INSTITUTO FEDERAL DE SANTA CATARINA

YAN LUCAS MARTINS

Protocolo de aplicação multiuso com canais assíncronos sobre o protocolo USB-CDC

São José - SC

abril/2021

PROTOCOLO DE APLICAÇÃO MULTIUSO COM CANAIS ASSÍNCRONOS SOBRE O PROTOCOLO USB-CDC

Pré-Projeto de trabalho de conclusão de curso apresentado à Coordenadoria do Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a aprovação do tema perante banca na disciplina de TCC1.

Orientador: Emerson Ribeiro de Mello

Coorientador: Roberto de Matos

São José - SC

abril/2021

RESUMO

Dispositivos *Universal Serial Bus* (USB) estão presentes no cotidiano das pessoas; desde *mouses* e teclados à *modens* e maquina de cartão de crédito. Ao desenvolver um dispositivo USB, para cada sistema operacional e plataforma na qual o dispositivo USB irá se comunicar, há um custo de produção durante a definição e implementação de um *driver* USB que seja capaz de atender as necessidades do dispositivo. Isso também implica que sistemas operacionais e plataformas menos populares não recebam suporte ao dispositivo. Este trabalho tem o objetivo de estudar e propor um protocolo de aplicação multiuso com múltiplos canais assíncronos que seja capaz de codificar mensagens e permitir a comunicação dispositivo-*host* de maneira assíncrona sobre a classe USB-*Communication Device Class* (CDC), evitando a necessidade do desenvolvimento de um *driver* proprietário.

Palavras-chave: USB-CDC. Protocolo. Canais assíncronos.

ABSTRACT

USB devices are present in everyday life, from mice and keyboards to modems and credit card machines. When developing a USB device, for every operating system and platform on which the USB device will communicate, there is a production cost during the definition and implementation of a USB driver that is capable of meeting the needs of the device. This also implies that less popular operating systems and platforms are not supported by the device. This paper aims to study and propose a multi-purpose application protocol with multiple asynchronous channels that is able to encode messages and allow device-host communication asynchronously over the USB-CDC class, avoiding the need for the development of a proprietary driver.

Keywords: USB-CDC. Protocol. Asynchronous channels.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Dispositivos USB estão presentes no dia a dia de uma grande parcela da população. Dentre eles pode-se destacar *mouses*, teclados, impressoras, *modens*, telefones e uma infinidade de outros dispositivos (COMPAQ et al., 2000).

Apesar da sua simplicidade de uso, quando conecta-se um dispositivo USB a um *host*, sendo o último geralmente um computador, pode haver a necessidade da instalação de *drivers*, os quais são responsáveis por garantir a compreensão da troca de informação entre o dispositivo USB e o *software* do *host* (COMPAQ et al., 2000). No entanto, para cada sistema operacional diferente que um *host* possa ter, por vezes é necessário desenvolver um *driver* específico para o mesmo, visto que os *drivers* de dispositivo USB geralmente são implementados com base em um determinado sistema operacional e plataforma (KARNE et al., 2012).

A especificidade ao desenvolver um *driver* de acordo, pode implicar em um prazo maior para o desenvolvimento de um dispositivo USB específico, pois cada sistema operacional possui suas particularidades de funcionamento sendo necessário conhecer individualmente como cada um opera, o que se mal implementado, impede que a comunicação entre dispositivo-*host* ocorra. Além disso, a necessidade de desenvolver uma massiva quantidade de *drivers* implica em um custo elevado de produção do periférico. Bem como, origina-se a necessidade de escolha, impedindo que um dispositivo USB seja compatível com qualquer *host*, acarretando que apenas sistemas operacionais mais populares possam receber a compatibilidade com o periférico. A partir desta dificuldade, este trabalho propõe um protocolo que irá operar sobre a classe USB-CDC. Desta forma, o único requisito que o dispositivo USB e o *host* devem possuir é o de prover suporte ao *driver* dessa classe.

A comunicação USB pode dar-se de duas maneiras. Na primeira, a síncrona, quando o dispositivo envia uma mensagem, esse permanece em um estado bloqueado, ou seja, não é capaz de realizar qualquer outra atividade a não ser aguardar a resposta do *host*. Neste cenário, somente uma mensagem por vez é trafegada pelo canal de comunicação e em um único sentido. Já no segundo caso, o assíncrono, ao enviar uma mensagem, o dispositivo permanece em um estado não bloqueante, permitindo que o mesmo possa realizar outra atividade independente de ter recebido ou não uma resposta do *host*. O *host*, por sua vez, irá responder apenas quando quiser ou quando puder.

Os dispositivos podem operar com um ou múltiplos canais. Quando o protocolo provê suporte a múltiplos canais de comunicação, o mesmo deve ser capaz de garantir a ordenação das mensagens vindas de diferentes fontes. Tal garantia pode ser determinada através do estabelecimento de sessões para cada canal ativo e mecanismos de controle de acesso ao meio, sendo esse meio o barramento USB. O acesso ao meio pode ser dado de três formas: por meio de um sistema de particionamento do canal por tempo, particionamento aleatório ou revezamento entre canais. Múltiplos canais assíncronos permitem que mais de uma atividade possa ser executada pelo dispositivo USB e *host*.

Outro obstáculo é a limitação de banda existente em um barramento USB, o qual, também, está suscetível à versão do dispositivo USB que o ocupa (COMPAQ et al., 2000). Logo, é necessário conhecer esses limites para que o empacotamento de mensagens esteja de acordo com o requisito do periférico. Tratando-se de um protocolo de uso genérico, as mensagens trafegadas devem estar em uma forma concisa, permitindo que o protocolo seja executado em dispositivos com baixa capacidade de memória e em barramentos com limitação de banda inferiores aos padrões definidos por especificações USB recentes.

Considerando as dificuldades de desenvolver *drivers* exclusivos a cada sistema operacional e o heterogêneo mercado de dispositivos compatíveis com o protocolo USB, neste trabalho é proposto uma implementação de um protocolo de aplicação multiuso com canais assíncronos sobre o protocolo USB-CDC.

1.1 Objetivo geral

O objetivo geral deste trabalho é propor um protocolo de aplicação multiuso com canais assíncronos sobre o protocolo USB-CDC, que seja capaz de atuar como uma camada de abstração sobre o protocolo USB. Este protocolo, deve inibir a necessidade de instalar *drivers* adicionais no sistema operacional do *host*, empacotar e desempacotar mensagens concisas e comunicar-se em modo de múltiplos canais assíncronos. Deste modo, atuará como um *middleware* que busca minimizar o tempo de desenvolvimento de dispositivos USB.

1.2 Objetivos específicos

Para atingir o objetivo geral, foram definidos os seguintes objetivos específicos:

1. Identificar os serviços essencias que o protocolo deve atender;
2. Definir um procedimento que garanta o envio de mensagens em um meio concorrido com múltiplos canais assíncronos;
3. Desenvolver uma prova de conceito da solução;
4. Elaborar um plano de testes para a validação da proposta.

