Projeto de Protocolos

Guilherme Roque
Engenharia de Telecomunicações, Instituto Federal de Santa Catarina
<guilhermelroque@gmail.com>

Marcone Augusto
Engenharia de Telecomunicações, Instituto Federal de Santa Catarina
<marcone.p.louzada@gmail.com>

Vinicius Luz
Engenharia de Telecomunicações, Instituto Federal de Santa Catarina
<viniciusls22@gmail.com>

Resumo- *Este relatório tem por objetivo descrever o funcionamento e a implementação de um protocolo de enlace ponto a ponto desenvolvido na disciplina Projeto de Protocolos, ministrada pelo professor Dr Marcelo Maia Sobral no curso superior de Engenharia de Telecomunicações do Instituto Federal de Santa Catarina (IFSC). Seguindo um modelo de camadas, desenvolvemos um protocolo que oferece serviços como garantia de entrega, detecção de erros e controle de sessão, desta forma, permitindo, por exemplo, a comunicação entre dois nós da rede através de um enlace sem fio.*

Palavras-chave: Protocolo de enlace. Projeto de protocolo.

Abstract- *The purpose of this report is to describe the operation and implementation of a peer-to-peer link protocol developed in the Protocol Design discipline, taught by Professor Dr Marcelo Maia Sobral in the Telecommunications Engineering course at the Federal Institute of Santa Catarina (IFSC). Following a layered model, we have developed a protocol that offers services such as delivery assurance, error detection and session control, thus allowing, for example, communication between two network nodes via a wireless link.*

Keywords: Link protocol. Protocol Design.

1 Introdução

Um protocolo de comunicação está relacionado aos mecanismos necessários para a entrega de mensagens entre duas aplicações quaisquer. Considerando uma arquitetura de redes em camadas como TCP/IP, protocolos de comunicação correspondem às camadas de enlace até transporte. Questões como garantia de entrega, controle de sequência, tratamento de erros, estabelecimento e término de sessão e delimitação de mensagens, fazem parte do projeto de tais protocolos(WIKI, 2019).

Este relatório tem como objetivo descrever a implementação de um protocolo para estabelecimento de enlace sem-fio ponto-a-ponto utilizando um transceiver RF, que foi proposto pelo professor Dr. Marcelo Sobral, como forma de atividade na disciplina de PTC029008 - Projeto de Protocolos (2019.2).

2 Desenvolvimento

O protocolo de enlace desenvolvido se destina a links com baixa taxa de transmissão e suscetíveis a erros frequentes. As mensagens são curtas, ou seja, até 1024 bytes, e transmitidas de forma confiável. O seu modelo de funcionamento pode ser visto na Figura 1.

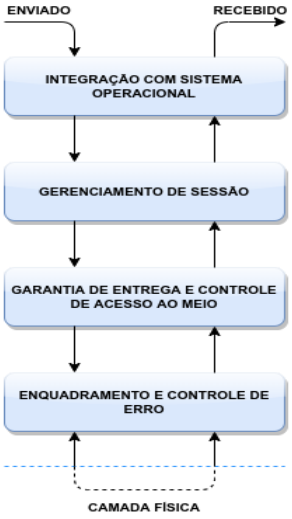

Figura 1 - Modelo do Protocolo

3 Implementação

O protocolo foi desenvolvido com o uso da linguagem C++. Na Figura 2, pode-se observar a tradução do modelo de funcionamento visto na Figura 1, na forma de implementação do protocolo com as suas devidas classes, divididas em camadas.

