PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
(PUCRS)

ESCOLA POLITÉCNICA

Jonathas Hernandes

T1 - Protocolo Ethernet e Raw Sockets

Porto Alegre

2021

Lista de ilustrações

Sumário

1 Introdução

Para a comunicação entre os diversos elementos de rede, são utilizados os mais diversos protocolos, cada um com suas próprias atribuições. Desta maneira, torna-se imprescindível para o conhecimento acerca do funcionamento de uma rede de computadores o conhecimento dos elementos que compõem a formação destes. Por esta razão, compreende-se a necessidade do aluno de ser capaz de desenvolver seu próprio protocolo a fim de compreender como os demais são formados.

Como objetivo de conhecer os conceitos relacionados a pilha de protocolo de redes, foi desenvolvida uma aplicação distribuída a partir da utilização de raw sockets. Com o objetivo de tornar possível a troca de breves informações entre os pares existentes na rede a partir da aplicação, compreende-se a necessidade da criação de um protocolo próprio para ela, a ser encapsulada no Protocolo Ethernet.

2 Desenvolvimento Protocolos

Para o desenvolvimento do trabalho conforme as especificações solicitadas, optou-se pela utilização da linguagem Python. Para tal, o primeiro passo a ser considerado foi a construção dos modelos de protocolos a serem enviados através do programa. A fim de adequar o formato optou-se a utilização do módulo “Struct”. Este módulo Python tem como função a realização de conversões entre os valores em Python e structs em C, representadas na linguagem via bytes. Considerando que os pacotes são enviados através da rede via bytes, compreende que faz sentido a utilização destes para o desenvolvimento dos pacotes que serão enviados.

Desta maneira, compreendendo o cabeçalho Ethernet, conforme observado na figura 1 podemos montar sua struct. Conforme a documentação do módulo de Struct em Python, definiu-se da seguinte forma: `!6s6sH`. Primeiramente, o sinal `!` representa se tratar da ordem de bytes de redes, que utiliza o formato “Big Endian” conforme definido na IETF RFC 1700. A partir destes, o símbolo `6s` define o formato do primeiro campo, do endereço de ethernet do destinatário como sendo uma lista composta de 6 elementos do tipo ‘char’, que representam 1 byte - ou seja, uma lista de 6 bytes. Considerando que o segundo campo do protocolo ethernet representa o endereço mac do destinatário, utiliza-se o mesmo formato por razões óbvias. Por fim, o último campo significativo do cabeçalho é o valor do ether type, composto por dois bytes, utiliza o símbolo `H`, que representa um “unsigned short” - no caso, o menor tipo de dados composto por dois bytes definido pelo

módulo.

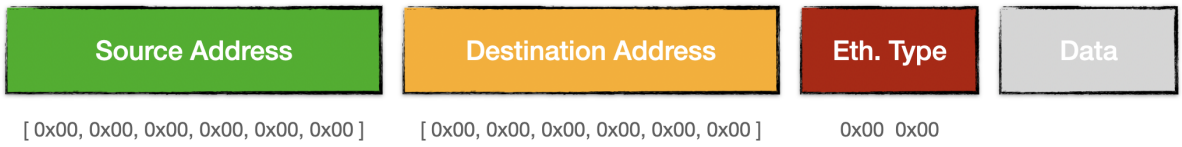

Figura 1. Modelo visual do header Ethernet

Para o protocolo de comunicação a ser desenvolvido optou-se pelo formato representado na figura 2 . O formato da struct, de acordo com a ilustração, fica como sendo `!b10s25s`. Como já descrito anteriormente o sinal `!` define este como um conjunto de bytes para redes. Com exceção do primeiro campo, que representa o tipo de mensagem enviada pelo protocolo representada por um um inteiro, os demais campos - nome e mensagem - se trata de strings. Desta maneira, são representados também por lista uma de chars, tendo seus valores encodados em bytes. Os valores `10s` e `25s` significam que os campos são representados por listas no tamanho de 10 e 25 itens, respectivamente.

Figura 2. Modelo visual do pacote do protocolo de comunicação desenvolvido

Tendo ambos pacotes bem definidos, torna-se possível o desenvolvimento de funções capazes de realizar o encapsulamento e desencapsulamento dos dados. Considerando a importância de manter ambos os processos de forma consistente, definiu-se um módulo capaz de realizar tais operações, capazes de receber os dados e realizar a conversão de maneira que os dados estejam em um formato adequado para o envio. Para o desencapsulamento dos dados da Struct, fez-se uso de sua compatibilidade com a estrutura de dados `named tuple` do Python, tornando mais simples a manipulação dos dados pelo programa, considerando que se pode obter os valores a partir dos campos nomeados.