1.3 Organização do texto

O presente documento está organizado da seguinte forma: o Capítulo 2 aborda os principais fundamentos teóricos julgados necessários para o desenvolvimento da compreensão e execução deste trabalho. Introduzindo o funcionamento do USB, princípios de projeto e propriedades desejáveis de protocolos e formatos de mensagens. Já o Capítulo 3, apresenta a proposta de desenvolvimento do trabalho, além de definir um cronograma para execução da mesma.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 USB

O USB é um padrão que define cabos, conectores e protocolos usados em um barramento para conectar, comunicar e fornecer energia entre computadores e outros dispositivos eletrônicos. Computadores pessoais, *tablets* e telefones, que têm portas USB, podem se conectar a dispositivos, desde teclados, *mouses* e *joysticks* a câmeras, impressoras, dispositivos de áudio, vídeo e outros. O USB é uma tecnologia versátil, confiável, rápida, de baixo consumo energético, barata e provê suporte a diversos sistemas operacionais (AXELSON, 2015).

2.1.1 Arquitetura USB

Uma comunicação USB requer um *host* com suporte a USB e dispositivos com uma porta USB. O *host* é um computador pessoal, um dispositivo portátil ou qualquer outro sistema que contém um *hardware* de *host controller* USB e um *root hub*. O *host controller* possui todas as permissões sobre o barramento, sua responsabilidade, juntamente com o *root hub*, é detectar a conexão e remoção de periféricos USB, gerenciamento do fluxo de controle e dados entre *host* e os dispositivos USB, bem como o fornecimento de energia para o periférico. Toda comunicação em um barramento são iniciadas pelo *host*, deste modo, não pode haver comunicação direta entre dispositivos USB (AXELSON, 2015).

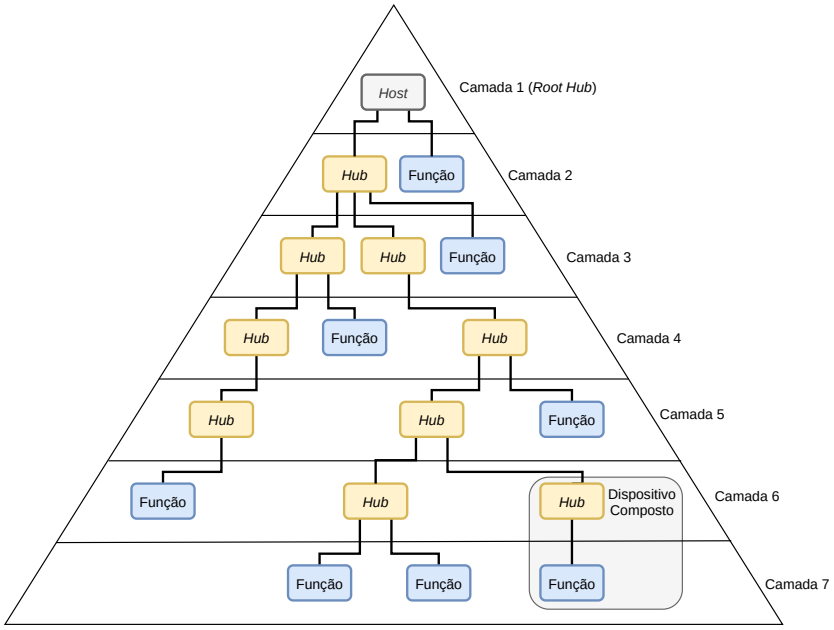
Os dispositivos USB, de maneira sucinta, são todos aqueles que compreendem e respondem operações do protocolo USB, estando esses divididos em duas categorias, *hub* e função. O conjunto dessas duas categorias é denominado dispositivo composto (COMPAQ et al., 2000).

- ***Hub***: cada *hub* converte um único ponto de conexão em vários pontos de conexão chamados de portas USB.
- **Função**: fornece recursos ao *host*, exemplos de um dispositivo deste tipo são: *mouse*, teclado ou impressora. Cada dispositivo de função contém informações de configuração que descrevem os recursos do mesmo, a fim de serem reconhecidos pelo *host*.
- **Dispositivo Composto**: um dispositivo composto contém um *hub* USB e um ou mais dispositivos de função, ocupando assim duas camadas; portanto, não pode ser habilitado se anexado no nível de camada sete.

A rede USB é baseada em uma topologia estrela em camadas, na qual há um dispositivo principal, o *host*, e até 127 dispositivos trabalhadores, o que inclui dispositivos de função e *hubs*. Sendo esses divididos em sete camadas: a primeira para o *host*, a sétima para dispositivos de função e as cinco demais camadas para *hubs* e dispositivos de função. Um dispositivo de função pode ser conectado a um *hub*, e esse *hub* pode ser conectado a outro *hub* e assim sucessivamente; desde que respeite o limite de sete camadas, conforme Figura 1 (COMPAQ et al., 2000).

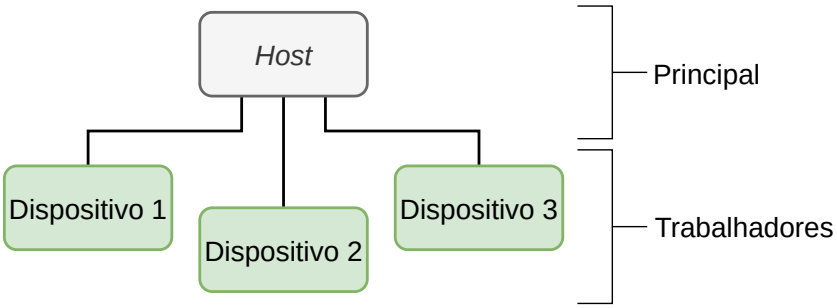
A rede USB lógica, por sua vez, aparece apenas como uma rede em estrela, sendo o centro da mesma, o *host*. Os *hubs* são transparentes neste cenário. Um dispositivo USB deve funcionar da mesma maneira se conectado diretamente ao *host* ou por meio de *hubs* intermediários, veja a Figura 2 (AXELSON, 2015).

Figura 1 – Topologia de uma rede de dispositivos USB.



Fonte: Adaptada de (Compaq et al. (2000))

Figura 2 – Topologia de uma rede lógica de dispositivos USB.



Fonte: Adaptada de (Compaq et al. (2000))

Todos os dispositivos USB estão disponíveis com nós endereçáveis nesta rede principal-trabalhador, graças a um endereço único fornecido pelo *host* para cada dispositivo USB presente na rede. Quando o *host* está transmitindo um pacote de dados, ele é enviado a todos os dispositivos conectados. Cada *hub* ressincroniza as transições de dados à medida que os retransmite. Unicamente um dispositivo, o endereçado, realmente aceita os dados; todos os outros recebem, mas os rejeitam. Apenas um dispositivo por vez é capaz de transmitir para o host, em resposta a uma solicitação direta do host. Cada *hub* repete todos os dados que recebe de uma camada inferior apenas em direção da camada superior (AXELSON, 2015).

2.1.2 Transferência de dados

Os dispositivos USB são classificados por velocidade do barramento. A versão 2.0 do USB (COMPAQ et al., 2000) especifica três dessas velocidades: *Low Speed*, *Full Speed* e *High Speed* conforme Tabela 1. Na versão 3.2 (APPLE et al., 2017), ainda são especificadas mais duas velocidades para um barramento: a *Super Speed* e a *Super Speed Plus*.

Tabela 1 – Vazão USB por especificação.