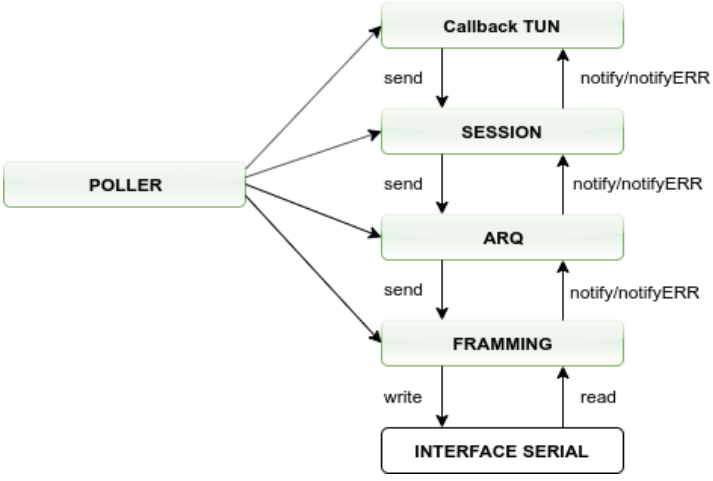

Figura 2 - Implementação do protocolo

O modelo de camadas facilita a implementação dos serviços que o protocolo irá oferecer, de modo que cada camada fica responsável por um fragmento do quadro e, desta forma, implementa a sua funcionalidade, permitindo o empilhamento de serviços. Na figura acima, as classes *Framming*, *ARQ* e *Session* são responsáveis pelo enquadramento, controle de acesso ao meio e controle de sessão, respectivamente (estes são os serviços oferecidos pelo protocolo). Nas seções subsequentes, detalharemos o funcionamento de cada uma destas camadas.

Além das camadas do protocolo, a figura 1 exibe as classes auxiliares *Poller* e *CallbackTun*. A classe *Poller* é um artificio utilizado para o desenvolvimento. Seu objetivo é monitorar entradas do sistema operacional e gerar *callbacks* para o nosso programa. Desta forma, podemos monitorar as entradas de dados na interface serial e na interface TUN, gerando *callbacks* para as camadas *Framming* e *CallbackTun*. Para as outras camadas (*ARQ* e *Session*), o *Poller* funciona como um timer não bloqueante, possibilitando o controle dos *timeouts* do protocolo.

A Figura 3, apresenta o diagrama UML do programa desenvolvido, a classe App apresentada no diagrama UML foi utilizada para execução de testes como uma *fake-layer*, antes da integração com o sistema operacional.