3 Desenvolvimento Programa

A partir de um módulo capaz de realizar as operações necessárias para o envio de dados de maneira adequada, compreende-se a necessidade de desenvolvimento de uma classe capaz de exercer o comportamento adequado perante as solicitações apresentadas no enunciado do trabalho. Em linhas gerais, ele deve ser capaz de permitir a comunicação

entre os dispositivos através do envio e recebimento de mensagens através do protocolo desenvolvido.

Em consideração ao enunciado, foi desenvolvida a classe "Komi Protocol", com seu nome inspirado na série "Komi can't Communicate". A classe é a pedra angular da aplicação. Seu principal método é o de envio de pacotes via ethernet, através da abertura de uma `raw socket`. Em sua inicialização, é informado o nome da máquina, seu endereço mac e interface, pois tais dados devem fazer parte de toda a comunicação realizada por ela. Além deste, possui também um método para iniciar a comunicação da máquina. Este método, além de enviar a mensagem de START inicial, tem o objetivo de inicializar as threads que realizarão as rotinas assíncronas do programa.

Durante a operação do programa, este deve realizar de maneira periódica e assíncrona as seguintes rotinas: envio de uma mensagem do tipo HEARTBEAT via broadcast para a rede e atualização da lista de endereços conhecidos pelo host. Além destes, o programa fica constantemente realizando o sniff da rede, interceptando os pacotes recebidos cujo valor do `Ether Type` representa o protocolo Komi. A partir do desencapsulamento do pacote, se torna possível obter o tipo de mensagem característica do protocolo e realizar a operação esperada de acordo com esta.

Dentre os tipos de mensagens existentes no protocolo, de acordo com o enunciado, enumera-se estes da seguinte maneira:

0. START → A mensagem do tipo START é respondida com o envio de uma mensagem do tipo HEARTBEAT, tendo como o destinatário o endereço de origem da mensagem START enviada em broadcast.
1. HEARTBEAT → O comportamento esperado para uma mensagem deste tipo é a inclusão de um item na lista de endereços existente, a partir dos dados de endereço mac e nome do host de origem.
2. TALK → Utilizando-se do campo `message` do protocolo, o programa exibe a string recebida do remetente.

4 Lista de Endereços

A classe responsável pelo controle dos endereços acessados pelo programa, chamada `AddressBook`, guarda uma lista de objetos do tipo `AddressData`. Dentre os dados, além do nome do host e seu mac, durante a inicialização é gerado um timestamp deste momento. Tal timestamp é importante devido a regra de que, após 15 segundos, caso um

determinado host não tenha enviado sinal de vida via mensagem do tipo `HEARTBEAT`, ele deve ser removido da lista de endereços.

Sendo assim, seus métodos se resumem a inclusão de itens e atualização de sua lista. Para a inclusão, a fim de que não haja itens repetidos, os dados anteriores referente ao endereço mac recebido são removidos, para que uma nova entrada seja incluída na lista com o timestamp atualizado. Já o método de atualização dos endereços se trata da rotina descrita anteriormente, iniciada junto do Komi Protocol. Operando assincronamente, a cada 1 segundo ela percorre a lista removendo os itens cujo timestamp tenha passado dos 15 segundos estipulados.

5 Arquitetura e Funcionamento Básico

A partir dos elementos descritos, é possível a diagramação da arquitetura utilizada pelo programa desenvolvido. Na figura 3 podemos observar o modelo de arquitetura utilizada. O programa inicia a partir do módulo `main`, que fará a instanciação e inicialização do protocolo Komi a partir dos dados informados via argumentos. Entre eles, o nome do host e interface são obrigatórios e o valor de tempo em segundos para o heartbeat, opcional. O main também é responsável pelo prompt do usuário, onde ele pode interagir com o programa, solicitando a lista de endereços ou envio de mensagens para outros hosts informando o endereço mac.