Velocidade	**Vazão máxima teórica**	**Aplicação**
Low Speed	1,5 Mbps	Dispositivos interativos: *mouse*, teclado e periféricos para jogos
Full Speed	12 Mbps	Áudio, impressoras e *scanners*
High Speed	480 Mbps	Vídeo e armazenamento em massa
Super Speed	5 Gbps	Vídeos em alta definição e armazenamento em massa
Super Speed Plus	10 Gbps	Vídeos em alta definição e armazenamento em massa

Fonte: Adaptada de Microchip (2020)

As transferências de dados USB ocorrem por meio de uma série de eventos chamados transações. Os pacotes de uma transação são classificados em: *token*, o qual descreve o tipo da transação; *data*, sendo este a carga útil de uma transação; e *handshake*, o qual define o estado de uma solicitação de comunicação. As transações são conduzidas em um intervalo de tempo controlado pelo *host*, denominado *frame*. A duração e a frequência das transações dependem do tipo de transferência que está sendo usado por um *endpoint*, conforme detalhado na subseção 2.1.3. O USB *frame* dura 1 milissegundo para dispositivos configurados em *Full* e *Low Speed* e 125 microssegundos para barramentos com *High Speed* (COMPAQ et al., 2000).

2.1.3 Modos de transferência USB

Os *endpoints* ou pontos de extremidade, podem ser descritos como fontes ou coletores de dados em um dispositivo USB, estando localizados na extremidade do fluxo de comunicação entre um dispositivo USB de função e um *host*. Devido a restrição de ocupação do barramento, os *endpoints* devem aguardar uma ordem de envio ou recepção de dados, vindo diretamente do *host*. Quando um dispositivo de função possui dados a serem enviados, cabe aos *endpoints* manter esses dados armazenados em *buffer*, aguardando até que recebam a autorização para a transmissão dos mesmos pelo barramento (COMPAQ et al., 2000).

É previsto na versão 2.0 do protocolo USB (COMPAQ et al., 2000) quatro tipos de *endpoint descriptors*, isto é, modos de transmissão diferentes para cada *endpoint* que um dispositivo USB possa ter. Cada modo fornece características diferentes de tratamento de erros, latência garantida, largura de banda e sentido de transmissão (COMPAQ et al., 2000). Esses modos de transmissão são introduzidos nos itens a seguir:

- ***Control transfer***: este modo é responsável por comunicações de comando, *status* ou configuração entre o dispositivo e o *host*. Todo dispositivo USB deve possuir pelo menos um *endpoint* com este modo de transferência, denominado de *endpoint 0*. Normalmente a transmissão dos pacotes ocorre em rajadas, iniciada pelo *host* e sem garantia de recebimento.

- ***Isochronous transfer***: neste modo existe uma garantia de largura de banda, taxa de transmissão e latência limitada. O sentido da transmissão será sempre uni-direcional. Existe a detecção de erros porém sem retransmissão. A característica da comunicação é contínua e periódica e geralmente contém informações sensíveis ao tempo como *stream* de áudio e vídeo.

- ***Interrupt transfer***: este modo de transmissão é utilizado quando o dispositivo precisa enviar ou receber informação do *host* sem uma determinada frequência porém com um período de transmissão bem determinado. Desta forma, este modo de operação garante um máximo período de transmissão para o dispositivo, bem como tentativas de retransmissão no próximo período em caso de erros de transmissão.

- ***Bulk transfer***: este modo é utilizado quando o dispositivo precisa transmitir grandes quantidades de informação e utilizar o máximo de largura de banda disponível no momento. Desta forma, este modo possui retransmissão em caso de erro, garantia de entrega dos dados transmitidos. No entanto, não é garantido neste modo largura de banda ou baixa latência.

Cada modo de transferência possui um limite de carga útil diferente, por consequência, emprega diferentes taxas de transferência para cada velocidade do barramento. Para melhor observação, esses dados foram estruturados na tabela Tabela 2.

Tabela 2 – Características de vazão no barramento por modo de transmissão

Velocidade	Modo de transmissão	Carga útil máxima	Transferência por frame	Vazão máxima teórica
Low Speed	Control	8 bytes	1	8 kB/s
Low Speed	Interrupt	8 bytes	1	8 kB/s
Low Speed	Bulk	não se aplica	não se aplica	não se aplica
Low Speed	Isochronous	não se aplica	não se aplica	não se aplica
Full Speed	Control	64 bytes	1	64 kB/s
Full Speed	Interrupt	64 bytes	1	64 kB/s
Full Speed	Bulk	64 bytes	até 19	1.2 MB/s
Full Speed	Isochronous	1023 bytes	1	1023 kB/s
High Speed	Control	64 bytes	1	512 kB/s
High Speed	Interrupt	1024 bytes	até 3	24 MB/s
High Speed	Bulk	512 bytes	até 13	53 MB/s
High Speed	Isochronous	1024 bytes	até 3	24 MB/s

Fonte: Microchip (2020)

O endereço de *endpoint* consiste em um número do *endpoint* e uma direção. O número do *endpoint*, para a transferência de dados, é um valor inteiro no intervalo de 1 a 15. A direção, por sua vez, é definida a partir da perspectiva do *host*: um *endpoint* IN fornece dados para enviar ao *host* e um *endpoint* OUT armazena os dados recebidos do *host*. O par zero é reservado ao *endpoint* de controle denominado *endpoint* zero. Este é o ponto que recebe todas as solicitações de controle e *status* dos dispositivos durante todo o tempo em que o dispositivo está operacional no barramento (COMPAQ et al., 2000).

2.1.4 Descritores USB

Dispositivos USB transmitem suas informações por meio de descritores, que são blocos de dados com formatos definidos e estruturados de forma hierárquica. Cada dispositivo USB possui quatro descritores padrão (COMPAQ et al., 2000):

- ***Device Descriptor***: representa as informações essenciais do dispositivo USB, existindo apenas um descritor deste tipo por periférico. O descritor possui informações como versão do protocolo, identificação de fabricante e a quantidade de *configuration descriptors*, por exemplo;
- ***Configuration Descriptor***: um dispositivo USB pode ter um ou mais *configuration descriptors*. Este é responsável por indicar informações como número de interfaces, consumo de energia do dispositivo e modo de alimentação;
- ***Interface Descriptor***: abaixo do *configuration descriptor* pode haver um ou mais *interface descriptors*. Este, por sua vez, é responsável por indicar a classe USB utilizada, bem como indicar quantos *endpoint descriptors* estão abaixo desta interface;
- ***Endpoint Descriptor***: este descritor é utilizado pelo *host* para, por exemplo, determinar o total de largura de banda que deve ser alocado pelo barramento, além de definir o modo de operação do *endpoint*, o que impacta diretamente na taxa de transferência do dispositivo.

2.1.5 Classes USB

A especificação do protocolo USB (COMPAQ et al., 2000) define classes para diferenciar e identificar as funcionalidades de um dispositivo USB, permitindo ao *host* carregar o *driver* necessário de acordo com a classe do dispositivo. Essa divisão, permite que dispositivos com funcionalidades semelhantes possam fazer uso do mesmo *driver*, não havendo a necessidade do desenvolvimento de um *driver* exclusivo para todo dispositivo. Esta seção busca determinar uma classe que ofereça uma máxima taxa de transferência e que possua drivers pré-instalados nos sistemas operacionais mais comuns.

2.1.5.1 Classe HID

A classe *Human Interface Device* (HID) inclui dispositivos, como teclados, *joysticks* e *mouses*. Para todos esses dispositivos, o *host* recebe dados que correspondem à entrada humana, como pressionamentos de tecla e movimentos do *mouse*. Além dos descritores padrão, conforme apresentados na subseção 2.1.4, esta classe possui três descritores específicos: *HID Descriptor*, *Report Descriptor* e *Physical Descriptor*. Um dispositivo com a classe HID se comunica com o *host* a partir de *endpoints* no modo de transferência *control* ou *interrupt* (AXELSON, 2015).

