apenas de indicar um salto de tempo nas mensagens e não indica como esse salto ocorreu. Portanto, para as duas situações de Mensagens Perdidas, calcula-se o número de mensagens perdidas através da seguinte equação (Trecho de Código 7.1):

```
Missing1 = (int)(floor (Missing1 + 10*abs(time2 - time1)- 0.5f));
```

TRECHO DE CÓDIGO 7.1 EXEMPLO DA ESTRUTURA DE CÁLCULO DO NÚMERO DE MENSAGENS PERDIDAS

Note a necessidade da utilização da função '*floor*', pois o número 'Missing1' é a parte inteira do resultado da diferença entre dois números do tipo *float* (ponto flutuante), time1 e time2. '*Floor*' retorna o maior numero inteiro menor ou igual ao resultado da expressão numérica. A subtração do valor '0.5' garante que o resultado seja arredondado para baixo sempre que a parte decimal for menor que 0,5.

O número de mensagens perdidas é mostrado no diálogo para que o usuário possa acompanhar o que está acontecendo enquanto o programa é executado. A verificação de perdas de mensagens é feita separadamente para cada um dos tipos de mensagens.

7.2.4 CRIAÇÃO DA MENSAGEM DE GPS NO PADRÃO OMNISTAR

Se for escolhido o método de envio das mensagens originais, no formato iNET-X, depois de verificada a perda de mensagens, o programa aciona a Quarta *Thread*, responsável pelo envio das mensagens.

Caso seja optado pelo envio das mensagens no padrão OmniSTAR, o processamento da Terceira *Thread* segue para a próxima etapa.

Utilizando a mesma função de captura de informações, *GetFieldFn*, todas as informações desejadas de cada uma das três mensagens, são capturadas e salvas em variáveis criadas especificamente para armazenar tais valores. Para facilitar a organização das informações e melhor identificar de onde elas procederam, utilizam-se três estruturas, uma para cada tipo de mensagem. As estruturas podem ser vistas no Trecho de Código 7.2:

```
//Estruturas do pacote OmniSTAR
#define MSGINFO     32
struct BESTXYZ_Block {
       char   X[MSGINFO];
       char   Y[MSGINFO];
       char   Z[MSGINFO];
       char   Sig_X[MSGINFO];
       char   Sig_Y[MSGINFO];
       char   Sig_Z[MSGINFO];
       char   VX[MSGINFO];
       char   VY[MSGINFO];
       char   VZ[MSGINFO];
       char   Sig_VX[MSGINFO];
       char   Sig_VY[MSGINFO];
       char   Sig_VZ[MSGINFO];
};
struct OMNIHPPOS_Block {
       char   Lat[MSGINFO];
       char   Lon[MSGINFO];
       char   Hgt[MSGINFO];
       char   Sig_Lat[MSGINFO];
       char   Sig_Lon[MSGINFO];
       char   Sig_Hgt[MSGINFO];
};
struct BESTVEL_Block {
       char   HorSpd[MSGINFO];
       char   TrkGnd[MSGINFO];
       char   VertSpd[MSGINFO];
};
```

TRECHO DE CÓDIGO 7.2 ESTRUTURAS DAS INFORMAÇÕES NECESSÁRIAS DE CADA MENSAGEM PARA A CONSTRUÇÃO DA MENSAGEM NO PADRÃO OMNISTAR

Conhecendo a posição da informação desejada dentro na mensagem, a função *GetFieldFn* seleciona tal informação e a copia para a variável desejada.

```
GetFieldFn(GpsMsgBuff[LineMtx3].Gps_Msg, 11, (char*) Omnihppos.Lat);
```

TRECHO DE CÓDIGO 7.3 EXEMPLO DE UTILIZAÇÃO DA FUNÇÃO GETFIELDFN

No Trecho de Código 7.3, a informação presente depois da '11ª' vírgula é lida da mensagem de GPS da linha 'LineMtx3' do *buffer* 'GpsMsgBuff' e armazenada em 'Omnihppos.Lat'. A mensagem obtida é a informação de Latitude.

A função *GetFieldFn* tem outra função importante: caso o dado desejado não seja obtido, o resultado devolvido à variável é uma mensagem de seis caracteres "x": 'xxxxxx'. Essa mensagem, indicando falha na captura do campo, também faz parte do padrão OmniSTAR para informações faltantes.

Quando o bloco de três mensagens (uma de cada tipo), todas correspondentes ao mesmo valor de tempo, é finalizado, ocorre o processo de escrita da nova mensagem de GPS OmniSTAR em um novo *buffer* e aciona a Quarta *Thread*.

No caso de mensagens perdidas, o programa é capaz de detectar um bloco incompleto e preencher as informações não obtidas com a *string* padronizada 'xxxxxx'. Nesse caso, há também um contador de mensagens OmniSTAR perdidas, visível para o usuário.

7.2.5 ENVIO DE MENSAGENS INET-X OU OMNISTAR

A Quarta *Thread*, é dividida em dois caminhos distintos: o envio de mensagens de GPS no padrão OmniSTAR, sem cabeçalho, ou o envio de mensagens originais, com o uso de cabeçalho iNET-X. Essa segunda parte compreende as possibilidades de envio das três mensagens com o mesmo cabeçalho ou das mensagens com cabeçalhos separados.

O envio das mensagens separadas possibilita a escolha de *Stream IDs* diferentes para cada uma delas, além de separar as informações do cabeçalho do iNET-X, como número de sequência, tempo, total de pacotes enviados, entre outros.

O envio das três mensagens com um único *Stream ID* é utilizado quando não é desejada a separação das mensagens. Essa função utiliza o mesmo cabeçalho para todas as mensagens, sem fazer a separação delas.

7.2.6 CARACTERÍSTICAS DA JANELA DO PROGRAMA

Na Figura 7.1, tem-se a janela, ou diálogo, criado com o auxílio da ferramenta Microsoft Visual Studio, permite que o usuário selecione em (A) a porta serial de onde as mensagens de GPS serão recebidas, assim como o *Baud Rate* de comunicação.

Para o envio, o usuário pode inserir em (B) o endereço de IP e o número da Porta de destino, além de escolher entre o envio da mensagem no padrão OmniSTAR ou iNET-X (C), com três ou um *Stream ID* (D).

Para visualização apenas, o programa mostra o Status do Recebimento e do Envio de mensagem (E) e o estado separado da mensagem OmniSTAR (F) e das três

mensagens originais de GPS (G). É também visível o número de mensagens perdidas (H).

Apenas para comparação, o diálogo mostra a informação de tempo contida nas mensagens de GPS e o horário do computador que está executando o programa (I).

Os botões *Start* e *Stop* (J) controlam o recebimento e envio de mensagens, que sempre devem ocorrer simultaneamente.