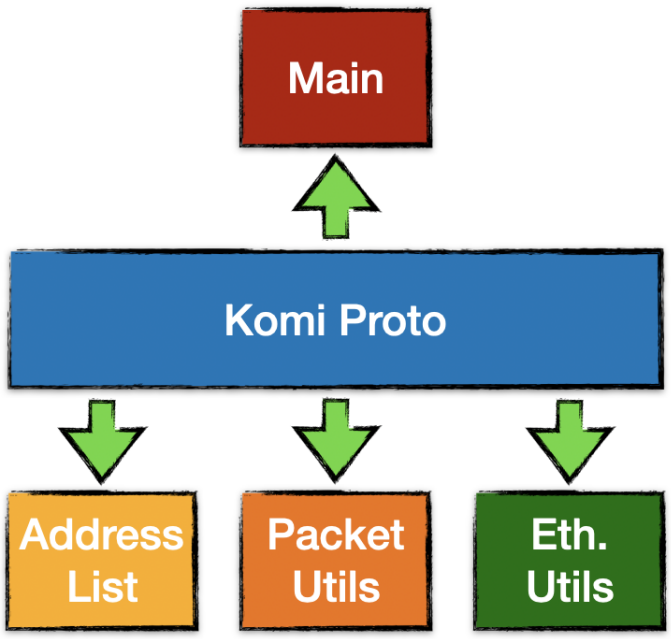

Figura 3. Arquitetura basica do projeto desenvolvido

Já a instância Komi Protocol faz dependência dos módulos de lista de endereços, `Packet Utils` e `Ethernet Utils`. Os dois primeiros, como descrito anteriormente, auxiliam, respectivamente, no controle de itens presentes na listas de endereços conhecidos do host e formação de pacotes a serem enviados na rede via raw socket. O módulo `Ethernet Utils`, não descrito em detalhes até o momento, se resume à criação de `raw sockets` e manipulação/parsing de dados relacionados a endereços mac.

6 Resultados Obtidos

A fim de verificar o funcionamento do programa desenvolvido, utilizando o programa Core Emulator, foi instanciada uma topologia contendo máquinas host interligadas através de um switch, conforme observa-se através da figura 4. A utilização da ferramenta core se mostra essencial para tal verificação pois suas funcionalidades incluem a utilização de terminais para os hosts bem como Wireshark para sniff da rede emulada.

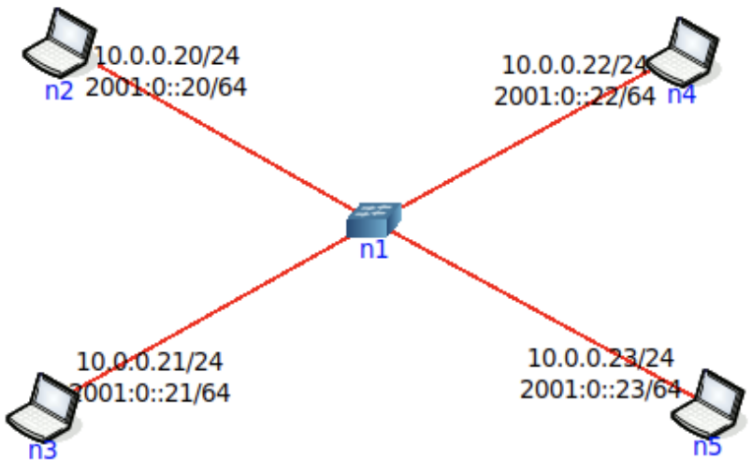

Figura 4. Topologia de testes utilizada no programa Core Emulator

A partir do acesso aos terminais referentes a cada uma das máquinas, foi inicializado o programa desenvolvido em cada uma delas. Através do Wireshark é possível acompanhar o tráfego dos pacotes pela rede. Conforme observa-se na figura 5 o programa apresenta o recebimento de diversos pacotes cujo `Ether Type` é o utilizado pelo protocolo desenvolvido, sendo estes as mensagens de START e HEARTBEAT enviadas automaticamente pelos hosts conectados que estão rodando o programa.