2.1.5.2 Classe CDC

A classe CDC é comumente utilizada em dispositivos de telecomunicações, tal quais *modens* e telefones, além de outras funções de comunicação, como portas seriais virtuais. Em sua especificação é definido que todo dispositivo pertencente a esta classe, deve possuir um descritor de interface denominado *Communications Interface Class*, o qual lida com o gerenciamento de controle do dispositivo. Esse gerenciamento gera notificações que podem ser transferidas utilizando o modo *interrupt* ou *bulk*. Um dispositivo do tipo CDC pode ainda conter uma interface para a troca de dados chamada de *Data Interface Class*. Para a troca de dados pode-se utilizar tanto o modo *bulk* quanto o modo *isochronous* (AXELSON, 2015). Por fazer uso desses dois últimos modos de transferência citados, esta classe permite uma troca de dados maior que a classe HID.

2.2 Protocolo de comunicação

A comunicação de dados pode ser entendida como a troca de informação entre dois dispositivos por meio de algum ambiente de comunicação. Nesse ambiente, temos a mensagem que é a informação a ser transmitida; o transmissor e o receptor, responsáveis por transmitir e receber a mensagem respectivamente; o meio de comunicação que é o caminho físico por onde a mensagem será enviada; e por fim, o protocolo, o qual define um conjunto de regras para a transmissão e recepção da mensagem. Um modelo simplificado é apresentado na Figura 3.

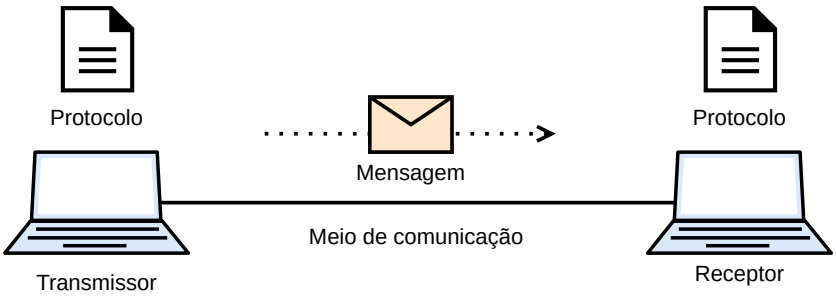
Todas as regras, formatos e procedimentos acordados entre transmissor e receptor são chamados de protocolo. Essa definição é importante pois formalizando a interação garante-se que os dados serão reconhecidos em ambas as extremidades da comunicação. Nas seções a seguir, são discutidas alguns elementos considerados essenciais em uma definição de protocolo.

2.2.1 Os elementos de um protocolo

Segundo Holzmann (1991), a especificação de um protocolo consiste em cinco elementos. Para uma especificação ser completa deve-se incluir:

1. O **serviço** a ser prestado pelo protocolo;

Figura 3 – Sistema básico de comunicação.



Fonte: Adaptada de (Holzmann (1991))

2. As **considerações** sobre o ambiente de comunicação em que o protocolo é executado;
3. O **vocabulário** das mensagens utilizadas pra a implementação do protocolo;
4. A **codificação** de cada mensagem do vocabulário;
5. O **comportamento** que define a consistência das trocas de mensagens.

Holzmann (1991) compara a definição de um protocolo com a definição de um idioma falado: um idioma contém um vocabulário e uma definição de sintaxe, o que representa o formato e codificação do protocolo; as regras de comportamento definem uma gramática; e a especificação do serviço define a semântica da linguagem. Holzmann (1991) ainda define que a linguagem do protocolo deve ser inequívoca, estando esta pronta para lidar com situações adversas como por exemplo, o tempo, condições de corrida e possíveis bloqueios (HOLZMANN, 1991).

2.2.2 Serviço e ambiente

Para que uma tarefa de nível superior possa ocorrer, como uma transferência de arquivos, uma variedade de funções de nível inferior, como sincronização e recuperação de erro devem ser executadas. Essas funções, devem ser introduzidas no protocolo de acordo com o ambiente que o mesmo irá operar, fazendo-se uma análise do meio de transmissão. A recuperação de erros, por exemplo, deve ser volátil, estando essa apta a corrigir o comportamento assumido em um meio de transmissão específico, o qual pode ser diverso como um canal ponto a ponto, dedicado à comunicação entre duas máquinas específicas, ou um canal de transmissão compartilhado, como a rede Aloha ou um enlace Ethernet (HOLZMANN, 1991).

O *software* do protocolo, é convenientemente estruturado em camadas. Funções mais abstratas são definidas e implementadas em termos de construções de níveis, onde cada camada oculta uma certa propriedade do canal de comunicação. Um *design* em camadas ajuda a indicar a estrutura lógica do protocolo, separando as tarefas de nível superior dos detalhes de nível inferior. Quando o protocolo deve ser estendido ou alterado, é mais fácil substituir um módulo do que reescrever todo o protocolo (HOLZMANN, 1991).

A *International Standards Organization* (ISO) reconhece as vantagens de padronizar uma hierarquia de serviços de protocolo, e estabelece um modelo de referência para projetistas de protocolo. A qual define sete camadas:

1. **Camada física:** transmissão de *bits* em um circuito físico.

2. **Camada de enlace de dados:** detecção e recuperação de erros.
3. **Camada de rede:** transferência e roteamento de dados.
4. **Camada de transporte:** transferência de dados de usuário para usuário. Trata controle de fluxo, ordenação dos pacotes e a correção de erros.
5. **Camada de sessão:** coordenação de interações em sessões entre *hosts*.
6. **Camada de apresentação:** interpretação da sintaxe em nível de usuário, por exemplo, para criptografia ou compressão de dados. Atua como um tradutor.
7. **Camada de aplicação:** ponto de entrada para processos de aplicação, promove uma interação entre máquina e usuário.

Cada camada na hierarquia define um serviço distinto e implementa um protocolo diferente. O formato usado por qualquer camada específica é amplamente independente dos formatos usados pelas outras camadas. A camada de rede, por exemplo, envia pacotes de dados, a camada de enlace de dados os converte em quadros e a camada física os traduz em *bytes* ou fluxos de *bits*. O receptor decodifica os dados brutos na camada 1, interpreta e exclui a estrutura do quadro na camada 2, para que a camada 3 possa reconhecer novamente a estrutura do pacote. O formato imposto pelas camadas inferiores deve ser transparente para as camadas superiores (HOLZMANN, 1991).

2.2.3 Vocabulário e formato

Existem alguns métodos para a formatação de mensagem. De acordo com Holzmann (1991), os três principais métodos de formatação são: orientado a *bits*, orientado a caractere e orientado a contagem de *bytes*. Esses, sem a adição de campos especiais, não previnem e não são capazes de tratar falhas. Em um protocolo orientado a caracteres, por exemplo, há apenas os códigos de controle, que são os bytes *start of text* (STX) e *end of text* (ETX) que servem como delimitadores de uma mensagem.