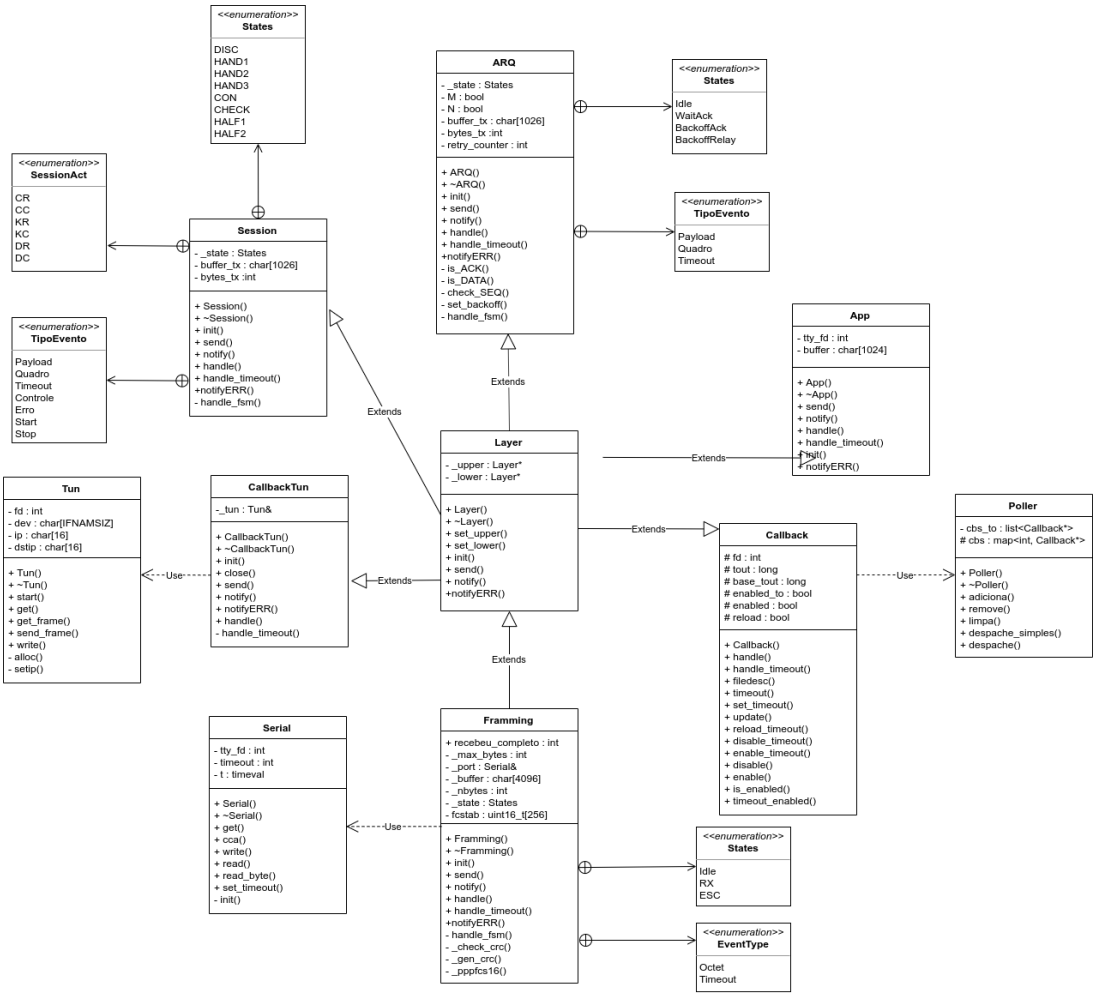

Figura 3 - Diagrama UML da implementação

3.1 Integração com sistema operacional

A integração com o sistema operacional é feita através da interface TUN. Interfaces Tun são interfaces apenas de software, o que significa que eles existem apenas no kernel e, diferentemente das interfaces de rede comuns, eles não têm componente físico de hardware. Quando o programa se conecta à interface tun, ele obtém um descritor de arquivo especial, cuja leitura fornece os dados que a interface está enviando. De maneira semelhante, o programa pode gravar neste descritor especial e os dados aparecerão como entrada para a interface tun (TUN, 2019). A Figura 4, apresenta a implementação da Interface TUN, baseada no modelo OSI (Open System Interconnection).

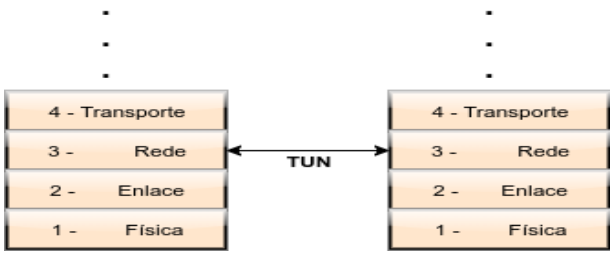

Figura 4 - Interface TUN

3.2 Gerenciamento de sessão

Responsável pelo gerenciamento de sessão, a camada *Session* é descrita pela máquina de estados exibida na Figura 5.