A partir da inicialização do programa, torna-se possível a verificação de recebimento das mensagens TALK entre os hosts. A partir do conhecimento do endereço mac de um host específico, é possível, através da opção `1` enviar uma mensagem de texto entre as

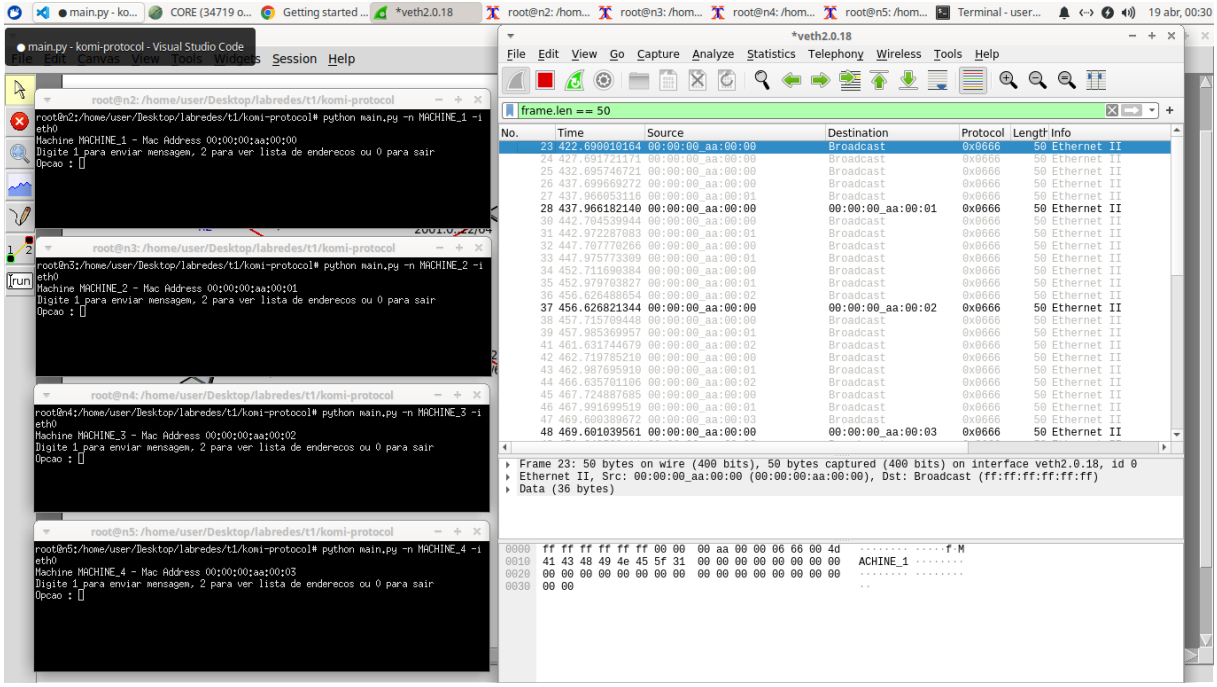

Figura 5. Sniff dos pacotes do host N2 via Wireshark

máquinas. Conforme demonstrado na figura 6, temos o host n2, denominado `MACHINE 1` enviando a mensagem para o destinatário `00:00:00:aa:00:01`, que recebeu o nome de `MACHINE 2`. No terminal do host n3, podemos confirmar o recebimento da mensagem completa. Conforme o texto impresso, podemos observar que o receptor foi capaz de realizar o parsing correto das informações, imprimindo o nome da máquina, seu endereço mac e também a mensagem. Na interface do Wireshark é possível analisar o pacote. Nos campos de source, destination e protocolo podemos observar que se encontram os valores corretos para o mac de origem e destino, bem como o código de protocolo utilizado. No campo dos bytes podemos confirmar também que os dados foram encriptados de maneira correta, apresentando o número correto do `Komi Type`, nome da máquina e sua mensagem.