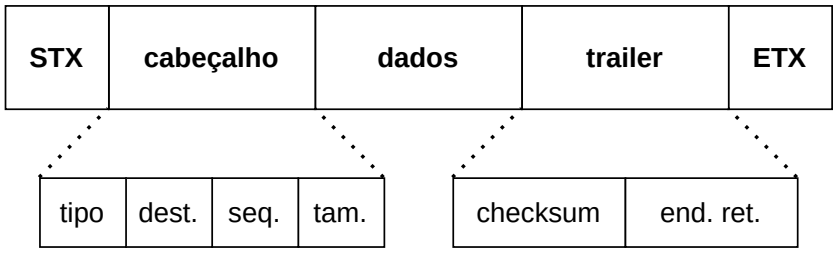
No entanto, um protocolo está suscetível a erros. Se um campo de contagem de bytes for distorcido ou um caractere de controle for perdido, o receptor poderá não identificar o quadro. São necessários esquemas de detecção de erros, os quais requerem a transmissão de informações redundantes, normalmente na forma de uma soma de verificação (*checksum*). Se forem adicionadas técnicas de controle de fluxo, por exemplo, para detectar perda ou reordenação de quadros de texto, um campo de número de sequência é anexado. Se mais de um tipo de mensagem for usado, deve-se incluir um identificador do tipo de mensagem que está sendo transferida (HOLZMANN, 1991).

Toda essa sobrecarga deve ser agrupada em estruturas separadas que encapsulam os dados. O formato da mensagem pode então ser definido como um conjunto ordenado de três elementos: cabeçalho, dados e trailer, conforme Figura 4. O cabeçalho e o trailer, por sua vez, definem subconjuntos ordenados de campos de controle, que podem ser definidos da seguinte forma:

- **Cabeçalho:** contendo o tipo, destino, número de sequência e tamanho, por exemplo.
- **Trailer:** incluindo, por exemplo, o *checksum* e endereço de retorno.

Não há uma regra geral sobre quais campos um protocolo deve possuir, é necessário conhecer o meio e os serviços a fim de julgar os campos vitais para o funcionamento do mesmo. Uma dessas adições é o campo tipo que pode ser usado para identificar as mensagens que compõem o vocabulário do protocolo. Dependendo da estrutura particular do vocabulário do protocolo, este campo pode ser ainda mais refinado (HOLZMANN, 1991).

Figura 4 – Formato da mensagem



Fonte: Adaptada de (Holzmann (1991))

2.2.4 Propriedades do protocolo

Ainda segundo Holzmann (1991), um protocolo deve possuir algumas propriedades desejáveis: simplicidade, modularidade, adequação, robustez e consistência.

- **Simplicidade:** quando um protocolo pode ser construído a partir de um pequeno número de partes bem projetadas e bem entendidas. Cada parte desempenha uma função e é capaz de executá-la de forma concreta. Para entender o protocolo, deve ser necessário apenas compreender o funcionamento de cada parte e a forma como interagem.
- **Modularidade:** quando cada parte do protocolo possui uma interação simples e bem definida. Cada parte menor é um protocolo que pode ser desenvolvido, verificado, implementado e mantido separadamente das demais partes. A estrutura de protocolo resultante é aberta, extensível e reorganizável, sem afetar o funcionamento das outras partes.
- **Adequação:** não é superespecificado, ou seja, não contém nenhum código inalcançável ou não executável. Tampouco deve estar incompleto. O protocolo deve ser limitado, não ultrapassando os limites do sistema, ser estável e adaptável.
- **Robustez:** deve funcionar em condições normais, bem como em situações imprevistas. Deve ser capaz de lidar com cada possível sequência de ações, em todas as possíveis condições. Deve ter um projeto mínimo, de forma a remover considerações não essenciais que poderiam impedir sua adaptação a condições não previstas.
- **Consistência:** protocolos não devem apresentar interações que os levem a falhar, tais como:
  - ***Deadlocks*:** estados nos quais nenhuma execução de protocolo adicional é possível, ou seja, quando todos os processos de protocolo estão esperando por outras condições que nunca podem ser cumpridas.
  - ***Livelocks*:** sequências de execução que podem ser repetidas indefinidamente, sem nunca realizar um progresso efetivo.
  - **Encerramentos inesperados:** a conclusão da execução sem satisfazer as condições de encerramento adequadas.

Em geral, esses critérios não podem ser verificados por uma inspeção manual da especificação do protocolo. Ferramentas mais poderosas são necessárias para evitá-los ou detectá-los. Algumas dessas ferramentas poderiam ser uma linguagem para modelagem de sincronização e coordenação de processos concorrentes como o *Protocol Meta Language* (PROMELA) e o verificador *Simple Promela Interpreter* (SPIN).

2.3 Formatos para representação de dados

Um formato de representação de dados define a forma que um conjunto de *bits* é organizado a fim de representar algo. Alguns formatos são projetados para tipos de dados muito específicos como: *Portable Network Graphics* (PNG) e *Graphics Interchange Format* (GIF), para o armazenamento de imagens; *MPEG-2 Audio Layer III* (MP3) e *Waveform Audio File Format* (WAV), para o armazenamento de áudios; *Portable Document Format* (PDF) e *Comma-separated values* (CSV), como formatos de arquivos de texto que contém um fluxo de caracteres. Ter um formato de representação de dados definido permite que este seja interpretado, armazenado e transferido corretamente.

2.3.1 Formatos textuais

Nesta subseção são expostos alguns formatos de representação de dados textuais. Uma das premissas para este modelo de representação, é a sua apresentação em um formato legível por humanos, os quais geralmente seguem uma representação ASCII ou Unicode (Maeda, 2012).

2.3.1.1 XML

Diferente do *HyperText Markup Language* (HTML), que tem seu uso focado em apresentação de dados em páginas *web*, o XML, por outro lado, originou-se a fim de ser uma linguagem de representação de dados que permite ser aplicada em uma variedade de contextos diferentes. Seus documentos são legíveis por humanos e podem ser facilmente interpretados por máquinas, o que permite que seu uso seja difundido em aplicações *Web Services*, por exemplo, onde há a necessidade de enviar solicitações e respostas através de uma linguagem universal intermediária (MYER, 2005).

A sintaxe básica consiste em definir *tags*, aqui denominados elementos, que melhor representam as informações que serão apresentadas. Todo documento XML deve conter um elemento raíz, sendo este pai de todos os demais elementos, conforme demonstra o Código 2.1.

Código 2.1 – Exemplo de um elemento raíz e seus descendentes em XML

```
1 <raiz>
2     <filho>
3         <neto>.....</neto>
4     </filho>
5 </raiz>
```

O prólogo ou cabeçalho XML, veja Código 2.2, é um elemento opcional, porém, se existir, deve ser o primeiro item do documento. O prólogo, tem como principal função, definir o tipo de codificação que o documento está representado. Se não definido, por padrão, todo documento XML será representado em codificação UTF-8.

Código 2.2 – Exemplo de um prólogo em XML

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

Existem algumas regras para o uso das *tags* em um documento XML: elas são *case sensitive*, ou seja, apresentam a diferenciação entre letras minúsculas e maiúsculas, a *tag* `<nome>` é diferente da *tag* `<Nome>`; todo elemento deve possuir uma *tag* de fechamento, e toda *tag* de fechamento deve ser encerrada dentro da mesma *tag* em que foi aberta. O Código 2.3 apresenta na linha 2 o aninhamento incorreto das *tags*, enquanto que a linha 5 apresenta o formato correto.