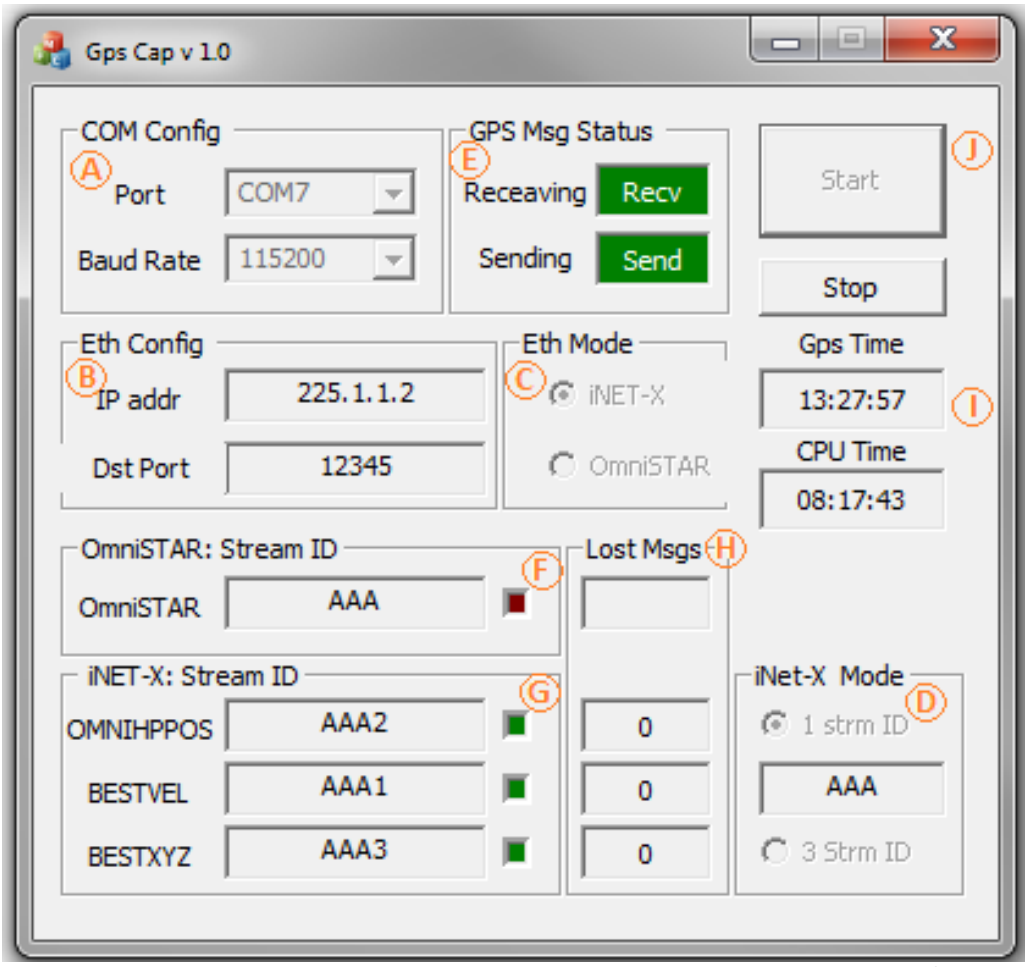

FIGURA 7.1 CAIXA DE DIÁLOGO DO PROGRAMA GPSCAP, INCLUINDO INDICADORES DE FUNÇÕES (EM LETRAS LARANJA)

7.3 RESULTADOS OBTIDOS

O programa foi desenvolvido ao longo do tempo, com cada módulo (*thread*) testada isoladamente à medida que o código se desenvolvia e as *threads* eram adicionadas. O Microsoft Visual Studio permite que o código seja testado passo a passo, oferece possibilidade de visualização do valor das variáveis e dos *buffers* de memória. Além disso, é possível a inserção *Breakpoints* no código, para que a execução seja interrompida quando a instrução selecionada for alcançada. Isso permite a fácil localização de erros no programa, permitindo rastrear, por exemplo, uma variável e detectar se esta é corrompida em algum momento da execução.

Os principais erros encontrados durante a construção do código serão tratados na próxima sessão.

7.3.1 ERRO: LEITURA E ESCRITA SIMULTÂNEA DE UMA MESMA MENSAGEM

O exato sincronismo de execução das diferentes *threads* é algo difícil de conseguir. Um dos problemas que o programa apresentou era quando uma das linhas de mensagens de GPS estava sendo lida pela Segunda *Thread* quando a Primeira sobrescrevia seus dados, ou então o oposto: a Primeira *Thread* escrevia uma linha e a Segunda já começava a ler, antes do processo de escrita terminar. A mensagem obtida era uma mensagem corrompida, acusada erroneamente como perda de mensagem durante a execução. Tal ocorrência poderia acarretar na destruição total da mensagem, ou apenas parcial. A destruição parcial forneceria dados incorretos para o programa.

A solução encontrada para esse problema foi a criação de variáveis do tipo booleana que indicavam o status da linha de mensagem do *buffer* de memória. Seguindo o critério, enquanto a linha estivesse na fase de escrita, a *thread* de leitura não poderia acessá-la. Esse procedimento também impediu que uma linha antiga fosse lida pela segunda vez, pois seu *status* é verificado. A Tabela 7.1 de indicadores pode ser vista a seguir.

TABELA 7.1 LÓGICA DE STATUS DO *BUFFER* DE MEMÓRIA PARA ESCRITA E LEITURA DAS MENSAGENS DE GPS RECEBIDAS

BuffStatus1 (bool)	BuffStatus2 (bool)	Significado
0	0	Linha disponível para escrita (já lida)
0	1	Escrevendo linha (proibida a leitura)
1	0	Linha disponível para Leitura (já escrita)
1	1	Lendo linha (proibida a escrita)

Caso a *Thread* de leitura encontre uma linha no *status* 01, o processo espera até a liberação da linha. Como as instruções de escrita e leitura são muito velozes, a espera não acarreta em adição de tempo relevante ao processo e evita colisões.

7.3.2 ERRO: MENSAGENS CORROMPIDAS

Caso a comunicação serial seja interrompida inesperadamente, ou ocorram falhas durante o recebimento de mensagens, o programa receberá uma mensagem corrompida. Essa mensagem pode apresentar valores incorretos das informações desejadas, ou então perder completamente o formato padrão das mensagens de GPS, impossibilitando, por exemplo, o processo de localização da informação de tempo, pois a mensagem não apresentará as vírgulas separando os dados.

Para evitar a leitura de mensagens corrompidas, criou-se um processo que verifica o tamanho da mensagem recebida (que é calculado como o número de caracteres existentes entre dois "#" consecutivos). Caso esse tamanho não esteja de acordo com o esperado (pois se sabe qual o tamanho padrão das mensagens de GPS tratadas

pelo programa), a mensagem é considerada corrompida e é descartada. O programa passa, então, a ler a próxima mensagem do *buffer* e verificá-la.

7.3.3 RESULTADOS: TESTE DO ENVIO E RECEBIMENTO DE MENSAGENS

Depois de considerado pronto, foi realizado o teste final do programa. Para isso, utilizou-se um simulador de GPS para enviar um arquivo de mensagens conhecidas (gravadas do GPS original) via comunicação serial para o GpsCap e em seguida, outro programa, destinado a receber e gravar as mensagens enviadas via Ethernet. O programa utilizado para monitorar toda a comunicação de rede (que recebeu as mensagens iNET-X) é de autoria da Embraer, conhecido como NetMon. Para monitorar as mensagens OmniSTAR, utilizou-se o programa Wireshark.

Depois de transmitido todo o arquivo de mensagens, verificou-se a relação entre o número de mensagens enviadas pelo simulador, o número de mensagens recebidas pelo GpsCap, e a relação entre o numero de mensagens enviadas pelo GpsCap para o programa de destino e a quantidade de mensagens de fato recebidas. Esse processo foi repetido para os três tipos de envio do GpsCap.

Como esperado, os números foram iguais, sem mensagens perdidas em nenhuma das formas de comunicação. O resultado pode ser visto na Tabela 7.2:

TABELA 7.2 RESULTADO DO TESTE DE ENVIO E RECEBIMENTO DE MENSAGENS

Modo de Transmissão do GpsCap	Msg Enviadas (simulador de GPS)	Msg Recebidas (GpsCap)	Msg Recebidas (NetMon ou Wireshark)	Msgs Perdidas no Recebimento	Msgs Perdidas no Envio
OmniSTAR	17438	17438	17438	0	0
iNET-X (*1 Strm ID*)	11570	11570	11570	0	0
iNET-X (*3 Strm ID*)	12486	4162	4162	0	0
		4162	4162	0	0
		4162	4162	0	0

Note que, no envio de mensagens com formato iNET-X e 3 *Stream ID*, foram enviadas o total de 3 x 4162, que resulta 12486 mensagens. Isso indica que todos os conjuntos com os três tipos de mensagens foram corretamente recebidos e enviados.

Para garantir também que as mensagens do tipo iNET-X foram enviadas sem alteração de conteúdo, utilizou-se um *software* capaz de comparar os dois arquivos de mensagens de GPS: o arquivo original, enviado pelo simulador e o arquivo final enviado pelo GpsCap. Novamente como esperado, o conteúdo das mensagens era idêntico em ambos os arquivos.

7.4 CONSIDERAÇÕES FINAIS E CONCLUSÕES

O programa GpsCap atende aos requisitos estipulados, recebendo e enviando corretamente as mensagens de GPS, sem alterar seu conteúdo quando utilizado o formato iNET-X.

A vantagem de permitir o envio das mensagens separadas em *Stream IDs* diferentes possibilita que tais mensagens sejam gravadas em arquivos diferentes, de acordo com a necessidade do usuário. Essa vantagem também pode ser estendida ao método de processamento das mensagens dentro do programa, que é feito de forma independente de acordo com o tipo de mensagem. Desse modo, o programa pode indicar ao usuário que o recebimento e o envio estão ocorrendo e indica também quais os tipos de mensagens recebidas (pode-se ter o recebimento de um pacote apenas com mensagens do tipo BESTVEL, por exemplo, sem prejudicar o restante do processamento).

Capítulo 8. PROGRAMA: PKTCONTROL

8.1 MOTIVAÇÃO

Para complementar a comunicação de telemetria, é preciso um programa capaz de receber todos os pacotes de mensagens vindos da *Ground Station* (recebimento em modo promíscuo) e selecionar apenas alguns deles a serem enviados para a aeronave (modo de envio seletivo). No sentido oposto de comunicação, o mesmo deve ocorrer. O programa deve ser capaz de receber todas as mensagens enviadas pela aeronave e selecionar dessa vez todos os pacotes para serem enviados para a *Ground Station*.