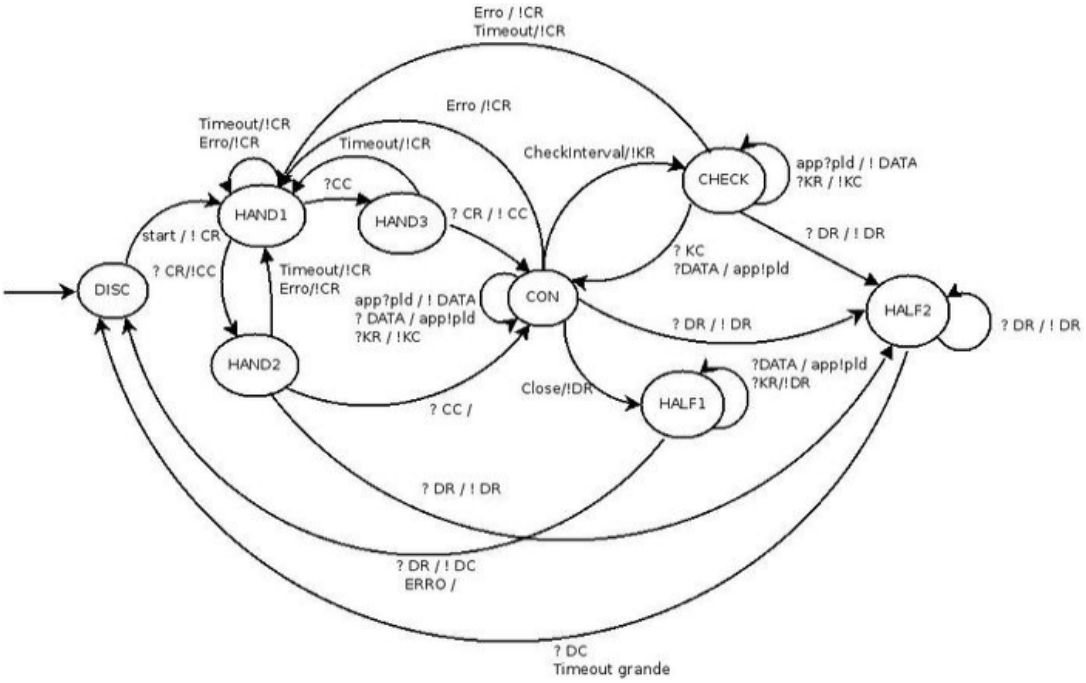

Figura 5 - Máquina de estados da classe Session

Esta máquina de estados é responsável por estabelecer uma sessão, mantê-la ativa e finalizá-la adequadamente, quando necessário. Para esta operação, é realizado um *handshaking* de 4 vias, de modo que ambos os comunicantes solicitem o início de uma sessão e enviem suas confirmações. Após a conexão ser estabelecida, um mecanismo de *Keep Alive* mantém a conexão ativa até que um dos lados perca a conexão ou solicite o encerramento da sessão. Para realizar este controle, é determinado um ID de sessão, que deve permanecer ativo enquanto a conexão se mantiver estabelecida. O protocolo tem capacidade de manter ativa uma única sessão, descartando, caso sejam recebidos, pacotes referentes à sessões diferentes da que está ativa no momento.

3.3 Garantia de entrega e controle de acesso ao meio

Para prover a garantia de entrega ARQ (Automatic Repeat reQuest) foi implementando o modelo Stop-and-Wait. Esta camada é responsável por enviar um dado e esperar a chegada de uma confirmação de recebimento pelo outro par comunicante. Caso esta confirmação não chegue, o mecanismo deve retransmitir o dado e novamente aguardar sua confirmação. Este processo é repetido três vezes até que o *ARQ*

indique um erro na transmissão para a camada superior. A confirmação de mensagens também é de responsabilidade desta camada. Na Figura 6, está representado a maquina de estados do ARQ.