Código 2.3 – Exemplo de aninhamento de *tags* em XML

```
1 <!-- Uso incorreto -->
2 <pessoa><nome>Lucas</pessoa></nome>
3
4 <!-- Uso correto -->
5 <pessoa><nome>Lucas</nome></pessoa>
```

Os dados em um documento XML não possuem tipos. Logo, todos os dados são definidos como *strings* (W3C, 2008). O Código 2.4 contém um exemplo de uma estrutura de dados em XML, cujos elementos `<nome>`, `<apelido>`, `<idade>`, `<altura>`, `<filhos>`, `<vivo>`, `<saldo>`, `<id>` e `<senha>` contêm texto. Os elementos `<pessoa>`, `<financeiro>` e `<senhas>` contêm elementos. E, por sua vez, o elemento `<telefone>` contém um atributo, o qual seu valor sempre deve estar entre aspas.

Código 2.4 – Representação de dados em XML

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <pessoa>
3    <nome>Lucas Martins</nome>
4    <apelido>Lucas "Galaxy" Martins</apelido>
5    <idade>47</idade>
6    <altura>1.85</altura>
7    <filhos>Maria</filhos>
8    <filhos>Vinicius</filhos>
9    <filhos>Amanda</filhos>
10   <vivo>true</vivo>
11   <telefone xsi:nil="true"/>
12
13   <financeiro>
14     <saldo>-10</saldo>
15     <senhas>
16       <id>1</id>
17       <senha>123</senha>
18     </senhas>
19     <senhas>
20       <id>2</id>
21       <senha>9999</senha>
22     </senhas>
23   </financeiro>
24 </pessoa>
```

2.3.1.2 JSON

O JSON foi projetado para ser uma linguagem de troca de dados legível por humanos e de fácil análise e uso por computadores. As informações presentes neste modelo tendem a ser mais concisas, apresentando um documento mais limpo. Por fazer uso apenas do símbolo de chaves para a delimitação das *tags* de marcação, quando dados são codificados nesta linguagem, o resultado tende a ser menor que outras codificações equivalentes, como o XML (BRAY, 2017).

Para cada valor a ser representado, atribui-se uma uma *tag*, denominada nome. O nome deve estar entre aspas duplas, e o par nome-valor deve estar separado por dois pontos. Os valores a serem representados podem ser tipos primitivos como:

- ***String*:** este tipo consiste em uma coleção de caracteres em Unicode, cujo valor para este tipo sempre deve ser escrito entre aspas duplas. Deve-se também, utilizar barras invertidas como caractere de *escape*. Um exemplo deste tipo é apresentado no Código 2.5 nas linhas 3 e 4;

- **Número:** este tipo permite números inteiros ou decimais, positivos e negativos, em base decimal. O JSON não provê suporte as bases octal e hexadecimal. Diferente do tipo *string*, este não deve estar entre aspas, veja as linhas 5, 6 e 12 do Código 2.5;
- **Booleano:** o valor booleano, ou lógico, é um tipo de dado primitivo que possui dois valores, verdadeiro ou falso. A nomenclatura utilizada para este tipo no JSON é *true* e *false*, respectivamente, um exemplo de uso é exibido no Código 2.5 conforme linha 8;
- **Nulo:** o valor nulo indica que os dados são ausentes ou desconhecidos. Destaca-se que nulo é diferente de valor vazio ou com valor zero. Um elemento JSON com valor nulo deve ser indicado com a palavra *null* conforme Código 2.5 na linha 9.

Código 2.5 – Representação de objeto em JSON

```
1  {
2      "pessoa": {
3          "nome" : "Lucas Martins",
4          "apelido": "Lucas \"Galaxy\" Martins",
5          "idade" : 47,
6          "altura": 1.85,
7          "filhos" : ["Maria", "Vinicius", "Amanda"],
8          "vivo" : true,
9          "telefone": null,
10
11         "financeiro" : {
12             "saldo": -10.00,
13             "senhas" : [ [1,123], [2,9999]]
14         }
15     }
16 }
```

A partir dos tipos primitivos, é possível elaborar tipos complexos:

- **Matriz:** é uma coleção de valores. Este tipo deve ser iniciado com um colchete de abertura e encerrado com um colchete de encerramento. Os valores individuais devem ser separados por vírgulas, conforme apresentado no Código 2.5 nas linhas 7 e 13;
- **Objeto:** é uma coleção de pares nome-valor. Este tipo deve ser iniciado com uma chave de abertura e encerrado com uma chave de encerramento. Cada par nome-valor devem ser separados por vírgulas, conforme exibido no Código 2.5 onde há dois objetos: `pessoa` e `financeiro`, linhas 2 e 11 respectivamente.

2.3.2 Formatos binários

Formatos de representação de dados binários, são aqueles que não são legíveis por humanos. A principal vantagem no uso deste tipo de formato, são documentos menores quando comparados ao formato de representação de dados textual.

2.3.2.1 MessagePack

O MessagePack é um formato de serialização binário desenvolvido por Sadayuki Furuhashi que destaca-se pela capacidade de codificar inteiros pequenos em um único *byte*, e *strings* curtas requerem apenas um *byte* extra. Este formato possui implementações oficiais com código aberto em diversas

linguagens, dentre elas pode-se destacar: C, C++, Go, Java, Python e outras (MESSAGEPACK, 2018). No Código 2.6 há uma representação do objeto JSON do Código 2.5 em MessagePack.

Código 2.6 – Representação de dados em MessagePack

```
DF 00 00 00 01 A6 70 65 73 73 6F 61 DF 00 00 00 08 A4 6E 6F 6D 65 AD 4C 75 63 61 73 20 4D 61 72 74 69
6E 73 A7 61 70 65 6C 69 64 6F B6 4C 75 63 61 73 20 22 47 61 6C 61 78 79 22 20 4D 61 72 74 69 6E 73 A5
69 64 61 64 65 2F A6 61 6C 74 75 72 61 CB 3F FD 99 99 99 99 99 9A A6 66 69 6C 68 6F 73 DD 00 00 00 03
A5 4D 61 72 69 61 A8 56 69 6E 69 63 69 75 73 A6 41 6D 61 6E 64 61 A4 76 69 76 6F C3 A8 74 65 6C 65 66
6F 6E 65 C0 AA 66 69 6E 61 6E 63 65 69 72 6F DF 00 00 00 02 A5 73 61 6C 64 6F CB C0 24 00 00 00 00 00
00 A6 73 65 6E 68 61 73 DD 00 00 00 02 DD 00 00 00 02 01 7B DD 00 00 00 02 02 CD 27 0F
```

A estrutura da codificação define que o primeiro *byte* determina qual é o tipo de dado que está armazenado, seguido pela informação. Em algumas estruturas, como *Raw*, *Array* e *Map* é adicionado um terceiro campo, que contém o tamanho do campo da informação (MESSAGEPACK, 2018).

2.3.2.2 CBOR

O CBOR é um formato de dados cujos objetivos de *design* incluem a possibilidade de tamanho de mensagem e código reduzidos e extensibilidade sem a necessidade de negociação de formato, ou seja, não é necessário haver um esquema definido, o decodificador é capaz de decodificar um único item do CBOR sem nenhum outro conhecimento (BORMANN, 2013). Esses objetivos o diferenciam de outros formatos binários, como *Abstract Syntax Notation One.* (ASN.1) e MessagePack (MESSAGEPACK, 2018). No Código 2.7 há uma representação do objeto JSON do Código 2.5 em CBOR.

Código 2.7 – Representação de dados em CBOR