Como a comunicação será realizada por transceptores (t*ransceivers*), o programa deve estar preparado para receber os seguintes tipos de pacotes:

- Pacote fora de ordem: Como o transceptor detecta a perda de pacotes e pede para o sistema o reenvio do pacote perdido, o programa deverá estar preparado para receber pacotes com o número de sequência fora de ordem, além de verificar se isso é de fato uma mensagem reordenada ou não.
- Pacote duplicado: É possível que o pacote seja enviado uma segunda vez, antes que o receptor tenha chance de responder que o primeiro pacote foi recebido corretamente. O sistema de transmissão dá ao receptor uma chance de responder se o pacote foi recebido dentro de um determinado tempo. Caso essa resposta não chegue no tempo programado, é enviada uma cópia do pacote anterior. O programa deve reconhecer pacotes duplicados e descartá-los.
- Pacote perdido: Quando o programa detectar um salto entre números de sequência de dois pacotes, não deve ser acusado de imediato um caso de pacote perdido. O programa deve verificar se o pacote em falta é recebido de modo reordenado ou não. Caso o pacote em falta não seja recebido posteriormente, esse sim será um pacote perdido. A situação de mensagens perdidas ocorre quando o transmissor desiste de enviar o pacote depois de oito tentativas sem sucesso.
- *Reset* de barramento: caso a comunicação seja cortada ou ocorra um *reset* nos números de sequência dos pacotes por algum motivo, o programa deve estar preparado para lidar com a situação, sem acusar erroneamente a perda de pacotes, nem a chegada de pacotes reordenados.
- Pacotes corrompidos: Toda comunicação é passiva de falhas. O programa deve ser apto a reconhecer pacotes mal formados e descartá-los.

Como o programa a ser utilizado no sistema embarcado tem função muito semelhante ao programa da *Ground Station,* optou-se pela criação de um único programa, capaz de realizar as duas funções.

8.2 DESENVOLVIMENTO DO PROGRAMA

Semelhante ao GpsCap, o PktControl exige que seja gasto o mínimo de tempo possível em suas funções, de modo a evitar a perda de pacotes recebidos e enviados. Por isso, optou-se pela utilização de *threads* também.

O recebimento de pacotes será em modo promíscuo, ou seja, o programa deverá receber todos os pacotes da rede e ser capaz de selecionar aqueles que lhe interessam (que fazem parte da comunicação *Ground Station*-Aeronave).

8.2.1 RECEBIMENTO DE PACOTES

O PktControl identifica os adaptadores de rede presentes no computador em que está sendo executado e disponibiliza para que o usuário selecione um desses adaptadores para o recebimento dos dados.

Em seguida, utilizando funções da biblioteca WinPcap [12], a Primeira *Thread* tem função de receber todos os pacotes da rede e armazená-los em um *buffer* de memória. Esse *buffer* é formado por varias linhas de uma estrutura construída especificamente para essa aplicação, onde se coloca em variáveis definidas de informações de cabeçalho (*Header*), dados do pacote (*Data*) e o tamanho do dado (*DataLenght*). O cabeçalho tem o formato definido pela biblioteca WinPcap.

Para controlar a leitura e escrita do *buffer* de memória, utilizou-se uma variável do tipo "*long*" (inteiro de 32bits) que contabiliza o número de pacotes escritos. Quando o pacote armazenado é lido pela próxima *thread*, essa variável é decrementada. A fim de evitar que o parâmetro seja mudado simultaneamente pelas duas *threads*, o que poderia causar o corrupção deste, utilizou-se as funções "*InterlockedIncrement*" e "*InterlockedDecrement*", que incrementam ou decrementam – respectivamente – a variável especificada em um. Funções do tipo *Interlocked* sincronizam o acesso à variável designada, que pode ser compartilhada entre múltiplas *threads*. Desse modo, enquanto uma *thread* estiver executando uma operação com essa variável, esta permanece indisponível para outras *threads*, que devem esperar pela disponibilização da variável.

Todo pacote recebido, antes de ser armazenado, tem seu tamanho real comparado à informação de tamanho contida em seu PCAP *header*. Esse procedimento permite a detecção e descarte de pacotes mal formados.

Depois do recebimento do pacote de rede válido e armazenamento no *buffer* de memória, é acionada a Segunda *Thread*, responsável pela leitura do pacote.

8.2.2 TRATAMENTO DE PACOTES DO TIPO INET-X

Nesta primeira fase do projeto, o programa PktControl está apto a receber e retransmitir apenas pacotes no padrão iNET-X. No futuro, o programa pode evoluir de modo a suportar diferentes formatos.

Os pacotes recebidos armazenados no *buffer* de memória são agora verificados interiormente, de modo a selecionar apenas os pacotes iNET-X.

O processo de seleção do pacote é feito em estágios, abrindo as camadas mais externas até se chegar às informações mais internas. Foi visto anteriormente no tópico 4.2 que o pacote iNET-X possui vários encapsulamentos, na seguinte ordem:

- PCAP *header*;
- MAC *header*;
- IP *header*;
- UDP *header*;
- iNET-X *header* e *payload*;

Como cabeçalho PCAP já foi separado na *thread* anterior, essa *thread* lida com o restante do pacote, ou seja, com os demais cabeçalhos e o *payload* do iNET-X.

Primeiro, o programa confere se o pacote tem, no mínimo, o tamanho do MAC *header*. Aqueles menores que esse valor, são descartados, pois não há possibilidade de se tratar de um pacote iNET-X.

Novamente, utilizando estruturas e ponteiros criados para capturar as informações dos cabeçalhos, é verificado se o *Ether Type* (informação contida no MAC *header*) do pacote possui valor 0x800. Caso positivo, este é um pacote *Ethernet* do tipo IP (*Internet Protocol*).

Confirmado esse valor, o processamento segue, agora extraindo as informações dos demais *headers*. A próxima verificação é a versão do IP, que deve ter valor 4. Esse valor está localizado no IP *header*. Em caso positivo, mais um passo é avançado no processamento, com a verificação do Protocolo, também contida no IP *header.* Este valor deve ser 17, indicando um pacote IPv4, com protocolo UDP.

Por fim, verificam-se as informações contidas do iNET-X *header*, para identificar se este se trata do pacote desejado. O cabeçalho dever ter *Version* 1, *OptionCount* 1, *Reserved* 0 e MessageFlags 0.

A confirmação desses quatro valores, entretanto, não garante prontamente que este seja um pacote iNET-X. É possível que pacotes corrompidos mantenham seus cabeçalhos, mas carreguem nada além de lixo em seu *payload*. Para evitar a retransmissão de pacotes corrompidos, são feitas duas verificações de extrema importância:

- O tamanho do pacote iNET-X deve ser igual ao tamanho declarado no cabeçalho UDP, menos o tamanho desde cabeçalho;
- O tamanho declarado no UDP *header* dever ser igual ao tamanho do pacote declarado do cabeçalho do PCAP, menos os tamanhos de todos os cabeçalhos MAC e IP.

O Trecho de Código 8.1 mostra a lógica utilizada:

`IpHdrLen = pIpHdr->ip_header_len * 4; /*multiples of 4 bytes */`
`XnetPktLen = ntohl(pXnetHdr->xnet_PacketLength);`
`UdpLen = ntohs(pUdpHdr->udp_length);`
	(a)

`if( XnetPktLen != ( UdpLen - sizeof(UDP_HDR) ) ){`
`  iNetXData[indx].bMalFormed = TRUE;`
`}`
`if( UdpLen != ((RxPktBuffer[ParserBufferIndex].PktDataLength) –`
`  sizeof(MAC_HDR) - IpHdrLen ) ){`
`  iNetXData[indx].bMalFormed = TRUE;`
`}`
	(b)

TRECHO DE CÓDIGO 8.1 LÓGICA UTILIZADA NA DETECÇÃO DE PACOTES MAL FORMADOS

Nesse trecho, tem-se em (a), a definição das constantes utilizadas. Em (b), as instruções de decisão. O índice "indx" controla o número de linha do *buffer* de mensagens iNET-X, quanto o "ParserBufferIndex" indica a linha do *buffer* de todas as mensagens recebidas

No procedimento seguinte, só serão utilizados pacotes iNET-X que não tenham a indicação de pacote mal formado. Os demais são descartados.