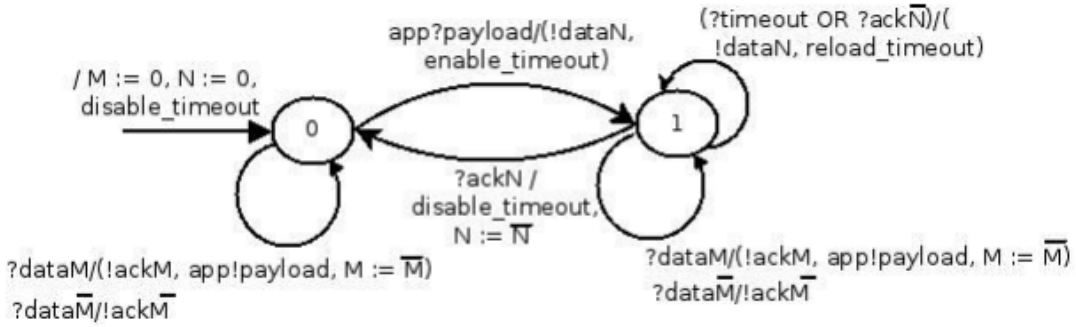

Figura 6 - Máquina de estados da classe ARQ

Para evitar a colisão durante o envio e recebimento de quadros, esta camada implementa o controle de acesso ao meio na camada de enlace baseado no protocolo Aloha. A cada intervalo de tempo em que o par comunicante não recebe a confirmação de recebimento de um quadro enviado, a estação irá aguardar um certo período de tempo aleatório, tentando evitar que ocorra uma colisão de quadro. Caso um par comunicante receba uma confirmação de recebimento de quadro, também ocorrerá uma espera aleatória antes que a estação esteja pronta para enviar o próximo quadro. Além disso, a máquina de estados preve um número de sequência de mensagens, de modo a garantir o sincronismo entre os comunicantes.

O modelo de funcionamento do protocolo aloha pode ser visto na Figura 7.

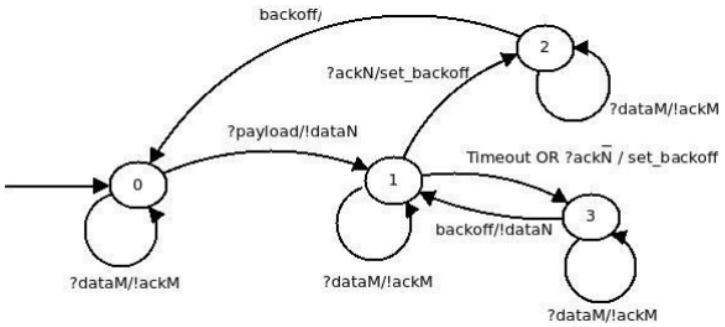

Figura 7 - Máquina de estados do Aloha

3.4 Enquadramento e controle de erros

O enquadramento é uma função do protocolo de enlace responsável por delimitar quadros na interface com a camada física. Deve-se ter em mente que a camada física oferece um serviço de envio e recepção de sequências de bytes sem qualquer estrutura. Cabe à camada de enlace delimitar as unidades de dados de protocolo (PDU) dentro dessas sequências de bytes(WIKI, 2019).

Existem diversas formas de abordagem para delimitar quadros, porém a técnica escolhida como mais adequada foi a Sentinela, especificamente a técnica aplicada pelo protocolo PPP (RFC 1661, 2019). Esta técnica foi a escolhida pois o protocolo foi pensado para funcionar através de um enlace sem fio, no qual requisitos temporais (sincronismo) são inviáveis e, além disso, o fato de não termos uma integração total com o *Hardware* do *Transceiver* inviabiliza a implementação de técnicas que necessitam deste requisito. Na técnica Sentinela implementada no protocolo PPP são

utilizadas as flags 7E (01111110) como delimitador de quadros e o byte de escape 7D (01111101) para preenchimento do octeto. Caso dentro do quadro ocorra a flag 7E, usa-se o byte de escape 7D para preenchimento do quadro e o byte seguinte é modificado através da operação lógica XOR com o valor 20. Ou seja, para transmitir o byte 7E no meio do quadro deve-se enviar a sequência 7D 5E. Ao receber o byte 7D, o par comunicante irá ter que traduzir o byte seguinte, novamente através do operação lógica XOR. A mesma lógica deve ser feita quando ocorra a necessidade de enviar o byte 7D dentro do quadro.