```
A1                                      # map(1)
   66                                   # text(6)
      706573736F61                      # "pessoa"
   A8                                   # map(8)
      64                                # text(4)
         6E6F6D65                       # "nome"
      6D                                # text(13)
         4C75636173204D617274696E73     # "Lucas Martins"
      67                                # text(7)
         6170656C69646F                 # "apelido"
      76                                # text(22)
         4C7563617320224761616C61787922204D617274696E73 # "Lucas \"Galaxy\" Martins"
      65                                # text(5)
         6964616465                     # "idade"
      18 2F                             # unsigned(47)
      66                                # text(6)
         616C74757261                   # "altura"
      FB 3FFD999999999999A              # primitive(4611010478483282330)
      66                                # text(6)
         66696C686F73                   # "filhos"
      83                                # array(3)
         65                             # text(5)
            4D61726961                  # "Maria"
         68                             # text(8)
            56696E6963697573            # "Vinicius"
         66                             # text(6)
            416D616E6461                # "Amanda"
      64                                # text(4)
         7669766F                       # "vivo"
      F5                                # primitive(21)
```

```
68                                  # text(8)
   74656C65666F6E65                 # "telefone"
F6                                  # primitive(22)
6A                                  # text(10)
   66696E616E636569726F             # "financeiro"
A2                                  # map(2)
   65                               # text(5)
      73616C646F                    # "saldo"
   F9 C900                          # primitive(51456)
   66                               # text(6)
      73656E686173                  # "senhas"
   82                               # array(2)
      82                            # array(2)
         01                         # unsigned(1)
         18 7B                      # unsigned(123)
      82                            # array(2)
         02                         # unsigned(2)
         19 270F                    # unsigned(9999)
```

2.3.3 Comparativo entre formatos de codificação de mensagens

Para a realização do comparativo, foi utilizado um objeto JSON base (veja Código 2.5), o qual foi utilizado como documento de entrada para os formatos CBOR (veja Código 2.7) e MessagePack (veja Código 2.6). Esse mesmo objeto JSON base foi convertido para XML, conforme Código 2.4. Fundado no resultado dessas conversões, foi verificado o tamanho de cada formato de representação de dado em sua forma minificada, ou seja, foram removidos todos os caracteres desnecessários sem alterar sua funcionalidade. O resultado desta análise está na Tabela 3.

Tabela 3 – Comparativo entre formatos de codificação de mensagens.

Formato	**Representação**	**Tamanho (Bytes)**
CBOR	Binária	165
JSON	Textual	220
MessagePack	Binária	191
XML	Textual	408

A partir da Tabela 3, percebe-se que os formatos de representação binários oferecem documentos menores que as representações textuais. Sendo o CBOR o qual apresenta o menor tamanho e o XML o maior. Esse tamanho superior encontrado no XML se deve principalmente ao fato de seus delimitadores para as *tags* serem maiores que os delimitadores utilizados no JSON.

3 PROPOSTA

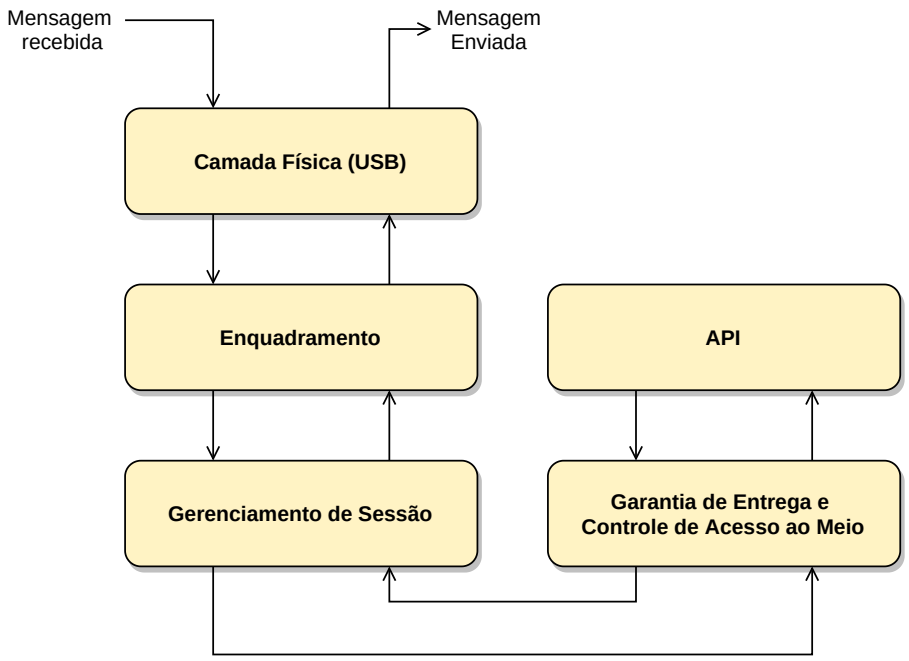
A proposta deste trabalho é desenvolver um protocolo de aplicação com múltiplos canais assíncronos sobre o USB-CDC. O protocolo atuará como uma camada de abstração acima do USB-CDC, permitindo a comunicação entre um dispositivo USB-CDC e um *host*, sem que haja a necessidade do desenvolvimento de *drivers* específicos para cada sistema operacional ou plataforma. A escolha da classe CDC é dada devido a sua maior taxa de transmissão quando comparada a outra classe USB popular como o HID, presente em *mouses*, teclados e outros dispositivos de interface humana. Outra vantagem do uso da classe CDC é o suporte nativo que sistemas operacionais como Android, Linux, macOS, Windows e outros fornecem à classe CDC. Remover a necessidade da implementação de um *driver* proprietário, acarreta em um custo menor no desenvolvimento de dispositivos USB e aumenta a abrangência de sistemas operacionais e plataformas que irão prover suporte ao dispositivo.

A comunicação assíncrona garante que o dispositivo USB-CDC não permaneça em um estado bloqueado enquanto aguarda uma resposta do *host*, podendo esse ocupar o tempo de espera com a realização de outra atividade. O protocolo deve estar apto a operar com um ou mais canais em execução, sendo necessário reservar um campo no cabeçalho responsável por identificar e gerenciar o dispositivo e o canal utilizado. Para a modelagem do protocolo e garantir o seu funcionamento completo, deverão ser especificados todos os serviços que o protocolo irá prover, o vocabulário e o seu comportamento. A fim de apoiar nessa etapa, serão produzidos diagramas de máquina de estados finita que irão representar o comportamento de cada subcamada do protocolo, sendo estas definidas na Figura 5. A subcamada Enquadramento será responsável por delimitar e identificar o tipo da mensagem, a Gerenciamento de Sessão adicionará um campo a fim de identificar cada canal operante e, por fim, a Garantia de Entrega e Controle de Acesso ao Meio irá inserir campos para controle, como número de sequência, além de gerenciar o acesso ao barramento USB de cada canal.