A utilização das funções "ntohl" e "ntohs" são necessárias, pois estamos lidando com informações vindas de uma rede TCP/IP. A função "ntohl" converte um número de 32 bits com ordem de Bytes da rede TCP/IP em um número de 32 bits com ordem de bytes invertida, para o computador *host*. A função "ntohs" faz o mesmo procedimento, porém com números de 16 bits.

8.2.3 VERIFICAÇÃO DOS PACOTES iNET-X

Um pacote válido iNET-X passa por uma complexa lógica de instruções a fim de se verificar a ocorrência de *Resets*, pacotes Duplicados e pacotes Reordenados.

Para facilitar o procedimento, utiliza-se um *buffer* de memória com estruturas criadas para conter todas as informações necessárias para o processo. Esse *buffer* é regido por um número de índice, o "indx", que por sua vez é determinado pelo *Stream ID* do pacote atual. Diferentemente dos demais *buffers*, portanto, o *buffer* iNetXData não tem seu índice de linha incrementado em um a cada passo do processo, mas ele caminha de forma não linear, podendo aumentar, diminuir ou se manter o mesmo, seguindo somente a ordem de sequência determinada elo *Stream ID* do pacote iNET-X conforma estes são recebidos. Desse modo, as informações referentes a um tipo de *Stream ID* ficam sempre armazenadas em uma mesma linha. Quando um novo pacote é lido, informações como o último número de sequência serão obtidas referentes a esse mesmo *Stream ID*, evitando o risco de se misturar informações de tipos de dados diferentes.

Escolhe-se a informação de *Stream ID* do pacote para se distinguir um pacote iNET-X do outro, pois este número é bem definido em telemetria. Cada tipo de informação possui uma sequência de *Stream IDs* específica. Através desse número, pode-se

conhecer a fonte dos dados, por exemplo. Caso o programa detecte um grande número de perda de pacotes em um dado tipo de pacote, todos com o mesmo *Stream ID*, é possível identificar a quem pertence esse dado e verificar o que há de errado com a fonte ou o barramento dessas informações. Tais números definem pacotes contendo dados de vídeo, PCM, informações de comandos do piloto à aeronave, dados de GPS, entre outros.

O Trecho de Código 8.2 a seguir mostra a estrutura de informações referentes aos pacotes iNET-X.

```
typedef struct iNetX
{
	bool	bFirst;		//indica que é o primeiro pkt recebido

	BOOL	bToBeSent;	//indica que o pacote deve ser enviado (TxThread)
	BOOL	bReceiv;	//indica que o pacote foi recebido

	BOOL	bSeq_Descont; //indicador de evento: Descontinuidade
	BOOL	bSeq_Dupl;	//indicador de evento: Pacote Duplicado
	BOOL	bSeq_Reord;	//indicador de evento: Pacote Reordenado
	BOOL	bSeq_Reset;	//indicador de evento: Reset
	BOOL	bMalFormed;	//indicador de evento: Pacote Mal Formado

	sockaddr_in Src_IpAddress; //IP Addr da fonte do ptk
	sockaddr_in Dst_IpAddress; //IP Addr de destino do pkt
	USHORT Src_UdpPort;  //Porta da fonte
	USHORT Dst_UdpPort;  //Porta de destino
	UINT	Packet_Len;	//tamanho do pkt
	UINT	SecCurrent;	//tempo do pkt em seg, no header
	UINT	NanoSecCurrent;//tempo do pkt em nseg, no header
	UINT	PTPSync;	//sincronismo
	UINT64 PTPSyncError; //erro de sincronismo
	WORD	StreamType;	//Tipo do Stream ID encontrado - prédeterminado!
	DWORD	StreamID;	//Stream ID do pkt

	int	Packet_Rate;  //iNetXData[x].Packet_Rate = (int)
(iNetXData[x].Total_Pkts - iNetXData[x].Last_Packets_Cnt); **PKT RATE**

	DWORD	Start_Seq_Nbr;//primeiro numero de sequencia do pkt recebido
	DWORD	Last_Seq_Nbr; //ultimo numero de sequencia recebido
	DWORD	Act_Seq_Nbr;  //numero de sequencia do pkt atual
	DWORD	NextExp_Seq_Nbr;//proximo numero de sequencia esperado

	DWORD	Run_Cnt;	//contador grupos de pacotes em sequencia
	DWORD	Reord_Cnt;	//numero de pkts reordenados (fora de ordem)
	DWORD	Ord_Cnt;	//numero de pacotes em ordem
	DWORD	Dupl_Cnt;	//numero de pacotes duplicados
	DWORD	Reset_Cnt;	//contador de resets
	DWORD	Seq_Desc_size;//tamanho da descontinuidade (salto de pacotes)

	DWORD	Total_Pkts;	//número total de pacotes (conta tb pacotes
duplicados)
	DWORD	Last_Total_Pkts;//ultimo numero de pacotes contados
	DWORD	Total_Err;	//total de erros nesse stream id. conta eventos de
descontinuidades + reordenados
	DWORD	Total_Lost;	//total de erros = numero de pkts fora de sequencia,
menos o numero de pacotes reordenados >> Total_Lost = Seq_Desc_size - Reord_Cnt
	DWORD	Total_Rec_Pkt;//total de pacotes recebidos  = pacotes reordenados +
pacotes em ordem (sem contar pacotes duplicados)
```

```
        DWORD  TotMalForm;   //total de pacotes mal formados

        float  Reord_R;      //porcentagem de pacotes reordenados >> Reord_R =
100 * Reord_Cnt / Total_Rec_Pkt
        float  Ord_R;        //porcentagem de pacotes em ordem >> Ord_R = 100 *
Ord_Cnt / Total_Rec_Pkt
        float  Dupl_R;       //porcentagem de pacotes duplicados (do total de
pacotes) >> Dupl_R = 100 * Dupl_Cnt / Total_Pkts
        float  Lost_R;       //porcentagem de pacotes em perdidos >>  Lost_R = 100
* Total_Lost  / Total_Rec_Pkt

        UINT   line;         //linha (para usar no LastPkts_Buff)
        DWORD  LastPkts_Buff [LASTPKTS_BUFF_SIZE];      //buffer para guardar os
ultimos pacotes recebidos

        char   Info[MAXMSGEXCEPTION]; //reservado para mensagens de erro (tabela)

} INETX;
```

TRECHO DE CÓDIGO 8.2 ESTRUTURA DE INFORMAÇÕES DOS PACOTES iNET-X

Para a verificação de *resets*, pacotes duplicados ou reordenados, observa-se o número de sequência do pacote. Essa informação está presente no cabeçalho iNET-X. O número de sequência de um pacote é criado e controlado pelo programa responsável pelo envio desses pacotes. Seu número inicial é zero e é incrementado em um a cada novo pacote, para o mesmo *Stream ID*.

8.2.3.1 *Reset*

O evento *Reset* ocorre quando o processo está à espera de um pacote com certo número de sequência, mas recebe um pacote de número zero. Isso indica que a transmissão de pacotes foi interrompida e reiniciada por algum motivo. Isso não indica necessariamente um erro de envio, pois o procedimento de reinicialização do envio pode ter sido proposital, mas o acontecimento deve ser indicado. Em outros casos, alguns tipos de pacotes possuem número de sequência cíclico, reinicializado toda vez que este número chega ao seu valor máximo. Esse tipo de *Reset* é considerado normal e não é gerado um aviso para o usuário, embora o evento continue sendo contabilizado.

8.2.3.2 Pacote duplicado

A verificação de pacotes duplicados é feita ao se comparar o número de sequência do pacote atual, com o número de sequência dos últimos "LASTPKTS_BUFF_SIZE" pacotes recebidos desse mesmo *Stream ID*. O valor da constante "LASTPKTS_BUFF_SIZE" determina o número de linhas do *buffer* que armazena todos os números de sequência recebidos do pacote. Este é um *buffer* circular: quando todas as linhas foram preenchidas, o processo de escrita recomeça a partir da primeira linha, mas sem limpar todo o *buffer*. No processo de escrita, apenas a linha a ser alterada é apagada e reescrita, enquanto as demais linhas permanecem como estão. O valor "LASTPKTS_BUFF_SIZE" é determinado dentro do programa, através do comando "*#define*". Esse valor pode ser alterado pelo programador de modo a tornar o programa mais ou menos sensível à detecção de pacotes

duplicados. Quando menor for o valor, menos sensível será o programa, pois ele irá comparar o valor do número de sequência atual com um menor número de valores anteriores.