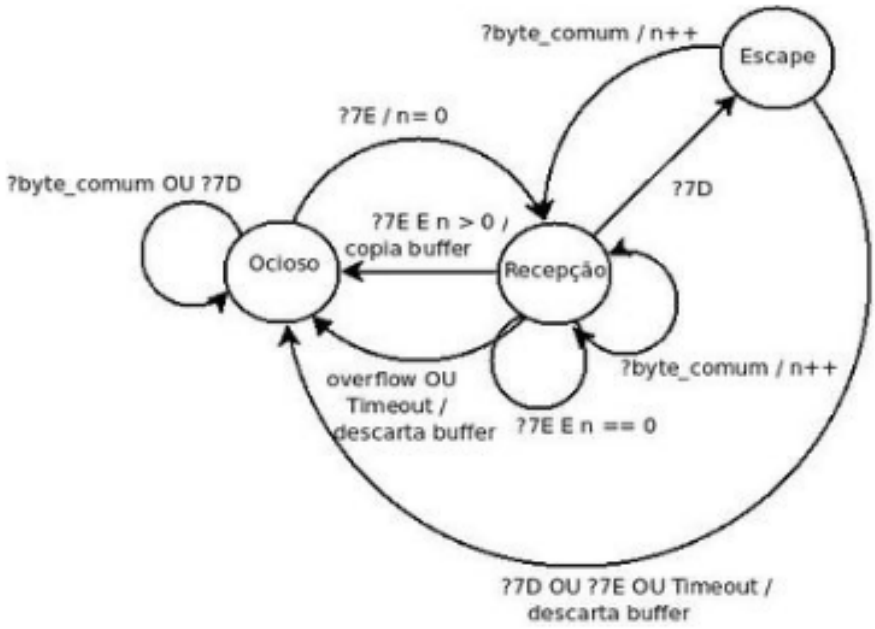

Figura 8 - Máquina de estados da classe Framming

A detecção e controle de erros é feita utilizando o CRC-16 (Cyclic Redundancy Check) que é uma técnica usada para detecção de erros na transmissão de dados digitais. No método CRC são adicionados check bits normalmente chamados de checksum. Estes são anexados à mensagem que irá ser transmitida. O receptor pode assim determinar se os check bits estão de acordo com os dados transmitidos e determinar com um certo grau de certezas se ocorreram erros na transmissão ou não.

3.5 Visão geral do protocolo

A Figura 9 exibe uma visão geral do protocolo, mostrando o fluxo de uma mensagem transmitida, sendo o transmissor a direita e receptor a esquerda.

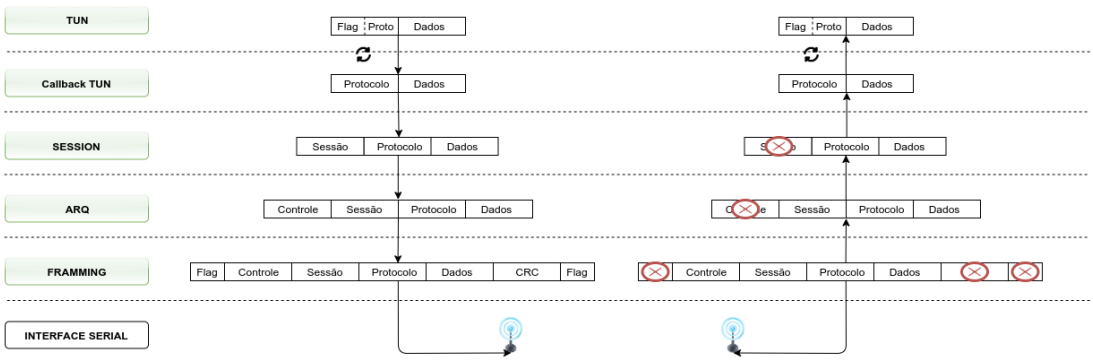

Figura 9 - Modelo de Funcionamento

Como é possível identificar, cada camada é responsável por ao menos um campo do quadro enviado/recebido. A camada *Framming* é responsável pelos campos *FLAG*

e *CRC*. A camada *ARQ* é responsável pelo campo *Controle*, a camada *SESSION* é a responsável pelo campo *Sessão* e, por fim, a camada *CallbackTun* faz a conversão do campo protocolo para o tipo utilizado no sistema operacional. Na transmissão, a medida que a mensagem desce para ser enviada, as camadas inserem seus cabeçalhos no quadro. Na recepção acontece o inverso, a medida que a mensagem sobe, as camadas vão retirando seus cabeçalhos, tratando-os e encaminhando a mensagem para a camada logo acima.