A fim de verificar todas as propriedades do protocolo, uma simples observação humana, por meio de uma máquina de estados finita, pode não garantir que as propriedades do protocolo não sejam violadas e de que eventos como *deadlocks*, *livelocks* ou encerramentos inesperados ocorram. Nesse caso, faz-se necessário o uso da linguagem para modelagem de sincronização e coordenação de processos concorrentes PROMELA e o verificador SPIN. O PROMELA é capaz de descrever protocolos de comunicação por meio da criação de canais de mensagens, o modelo permite a análise de mensagens síncronas ou assíncronas. Um benefício dessa linguagem é a sua abstração aos detalhes do modelo que não são relacionados à interação entre os processos que modelam o comportamento dos canais. Já o SPIN irá realizar a verificação do que será implementado em PROMELA e validar o modelo.

Definido o modelo do protocolo, propõe-se a implementação do mesmo em algumas plataformas a fim de verificar o seu comportamento em um ambiente real. A Raspberry Pi Zero W possui uma porta *host* USB On-The-Go (OTG) o que permite a mesma operar em modo *device* e modo *host*, sendo assim, por possuir o modo *device* é possível permitir com que a placa opere como um dispositivo alvo da proposta deste trabalho. Para o *host* planeja-se a implementação em dois computadores pessoais: um com Linux e um outro com Windows, e em um *smartphone* Android, os três sistemas operacionais também contêm *drivers* USB-CDC nativos. Segundo StatCounter (2020), o Windows e o Android, foram os dois sistemas operacionais mais populares em computadores pessoais e em *smartphones* no ano de 2020, respectivamente. Já o Linux, é um sistema operacional menos popular que os dois citados e com estatísticas de uso também inferiores às do macOS. Ao propor o desenvolvimento em sistemas com diferentes níveis

Figura 5 – Hierarquia das camadas do protocolo proposto.

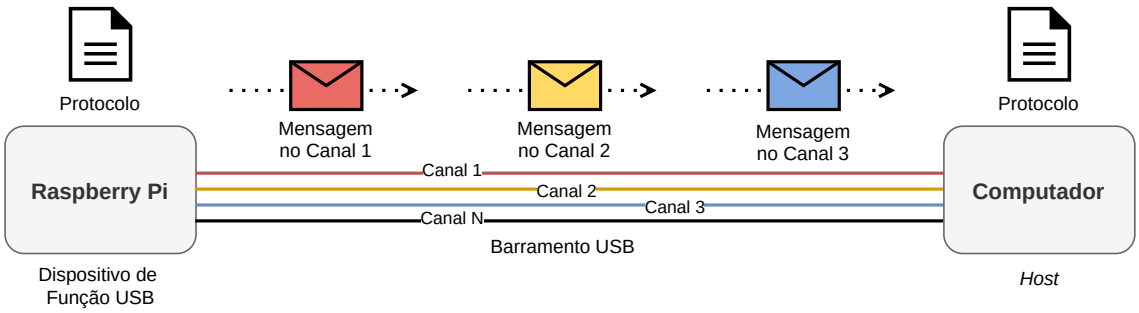


Fonte: Própria

de popularidade, busca-se provar que independente da plataforma ou do sistema operacional, o protocolo deverá proporcionar dificuldades de implementação semelhantes, estando essas mais atreladas a escolha da linguagem de programação do que ao próprio sistema ou plataforma.

Os testes serão realizados por meio da conexão da Raspberry Pi via cabo USB a um dos três *hosts* citados. O protocolo deverá estar em execução em cada uma das extremidades da comunicação. A Figura 6 exemplifica o cenário de teste da validação do protocolo.

Figura 6 – Cenário de comunicação entre dispositivo USB e *host*.



Fonte: Própria

Para validar a solução a ser implementada, alguns testes deverão ser feitos. Espera-se que o dispositivo USB seja reconhecido e encontre-se em modo operante entre todos os *hosts* por meio do protocolo proposto. Isso significa que: não deve ser realizada a instalação de nenhum *driver* extra, as mensagens devem ser compreendidas entre ambas as extremidades, o protocolo deve ser resistente a falhas e capaz de ressincronizar-se de maneira autônoma. Serão enviadas mensagens válidas e inválidas por meio de N canais assíncronos, além de forçar dessincronizações simulando possíveis falhas. Também será

verificado o desempenho do protocolo quando este é submetido a múltiplos canais assíncronos.

3.1 Cronograma de atividades

Diante do panorama inicial desta proposta, a fim de alcançar o objetivo deste trabalho, um cronograma de atividades previstas foi elaborado de acordo com a Tabela 4.

Tabela 4 – Cronograma das atividades previstas

Atividades	**Semanas**																	
	Mai.			**Jun.**				**Jul.**					**Ago.**				**Set.**	
	01	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
A1	√	√	√															
A2				√	√	√												
A3						√	√	√	√	√								
A4											√	√						
A5						√							√	√				
A6															√	√		
A7			√			√				√		√		√		√	√	√

- A1: modelagem do protocolo assíncrono;
- A2: implementação do protocolo no dispositivo;
- A3: implementação do protocolo nos *hosts*;
- A4: estudo das questões de compatibilidade entre diferentes *hosts*;
- A5: testes das funcionalidades definidas no protocolo;
- A6: testes de desempenho na utilização de múltiplos canais assíncronos;
- A7: documentação do desenvolvimento.

REFERÊNCIAS

APPLE et al. *Universal Serial Bus 3.2 Specification*. [s.n.], 2017. Disponível em: <https://www.usb.org/document-library/usb-32-specification-released-september-22-2017-and-ecns>. Citado na página 18.

AXELSON, J. *USB Complete: The Developer's Guide*. [S.l.]: Lakeview Research, 2015. Citado 3 vezes nas páginas 17, 18 e 21.

BORMANN, P. H. C. *Concise Binary Object Representation (CBOR)*. 2013. Disponível em: <https://tools.ietf.org/html/rfc7049>. Acesso em: 10 mar. 2020. Citado na página 28.

BRAY, E. T. The javascript object notation (json) data interchange format. Dez. 2017. Disponível em: <https://tools.ietf.org/html/rfc8259>. Acesso em: 15 abr. 2020. Citado na página 26.

COMPAQ et al. *Universal Serial Bus Specification*. [s.n.], 2000. Universal Serial Bus Specification Revision 2.0. Disponível em: <https://www.usb.org/document-library/usb-20-specification>. Citado 6 vezes nas páginas 15, 17, 18, 19, 20 e 21.

HOLZMANN, G. J. *Design And Validation Of Computer Protocols*. [S.l.]: Prentice Hall Software Series, 1991. Citado 4 vezes nas páginas 21, 22, 23 e 24.

KARNE, R. K. et al. A bare pc mass storage usb driver. *Computer and Information Sciences,Towson University*, 2012. Citado na página 15.

Maeda, K. Performance evaluation of object serialization libraries in xml, json and binary formats. In: *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*. [S.l.: s.n.], 2012. p. 177–182. Citado na página 25.

MESSAGEPACK (Ed.). *MessagePack specification*. 2018. Disponível em: <https://github.com/msgpack/msgpack/blob/master/spec.md>. Citado na página 28.

MICROCHIP (Ed.). *USB Speeds and Specifications*. 2020. Disponível em: <https://microchipdeveloper.com/usb:speeds>. Citado 2 vezes nas páginas 19 e 20.

MYER, T. A really, really, really good introduction to xml. Ago. 2005. Disponível em: <https://www.sitepoint.com/really-good-introduction-xml/>. Acesso em: 15 abr. 2020. Citado na página 25.

STATCOUNTER. Desktop operating system market share worldwide. 2020. Disponível em: <https://gs.statcounter.com/os-market-share/mobile/worldwide/2020>. Acesso em: 15 abr. 2021. Citado na página 31.

W3C. Extensible markup language (xml) 1.0 (fifth edition). Nov. 2008. Disponível em: <https://www.w3.org/TR/xml/>. Acesso em: 15 abr. 2020. Citado na página 26.