O número de pacotes recebidos em duplicata é contabilizado para o cálculo do percentual de pacotes duplicados, utilizando o Trecho de Código 8.3:

```
iNetXData[indx].Dupl_R = 100 * ((float)iNetXData[indx].Dupl_Cnt /
(float)iNetXData[indx].Total_Pkts);
```

TRECHO DE CÓDIGO 8.3 CÁLCULO DO PERCENTUAL DE PACOTES DUPLICADOS, EQUIVALENTE AO NÚMERO DE PACOTES RECEBIDOS EM DUPLICATA EM RELAÇÃO AO NÚMERO TOTAL DE PACOTES

O alto percentual de pacotes duplicados é uma informação preocupante, pois indicam falhas frequentes de comunicação, erros na detecção de pacotes recebidos, além do desperdício de banda para transmissão de informações já enviadas anteriormente.

8.2.3.3 Pacote em ordem

Depois de tratar um dado pacote, o programa calcula o número de sequência do próximo pacote esperado, que, em situações normais, é igual ao número de sequência do pacote atual adicionado em um. Quando o próximo pacote é recebido, verifica-se se seu número de sequência é igual ao valor esperado.

Em caso positivo, o pacote recebido está em ordem correta e o processo segue em frente, passando a esperar pelo número de sequência seguinte. O número de pacotes recebidos em ordem é contabilizado.

Para verificar a qualidade da comunicação, calcula-se o percentual de pacotes em ordem, através do Trecho de Código 8.4:

```
iNetXData[indx].Ord_R = 100 * ( (float)iNetXData[indx].Ord_Cnt /
(float)iNetXData[indx].Total_Rec_Pkt );
```

TRECHO DE CÓDIGO 8.4 CÁLCULO DO PERCENTUAL DE PACOTES EM ORDEM (ORDER RATIO), EQUIVALENTE AO NÚMERO DE PACOTES EM ORDEM (ORDER COUNTER) EM RELAÇÃO AO TOTAL DE PACOTES RECEBIDOS

Para esse cálculo, o total de pacotes recebidos não leva em consideração os pacotes duplicados.

Obviamente, quanto maior o percentual de pacotes em ordem, melhor é a qualidade da comunicação, pois menos pacotes foram perdidos.

8.2.3.4 Pacote perdido (Descontinuidade)

Caso o número de sequência do pacote atual seja maior que o número esperado, os pacotes faltantes são considerados perdidos. O número de pacotes perdidos é calculado pela diferença entre os números de sequência atual e esperado.

Por exemplo, suponha que o processo está à espera do pacote de número 15, quando recebe o pacote de número 18. Os pacotes 15, 16 e 17 são, então, considerados perdidos, totalizando três pacotes perdidos. Note que o que diferencia a situação dos pacotes 15, 16 e 17 entre pacote perdido ou não, será o recebimento destes posteriormente, de forma reordenada.

Para evitar confusões, o número de pacotes faltantes, calculado pela diferença entre os números de sequência atual e esperado, não são chamados de pacotes perdidos, mas sim de "tamanho da descontinuidade de sequência".

8.2.3.5 Pacote reordenado

Caso o número de sequência do pacote atual seja menor que o número esperado, o pacote é considerado reordenado.

O número de pacotes reordenados é subtraído do número do tamanho total de descontinuidades, para o cálculo do valor real de pacotes perdidos.

O percentual de pacotes reordenado também é calculado, pois esse valor também indica a qualidade da transmissão e recebimento de dados. Veja o Trecho de Código 8.5:

```
iNetXData[indx].Reord_R =  100 * ((float)iNetXData[indx].Reord_Cnt /
(float)iNetXData[indx].Total_Rec_Pkt);
```

TRECHO DE CÓDIGO 8.5 CÁLCULO DO PERCENTUAL DE PACOTES FORA DE ORDEM, EQUIVALENTE AO NÚMERO DE PACOTES EM REORDENADOS EM RELAÇÃO AO TOTAL DE PACOTES RECEBIDOS

Para o cálculo acima, o total de pacotes recebidos não leva em consideração os pacotes duplicados.

O alto percentual de pacotes reordenados indica a perda constante de pacotes pelo transceptor e a tentativa de reavê-los. Este valor, entretanto, deve ser considerado sempre em conjunto com o valor de pacotes de fato perdidos. Se o número de pacotes perdidos for nulo, ou ao menos muito próximo disso, em relação ao total de pacotes recebidos, temos uma situação em que houve perda de pacotes durante a comunicação, mas todos, ou quase todos, foram devidamente recuperados. Ainda assim, o valor ideal de pacotes reordenados é pequeno, mas não nulo, já que é impossível que a comunicação entre a *Ground Station* e a aeronave seja perfeita.

8.2.4 RETRANSMISSÃO DOS PACOTES SELECIONADOS

Mesmo com a existência de pacotes reordenados, duplicados e *resets* de barramento, todos os pacotes devem ser retransmitidos da mesma maneira que foram recebidos. Não cabe ao programa realizar a reordenação de pacotes, pois isso tomaria muito tempo de processamento. Os únicos pacotes que são descartados são os pacotes mal formados. Dessa forma, o programa tem a função de uma espécie de filtro, evitando que pacotes contendo informações corrompidas sejam passados em frente, tanto na comunicação Aeronave-*Ground Station* quanto na comunicação de direção oposta.

O próximo passo do programa é a seleção e preparação dos pacotes para a retransmissão.

Como foi dito inicialmente, o programa deve ser capaz de selecionar apenas alguns pacotes desejados para retransmissão ou então ser capaz de retransmitir todos os pacotes disponíveis.

Para a seleção de pacotes, o programa verifica se o *Stream ID* do pacote está em uma lista de *Stream IDs* a serem retransmitidos. Essa lista é fixa e não pode ser alterada durante a execução do programa. Apenas o programador, que tem acesso ao código do programa, pode alterar a relação de *Stream IDs*.

Se o pacote atual faz parte dos pacotes a serem enviados, suas informações são copiadas para outro *buffer* e permanece lá até ser enviado. A cada informação colocada no *buffer* de transmissão é acionada a Terceira *Thread*, responsável pelo envio dos pacotes.

O *buffer* de transmissão também é composto por linhas de uma estrutura que contém o pacote iNET-X (incluindo informações do cabeçalho e *payload*), além de variáveis do tipo "*bool*" para auxiliar no processamento, indicado o envio do pacote.

O cabeçalho e o corpo do pacote iNET-X é retransmitido sem sofrer alterações. Mesmo o número de sequência e as informações de tempo são mantidos como as originais. Isso é importante, pois o programa não deve influenciar de modo algum nas informações dos pacotes, mantendo a correspondência com a versão original.

Utilizando a biblioteca BlockSocket, a mesma utilizada no programa GpsCap, o pacote é enviado na rede, através do adaptador de rede, para a porta de destino e endereço IP de destino selecionados pelo usuário. Caso o envio seja realizado com êxito, o programa incrementa o número de pacotes enviados e passa para o próximo pacote.

A leitura do *buffer* de transmissão, semelhante ao procedimento da leitura do *buffer* de pacotes recebidos, é controlada por uma variável chamada "TXBufferQueue". Que é alterada somente pela função "*InterlockedDecrement*" ou "*InterlockedIncrement*". Enquanto houver pacotes a serem transmitidos (TXBufferQueue não nulo), a *thread* de transmissão continua seu processo. Quando não há pacotes a serem transmitidos, a *thread* fica a espera da chegada de um novo pacote.

O número total de pacotes enviados é calculado e indicado na tela do programa para o usuário.

Caso o usuário selecione a opção de enviar todos os pacotes, o processo de envio é o mesmo, embora todos os pacotes sejam selecionados para a retransmissão ao invés de apenas alguns. Nessa situação, quando essa opção é selecionada, o

processamento não compara o *Stream ID* do pacote com aqueles da lista, simplesmente entendendo que todos serão enviados.

8.2.5 CARACTERÍSTICAS DA JANELA DO PROGRAMA

Na Figura 8.1, temos a janela do programa PktControl, onde suas características estarão explicadas a seguir.