4 Execução do código

Para executar o código fornecido em conjunto com este relatório, primeiramente deve-se executar o arquivo *serialemu* indicando o valor 9600 *bps* (bits por segundo) no parâmetro taxa de bits, conforme descrito a seguir:

./serialemu -B 9600

O programa *serialemu* emula um link serial e, portanto, a saída do comando acima deverá mostrar os caminhos das duas portas seriais emuladas no sistema.

Após emular as portas seriais, para compilar o código do protocolo, acesse a pasta *PTC_CommunicationProtocol/Release* e execute o comando *make all*. Com o protocolo compilado, execute o binário gerado com o nome *PTC_CommunicationProtocol* informando os seguintes parâmetros:

./PTC_CommunicationProtocol <FD_serial> <id_sessao> <print_log> <IP_origem> <IP_destino>

Onde:

- FD_serial: porta serial emulada pelo programa *serialemu* (cada ponta do enlace deve executar em uma porta)
- Id_sessao: valor entre 0 e 255 que identificará a sessao (deve ser utilizado o mesmo valor para as duas pontas do enlace)
- Print_log: *true* ou *false*. Indicam se os logs devem ser exibidos ou não.
- IP_origem: IP de origem do respectivo lado do enlace
- IP_destino: IP de destino (outro lado do enlace)

Observação: caso não sejam informados os IPs de origem e destino, o protocolo será executado com um *link fake* no lugar da interface de rede (TUN).

5 Conclusão

O desenvolvimento de protocolos tem por objetivo criar uma linguagem comum entre diferentes sistemas e implementações, de modo que interajam entre si de maneira confiável e estável. Todavia, apesar de parecer simples, nem sempre é tão fácil definir um modelo de comunicação estável, que se recupere de falhas e com todas as situações previstas.

No desenvolvimento deste projeto, pudemos constatar o quão importante é a especificação e a definição dos estados de um protocolo. Sem essa definição bem clara, a implementação pode variar de um sistema para outro, fazendo com que o protocolo

não funcione da maneira correta. Além disso, observamos que muitas vezes podemos ir para a implementação e, só então, identificar um caso não mapeado, fazendo com que retornemos à fase de especificação. Desta forma, podemos dizer que este é um fluxo normal e importante para que tenhamos um protocolo estável.

Também foi possível observarmos vantagens do modelo em camadas, já utilizado em protocolos reais, que estão no mercado há bastante tempo. Com este modelo, é possível definirmos a responsabilidade de cada camada e incluir novos serviços ao protocolo de maneira eficiente.

Por fim, podemos dizer que o desenvolvimento do protocolo trouxe à tona as complexidades do estabelecimento de um linguajar comum entre sistemas computacionais, bem como boas práticas para elaboração de um protocolo estável, confiável e robusto.

Referências

RFC 1661. *O protocolo ponto a ponto (PPP)*. 2019. Disponível em: <https://tools.ietf.org/html/rfc1661>. Acesso em: 14 out. 2019.

TUN. *TUN layer*. 2019. Disponível em: <https://backreference.org/2010/03/26/tuntap-interface-tutorial/>. Acesso em: 14 out. 2019.

WIKI. *Pagina da disciplina PTC*. 2019. Disponível em: <https://wiki.sj.ifsc.edu.br/wiki/index.php/PTC29008:_Projeto_1:_um_protocolo_de_comunicaÃğÃčo>. Acesso em: 14 out. 2019.