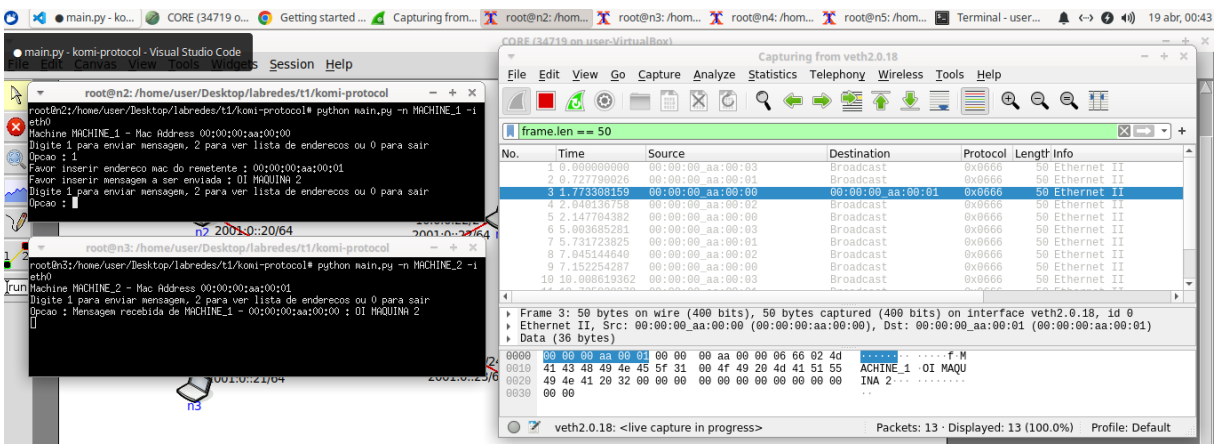

Figura 6. Demonstração de comunicação entre duas máquinas através do protocolo Komi

Da mesma maneira, podemos verificar que através do recebimento das mensagens do tipo HEARTBEAT já demonstradas na figura 5 que os hosts são capazes de incluir itens na sua lista de endereços. Conforme a figura 7, observa-se que cada um dos hosts possui os demais endereços ligados à rede em sua lista.

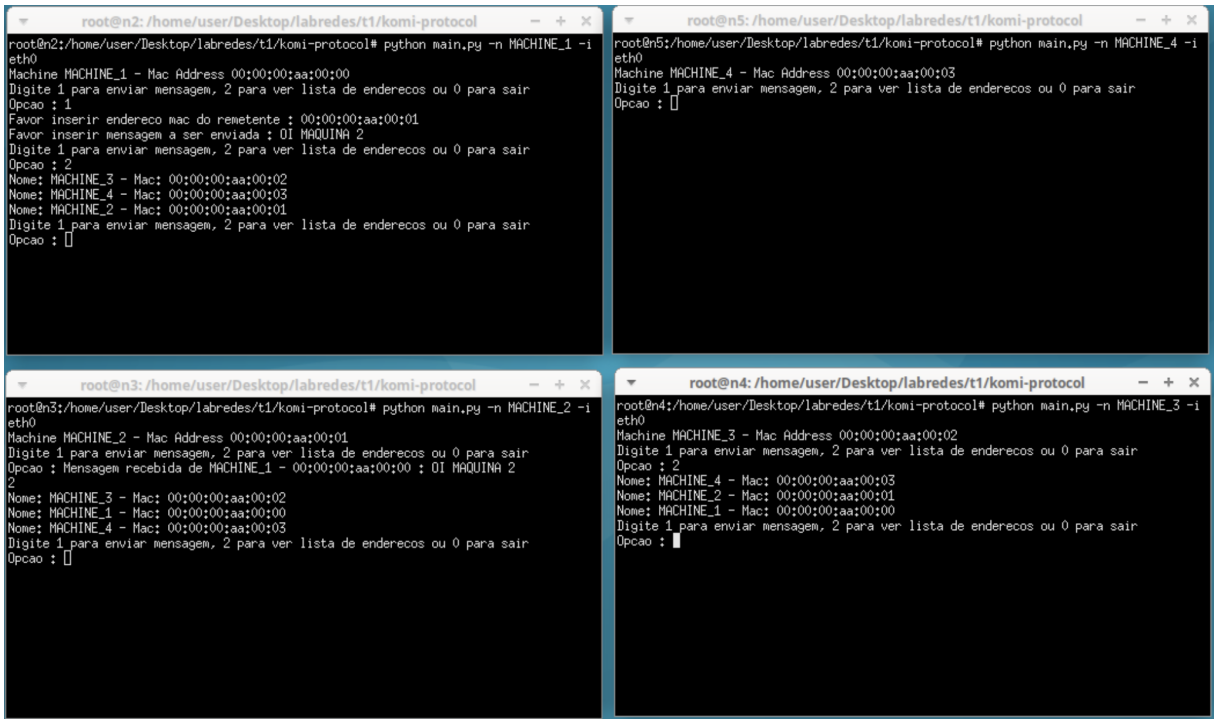

Figura 7. Lista de endereços de cada um dos hosts rodando o programa desenvolvido

No mesmo momento em que o host n5 (`MACHINE 4`) tem seu programa terminado, compreende-se que o processo de envio de mensagens do tipo HEARTBEAT será encerrado. Desta maneira, depois de até 15 segundos, conforme estipulado no enunciado do trabalho, ao solicitar a lista de endereços de cada um dos hosts, podemos notar que o host denominado `MACHINE 4` foi removido de todas as listas. Pode-se concluir, desta maneira, que o processo de atualização da lista de endereços é realizado corretamente. Os novos endereços apresentados sao demonstrados na figura 8, onde podemos notar que nenhum deles possui o item referente a este host.