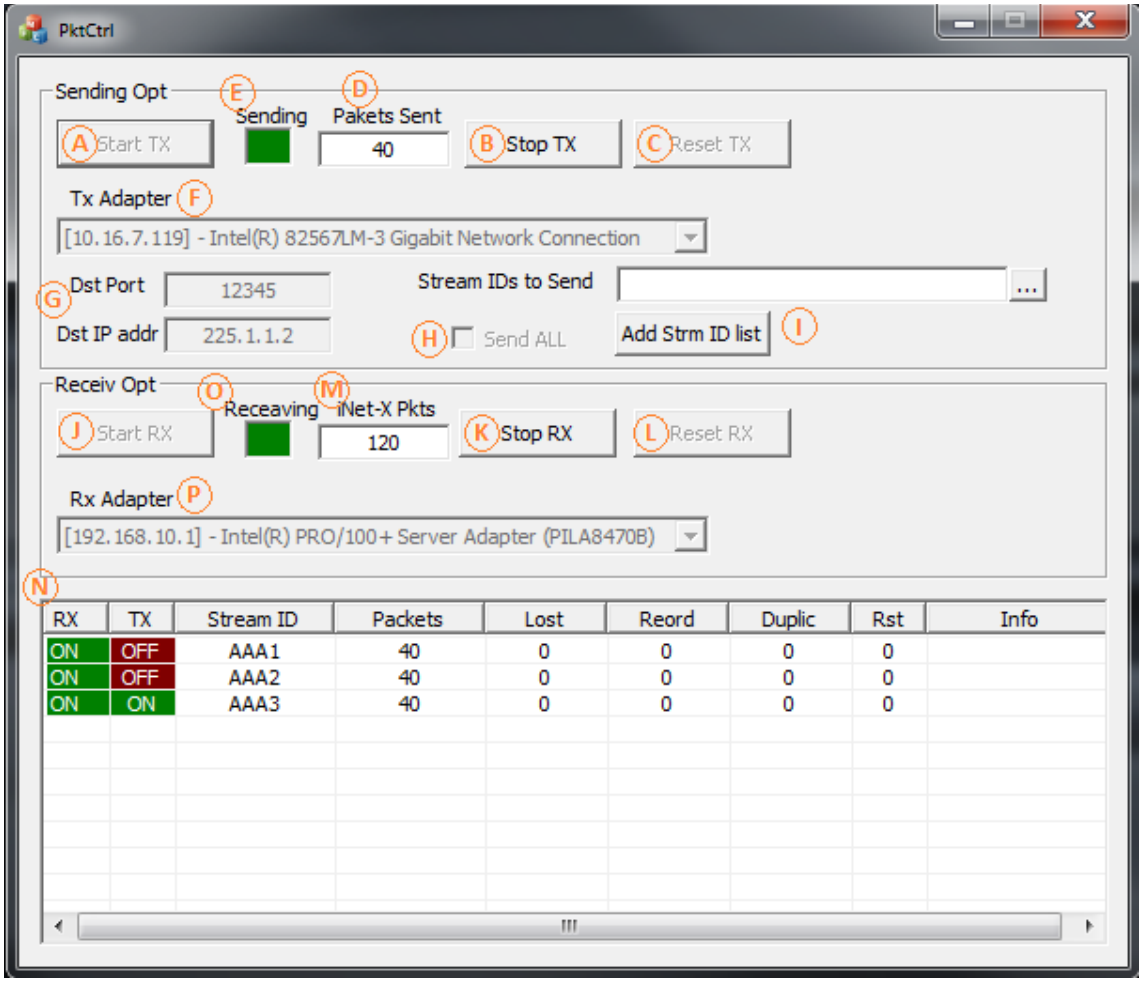

FIGURA 8.1 JANELA DO PROGRAMA PKTCONTROL, COM SUAS FUNÇÕES INDICADAS EM LETRAS LARANJA

8.2.5.1 Transmissão

A janela do programa possui opções diferentes para o envio e o recebimento de pacotes.

Na parte superior da janela, encontram-se os botões "*Start* TX" (A) e "*Stop* TX" (B), que inicializam e pausam a transmissão. O botão "*Reset* TX" (C) é responsável por apagar todas as informações de envio até então calculadas, como o número de pacotes enviados (D), por exemplo. Próximo ao botão "*Start* TX" há um quadro que indica que a transmissão está sendo realizada (quando fica verde) ou não (vermelho) (E).

O Adaptador de Rede para transmissão é selecionado em (F) e as configurações de Porta e Endereço IP são inseridas pelo usuário em (G). Caso essas caixas sejam deixadas em branco, o programa utiliza valores predefinidos armazenados em seu código.

Ainda nas opções de transmissão, o usuário pode selecionar a opção para enviar todos os pacotes recebidos (H). Se a opção for deixada em branco, o programa selecionará os pacotes em função da lista de *Stream IDs* em sua memória.

Ainda em processo de criação, há a possibilidade de fornecer ao programa uma lista própria de *Stream IDs (I)*. O programa deve ler e adicionar as informações do arquivo selecionado (com extensão ".txt") à sua lista interna de *Stream IDs*. Essa função ainda não está totalmente pronta e a alteração da lista ainda poderá ser feita de modo manual pelo programador.

8.2.5.2 Recebimento

As opções relacionadas ao recebimento de pacotes são semelhantes às de transmissão.

Os botões "Start RX" (J) e "Stop RX" (K), que inicializam e pausam o recebimento. O botão "Reset RX" (L) é responsável por apagar todas as informações de até então calculadas, como o número de pacotes iNET-X recebidos (M) e também apaga a lista de informações (N), na parte inferior. Próximo ao botão "Start RX" há também um quadro para indicar se existem pacotes sendo recebidos (quando fica verde) ou não (vermelho) (O).

O Adaptador de Rede de recebimento é selecionado pelo usuário em (P).

Por fim, o programa exibe uma lista de informações (N) de todos os pacotes recebidos, e indica aqueles a serem transmitidos. Além disso, a lista mostra as seguintes informações (Tabela 8.1):

TABELA 8.1 : INFORMAÇÕES MOSTRADAS NA LISTA (N) DO PROGRAMA

Coluna:	Função:
RX	Pacotes recebidos e o status de recebimento (on – verde, off – vermelho)
TX	Pacotes a serem enviados (verde) e aqueles que não serão enviados (vermelho)
Stream ID	Stream ID do pacote (em formato hexadecimal)
Packets	Número de pacotes recebidos
Lost	Número de pacotes perdidos (amarelo, se houver pacotes perdidos)
Reord	Número de pacotes reordenados
Duplic	Número de pacotes duplicados
Rst	Número de resets ocorridos (amarelo, se houver)
Info	Demais informações, se necessárias (reservado para avisos especiais)

8.3 RESULTADOS OBTIDOS

8.3.1 ERRO: FALSO POSITIVO PARA PACOTES DUPLICADOS DEPOIS DE UM RESET

Quando ocorria um *Reset* de barramento, o programa passava a acusar erroneamente a ocorrência de pacotes duplicados. Isso acontecia pois a lista dos últimos pacotes recebidos era mantida. Quando um pacote era recebido depois de um reset, se seu número de sequência era encontrado na lista de pacotes anteriores, este era considerado um pacote duplicado. Por exemplo, suponha que antes da lista ser completamente preenchida, logo no início da comunicação, houve um *reset*. Ao detectar novamente os pacotes 1, 2, 3 e assim por diante, estes eram considerados duplicados.

A solução para esse problema é bastante simples: a ocorrência de um *reset* implica em recomeçar a escrita dos últimos pacotes recebidos, apagando toda a lista.

8.3.2 RESULTADOS: TESTE DE RECEBIMENTO E ENVIO DE PACOTES:

8.3.2.1 Testes variados

Para testar o programa em suas varias fases de criação, foram utilizados arquivos com pacotes conhecidos de mensagens de telemetria, onde já eram conhecidos os *Stream IDs*, o número de pacotes perdidos, ocorrências de resets, entre outros. Utilizando o NetMon, que é capaz de receber e enviar arquivos PCAP, esses arquivos eram enviados para o PktControl e verificavam-se os resultados obtidos. Durante a fase de teste de envio, o NetMon também era utilizado para receber os pacotes retransmitidos do PktControl e verificar erros de transmissão e perdas de pacotes.

No decorrer dos testes, os pequenos problemas de programação foram corrigidos, até o programa ser considerado pronto para o teste final.

8.3.2.2 Teste final

Para a verificação final do programa, utilizaram-se dois computadores em rede. O primeiro, continha o programa de Simulador de GPS e o programa GpsCap. O segundo computador executaria o programa PktControl e o NetMon.

O Simulador de GPS enviava via uma porta serial virtual um pacotes contendo mensagens de GPS (do tipo BESTVEL, BESTXYZ e OMNIHPPOS). Recebendo desta porta serial, o programa GpsCap capturava as três mensagens de GPS e as enviava separadamente no formato iNET-X, cada uma com um *Stream ID* diferente, para a rede, utilizando IP de destino *multicast* (IP 225.1.1.1).

Operando em modo promíscuo, no outro computador, o PktControl tinha o Adaptador de Rede de recebimento de pacotes selecionado, de modo a receber todos os pacotes recebidos na comunicação entre os dois computadores. Dentre os muitos pacotes, o programa deveria identificar somente os três pacotes iNET-X, referentes às três mensagens de GPS. A lista de *Stream IDs* a serem enviados, para esse teste continha o *Stream ID* de apenas uma das mensagens de GPS. Para a transmissão, o

programa enviava para outro adaptador de rede e outra porta de comunicação a mensagem de GPS. O NetMon estaria também em execução, recebendo essa única mensagem e verificando por erros de transmissão. No primeiro computador, outro programa Netmon estaria em execução, capturando as mensagens de GPs transmitidas pelo GpsCap.

O esquema simplificado do processo de teste encontra-se na Figura 8.2:

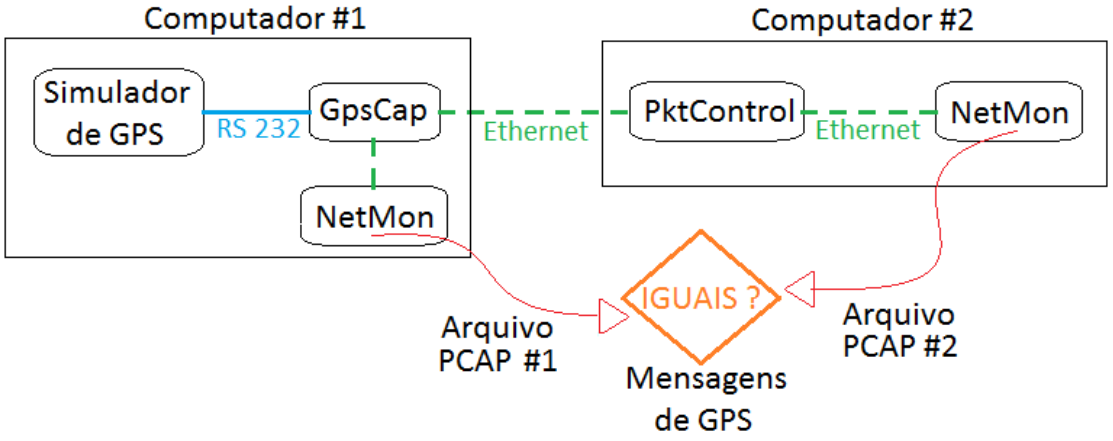

FIGURA 8.2 DIAGRAMA PARA O TESTE FINAL DOS PROGRAMAS GPSCAP E PKTCONTROL

O resultado do teste final foi positivo. Todas as mensagens de GPS foram corretamente recebidas e enviadas pela rede através do GpsCap. O mesmo ocorreu com os resultados de recebimento e transmissão do PktControl.

Na Tabela 8.2, encontra-se os resultados de um desses testes, indicando o número de pacotes enviados e recebidos por cada um dos programas:

TABELA 8.2 RESULTADOS DO TESTE FINAL DOS PROGRAMAS GPSCAP E PKTCONTROL

PktControl	Simulador GPS (enviados)	GpsCap (recebidos)	GpsCap (env.) #3 Strm IDs	PktControl (receb.)	PktControl (env.)	NetMon (receb.)
Send ALL	8934	8934	8934	8934 (2978 de cada)	8934	8934 (\2978 de cada)
1 Strm ID (AAA3)	8934	8934	8934	8934 (2978 de cada)	2978	2978

Além do teste de envio de apenas uma das mensagens de GPS através da seleção do *Stream ID*, o teste foi realizado com a opção de envio de todos os pacotes iNET-X, obtendo resultados igualmente positivos, sem perda de pacotes.

Para garantir que as mensagens foram enviadas sem alteração em suas informações, utilizou-se o NetMon para gravar em formato PCAP os pacotes enviados pelo PktControl. Em seguida, esse arquivo PCAP foi comparado ao arquivo PCAP de mensagens de GPS original, gerado pelo GpsCap e gravados também pelo NetMon.

Estes dois arquivos estavam idênticos, como era esperado, provando o correto funcionamento do sistema.

8.4 CONSIDERAÇÕES FINAIS E CONCLUSÕES

O programa PktControl atende aos requisitos desejados, sendo capaz de receber, verificar e retransmitir mensagens tipo iNET-X sem alterar seu conteúdo e sem perder mensagens durante o processamento.

Nenhum programa está livre de atualizações e pequenas correções. O PktControl ainda tem funções a serem adicionadas, como a possibilidade do usuário fornecer a lista de *Stream IDs* a serem enviados, função esta, atualmente em desenvolvimento.

Outra atualização futura importante é a expansão do código do programa para que este seja capaz de lidar com outros tipos de mensagens além do iNET-X. O sistema de telemetria utiliza inúmeros padrões e formatos de comunicação que podem ser adicionados ao processamento do PktControl.

Sobre a detecção de pacotes perdidos e reordenados, existe a proposta para que o programa seja capaz de identificar exatamente quais números de sequência dos pacotes estão de fato perdidos. Esse é um procedimento relativamente simples, onde o número de sequência dos pacotes perdidos é armazenado em um *buffer* de pacotes esperados. Quando um pacote é recebido de modo reordenado seu número de sequência é comparado aos números dos pacotes esperados, de modo a excluir do *buffer* o pacote recebido. No fim do recebimento, o programa pode fornecer a lista de pacotes não recebidos (aqueles que não foram excluídos do *buffer*). Embora seja um processo possível de ser realizado, este procedimento não está entre as funções exigidas para o PktControl, ao menos por enquanto. A informação do número de pacotes perdidos é suficiente para análise.

Por fim, através desse projeto, será possível controlar a comunicação entre a *Ground Station* e a aeronave, desde a seleção das informações a serem enviadas até a verificação da qualidade desta comunicação. O usuário poderá saber em tempo real as ocorrências de *resets*, perda e recuperação de pacotes. Desse modo, o sistema de telemetria se torna mais robusto, permitindo a comunicação *Ground Station-A*eronave nas duas direções.

Capítulo 9. ATUALIZAÇÃO DE OUTROS PROGRAMAS ANTERIORMENTE CRIADOS

Com o desenvolvimento desse projeto para o sistema de telemetria, é necessária a atualização do programa utilizado na validação dos arquivos gravados durante o ensaio em voo, chamado PCheck. Anteriormente, esse programa tinha função de ler e verificar saltos no número de sequência em todos os pacotes, de todos os *Stream IDs* gravados no arquivo. Muitas vezes, esses arquivos possuem vários gigabytes de tamanho, referentes a algumas horas de testes.

Embora não exija processamento em tempo real, o PCheck precisa ser muito veloz, capaz de verificar arquivos muitos extensos em questão de poucos minutos.

Com o acréscimo do sistema de comunicação bidirecional entre Aeronave e *Ground Station* e o surgimento da possibilidade de pacotes reordenados, o PCheck não pode simplesmente verificar por saltos de número de sequência.

Utilizando uma lógica de programação semelhante àquela do PktControl, atualizou-se o modo de operação do PCheck para que ele também passe a verificar *resets*, perda de pacotes, pacotes em duplicata e pacotes reordenados.

Em adição às alterações realizadas, o programa PCheck está em fase de desenvolvimento de sua mais recente versão, onde verificará não só erros de pacotes no formato iNET-X, mas também será capaz de tratar de modo diferenciado todos os tipos e padrões de mensagens, encapsulados dentro do pacote iNET-X, também verificando por possíveis erros de comunicação.

Um exemplo é a alteração no tratamento de pacotes do tipo PCM. Dentro de um mesmo pacote para PCM iNET-X, existem vários *minor frames*, todos com o próprio número de sequência. Este número é cíclico, alcançando um valor máximo antes de recomeçar sua contagem. Na nova versão do PCheck, em adição à verificação de erros de sequência do pacote iNET-X, os pacotes detectados com informações PCM (através do conhecimento de seu *Stream ID*), são lidos interiormente e o número de sequência de seus *minor frames* também são validados, verificando *frames* fora de ordem, ou falta de *frames* intermediários. Esse processo só é possível por ser conhecida a estrutura desses pacotes, a localização das informações desejadas, o número máximo de *minor frames* por pacote e o número de sequência máximo de *minor frames*.

As atualizações prosseguirão, até que todos os tipos de pacotes e informações sejam englobados, contribuindo para uma validação cada vez mais detalhada e segura dos dados da aeronave.

Outra atualização do PCheck, foi a adição do comando “Get GPS”. Nas versões antigas, o PCheck era utilizado também para a extração dos pacotes desejados de um

arquivo PCAP gravado durante um ensaio. O botão “Get PCM”, por exemplo, extraia todos os pacotes PCM relativos ao *Stream ID* desejado e criava um novo arquivo, com extensão apropriada para outros programas, capazes de ler e tratar esse tipo de mensagem. Com o novo botão “Get GPS”, o PCheck passa a extrair mensagens de GPS geradas pelo GpsCap e armazená-las num arquivo de extensão .csv (*comma-separated values*), onde a mensagem de GPS pode ser lida em um programa como o Microsoft Excel, por exemplo, com as informações separadas em colunas.