7 Conclusão

O projeto desenvolvido apresentou resultados positivos em relação ao objetivo proposto pelo trabalho enunciado. Através do desenvolvimento de protocolos de comunicação através da rede foi possível compreender de forma clara o tráfego de pacotes. Embora desenvolvido em Python, conhecido por ser uma linguagem de alto nível, esta apresenta ferramentas adequadas para a manipulação de dados a nível de bytes. A utilização de

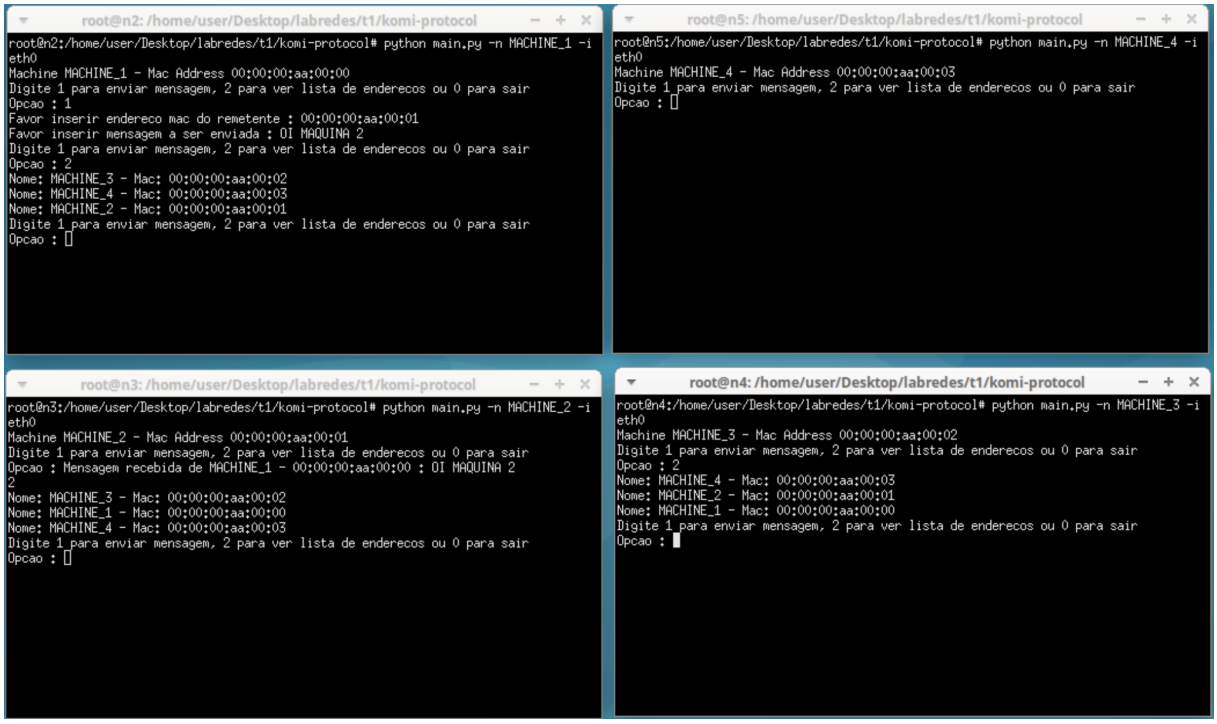

Figura 8. Lista de endereços atualizada, depois que o processo do host n5 foi encerrado

Structs e objetos do tipo Bytes tornou possível trabalhar com os dados tal qual estes são observados nos pacotes capturados pelo Wireshark.

Da mesma maneira, tal projeto possibilitou o desenvolvimento utilizando diversos elementos importantes para a computação. Além da manipulação de dados em baixo nível, foi necessário a utilização de encoding e decoding de valores, threads e sockets. Tais elementos foram muito importantes para o desenvolvimento de um trabalho completo, tendo todas suas especificações implementadas. Tais elementos foram muito importantes para a compreensão, não apenas do funcionamento do tráfego em uma rede, mas também de como é feito o tratamento dos dados em nível de software e a comunicação entre os diversos hosts ligados à rede.