Capítulo 10. DESENVOLVIMENTOS FUTUROS

O início da Segunda Fase deste projeto tem como principal objetivo o desenvolvimento de um programa que envie dados da estação meteorológica para a aeronave.

Inicialmente, o programa em desenvolvimento busca os dados meteorológicos no *site* interno do Sistema de Telemetria da Embraer, trata esses dados e os transmite via Ethernet.

De modo semelhante ao desenvolvimento dos programas anteriores, o novo programa exige o estudo detalhado da estação meteorológica, dos tipos de dados envolvidos e dos métodos a serem utilizados em seu código. Também será utilizado o sistema de *threads*, para aumentar a velocidade de processamento. Com o desenvolvimento de novos módulos do aplicativo, ocorrerão variados testes de desempenho, garantindo a correção de possíveis erros de programação e o cumprimento dos requisitos desejados.

Capítulo 11. CONCLUSÃO

A aquisição e processamento de dados em uma aeronave são extremamente importantes, não só para auxiliar o piloto em sua tarefa, mas também para todos aqueles responsáveis pela aeronave de alguma maneira.

Desde o momento em que uma aeronave é fabricada, e por toda sua vida, seus dados serão armazenados. Esses dados servirão como base para estudos de qualidade da aeronave, de seu desempenho, seus defeitos, entre outros.

Através dos programas desenvolvidos pelo setor de Engenharia de Desenvolvimento e Integração de Sistemas Computacionais e Processamento de Dados de Ensaios, tanto os controladores da aeronave, quando a estação em terra, podem ter acesso a informações importantes, como altitude, velocidade, inclinação, aceleração, temperatura, entre outros. Esses dados são disponibilizados em tempo real para os engenheiros e pilotos.

Durante a criação de uma nova aeronave, seu protótipo será exaustivamente testado e avaliado. Os dados adquiridos em todos os testes serão armazenados e estudados. Em uma aeronave protótipo, seu interior possui apenas a parte de instrumentação, com grandes *raques* de equipamentos, monitores, *switches*, cabos e câmeras. Todo esse equipamento será no futuro condensado e miniaturizado, de modo a liberar espaço para os futuros assentos da aeronave.

Além da condensação dos equipamentos, é importante também a utilização de métodos de contorno de possíveis erros ou falhas de equipamento, também chamado de Redundância. Ou seja: para todo equipamento de telemetria, há outro como reserva, que passará a funcionar caso o primeiro venha a apresentar problemas. Isso é feito também para cabos de alimentação e de dados, fornecendo caminhos alternativos em caso de defeitos ou problemas. Assim como os equipamentos de instrumentação, os próprios sensores e transdutores da aeronave estão presentes de forma redundante. A falta de alguma informação para o piloto ou a estação em solo pode acarretar em acidentes graves.

A comunicação PCM (*Pulse Code Modulation*) possui algumas desvantagens, como a sincronização do *clock* (sincronização de mensagens recebidas e enviadas) e principalmente a detecção de erros de comunicação. Em cada tempo determinado, uma nova informação é enviada e o dado é armazenado pelo sistema. Porém, caso não haja alteração no estado atual da informação, a mesma informação enviada anteriormente é reenviada. Caso ocorra um erro na aquisição de dados, ou algum outro problema que impeça a aquisição de um novo dado, ocorrerá novamente o reenvio da mesma informação. Dessa forma, não se pode identificar a diferença entre uma falha e a não alteração de um dado, ao verificar os arquivos armazenados.

Na comunicação via Ethernet, entretanto, não ocorre o reenvio de uma informação antiga. Para cada intervalo de tempo, é enviada a informação nova, ou uma

sequência específica predeterminada caso seja detectado algum erro ou falha. Para esse tipo de comunicação, se não houver dados novos a serem enviados, o sistema permanece sem enviar nada, o que indica outra vantagem: a diminuição do envio de informações desnecessárias, diminuindo, consequentemente, o gasto de energia para isso. Além disso, a comunicação via Ethernet é mais veloz e cada informação ocupa menos memória do que a comunicação PCM.

Em relação ao projeto em andamento, pode-se ver a importância do recebimento e envio correto de mensagens do GPS. Através dessas mensagens, a aeronave terá informações sobre seu atual posicionamento e o comportamento de seu deslocamento.

Outra importância do programa está na possibilidade de sincronização entre todas as máquinas e programas na aeronave, pois cada informação recebida e gravada está relacionada a uma estampa de tempo. Assim é possível relacionar várias informações, vindas de sensores e programas diferentes. A informação de tempo do GPS é bastante apurada e sofre ajustes constantes, ideal para este tipo de utilização.

A criação de uma nova mensagem de GPS a partir das três recebidas diminui o espaço necessário para o armazenamento destas e facilita a identificação de informações perdidas em um dado tempo.

Atualmente, receptores de GPS mais modernos já possuem a opção de envio de mensagens via Ethernet. O programa desenvolvido permite também a continuação da utilização de receptores antigos, podendo ser aplicado em aeronaves que já estão em circulação, sem a necessidade de trocar seus equipamentos interiores.

Buscando garantir a recuperação de pacotes de informações perdidos durante a comunicação, o sistema de telemetria deve sofrer alterações, tornando-se mais robusto. A recuperação de pacotes, entretanto, acarreta a possibilidade de ocorrência de pacotes com número de sequência fora de ordem, ditos pacotes reordenados. É importante que o sistema de telemetria esteja preparado para lidar com esse tipo de pacote, sem acusar erroneamente o numero de pacotes perdidos. As informações recuperadas podem ser tratadas e dispostas novamente em ordem, caso seja necessário uma análise mais detalhada dos dados recebidos.

REFERÊNCIA BIBLIOGRÁFICA

[1] L-3 Telemetry-West, “PCM Tutorial,” L-3 Telemetry-West, [Online]. Available: http://telemetry-products.com/sites/default/files/PCM_Tutorial.pdf. [Acesso em Outubro 2012].

[2] R. G. Seippel, Transducers, sensors & detectors, Reston Pub. Co., 1983.

[3] J. Brignell e N. White, Intelligent Sensor Systems, Londres: IOP Pub., 1994.

[4] K. Ogata, Modern Control Engineering, 5 Ed, Prentice Hall, 2010.

[5] K. Feher, Telecommunications Measurements, Analysis, and Instrumentation, SciTech Publishing, 1996.

[6] L-3 Telemetry-West, “Telemetry Tutorial,” L-3 Telemetry-West, [Online]. Available: http://www.tw.l-3com.com/tutorial/preface.html. [Acesso em Setembro 2012].

[7] Curtiss Wright Controls - Avionics & Electronics, “Technical Note 050,” Curtiss Wright Controls, 2012.

[8] NovAtel, OEMV Family Firmware Reference Manual - Version 3.200, NovAtel Inc., 2007.

[9] Microsoft, “Visual Studio 2012,” [Online]. Available: http://www.microsoft.com/visualstudio/ptb. [Acesso em Março 2012].

[10] Wireshark Foundation , “Wireshark,” [Online]. Available: http://www.wireshark.org/. [Acesso em Fevereiro 2012].

[11] Microsoft, “Microsoft Developer Network,” [Online]. Available: http://msdn.microsoft.com/en-us/default.aspx. [Acesso em Fevereiro 2012].

[12] Riverbed Technology, “WinPcap,” [Online]. Available: http://www.winpcap.org/. [Acesso em Agosto 2012].

[13] Curtiss Wright Controls - Avionics & Electronics, “Curtiss Wright Controls,” [Online]. Available: http://www.cwc-ae.com/. [Acesso em Abril 2012].

[14] AGARD - Advisory Group for Aerospace Research & Development, Flight Test Instrumentation Series - Vol 1, Lancaster, Califórnia: AGARD, 1974.

[15] NovAtel, “NovAtel,” [Online]. Available: http://www.novatel.com/. [Acesso em Setembro 2012].

[16] OmniSTAR, “OmniSTAR,” [Online]. Available: http://www.omnistar.com/. [Acesso em Setembro 2012].

[17] J. O. Strock, Introduction to Temeletry, Instrument Society of America, 1987.

[18] C. Spurgeon, Ethernet: The Definitive Guide, O'Reilly Media, Inc., 2000.

[19] A. P. Bishop, The Role of Computer Networks in Aerospace Engineering, 1993.

[20] D. Reynders e E. Wright, Practical TCP/IP and Ethernet Networking for Industry, Newnes, 2003.

[21] Curtiss Wright Controls - Avionics & Electronics, iNET-X Handbook, Curtiss Wright Controls, 2012.