

Avaliação de Redes: Camada de transporte

1st Allan de Lima da Silva
Ciências de Computação
Universidade do Vale do Itajaí (UNIVALI)
Itajaí, Brasil
allanlima@edu.univali.br

2nd João Vitor Specht Kogut
Engenharia de computação
Universidade do Vale do Itajaí (UNIVALI)
Itajaí, Brasil
Kogut@edu.univali.br

Resumo—O presente relatório tem como objetivo apresentar o desenvolvimento do projeto proposto pelo professor para disciplina de Redes, onde será consolidado o aprendizado sobre algoritmos usados para a construção da camada de transporte.

Palavras-chave—Redes de computador, projeto de rede, camada de transporte, TCP, Wireshark, Python

I. PROPOSTA

Projeto 1: Nesse projeto, como forma a estimular a integração dos conceitos da M1 e do conteúdo relativo à camada de transporte, você(s) irá(ão) desenvolver um cliente para chat de mensagens (podem chamar de “MRCI”). Para o trabalho, vocês deverão desenvolver essa aplicação que permita que pelo menos dois usuários conversem (troquem mensagens). Os dados a serem trocados podem se resumir a apenas texto (podem expandir para troca de imagens). O cliente poderá usar qualquer dos dois tipos de protocolos de transporte (UDP ou TCP), lembrando que vocês devem justificar suas escolhas, além disso, pode-se utilizar o protocolo QUIC (há o link para uma biblioteca Python em Materiais Extras).

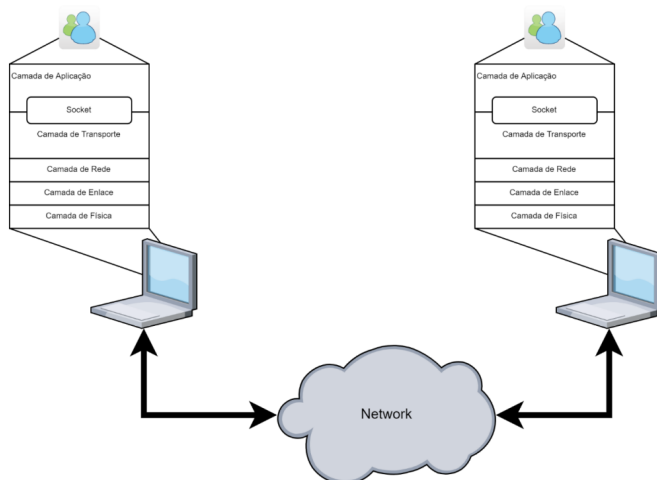


Fig. 1. Ilustração do enunciado

Requisitos para o desenvolvimento:

- Para a comunicação entre os dois clientes, pode usar:

Identify applicable funding agency here. If none, delete this.

localhost, Máquina Virtual (ex: VirtualBox), rede local (se possuir dois computadores na casa).

- Deve ser permitido identificar o usuário, além do uso da porta que é usada. Não há um requisito de porta, mas você deve justificar a escolha do número usado.
- Use um marcador de hora-minuto que a mensagem foi enviada (pode adicionar segundo).
- Ao receber uma mensagem, você deve exibir o nome do usuário e o momento em que a mensagem foi recebida.
- Os dois clientes devem conseguir enviar mensagens no momento que quiserem, não sendo necessário esperar uma iteração de um iniciador.

O uso do protocolo de transporte fica a critério do desenvolvedor, mas a escolha deve ser justificada. Podem usar variações de protocolo, passando parte do QoS (Quality of Service) para a camada de aplicação (ex: QUIC).

- Deve-se adicionar um dos dois aspectos (ou os dois) na implementação:

- Segurança – criptografia ou técnica similar
- Notificação de entrega e leitura

Projeto 2: Nessa segunda parte, você deve expandir o conceito para 3 clientes (2 clientes no mesmo host) e utilizar os recursos para que mais de um processo utilize a mesma porta. Para isso você pode usar um dos dois comandos:

`SO_REUSEPORT` ou `SO_REUSEADDR`
(ou equivalente na linguagem utilizada)

Fig. 2.

Pesquise sobre a utilização dos mesmos e como usar. Caso a linguagem/sistema operacional que você use não ofereça suporte, você pode dissertar os motivos que levaram a essa questão. Caso contrário, descreva os resultados obtidos com os experimentos.

Observação para os dois projetos

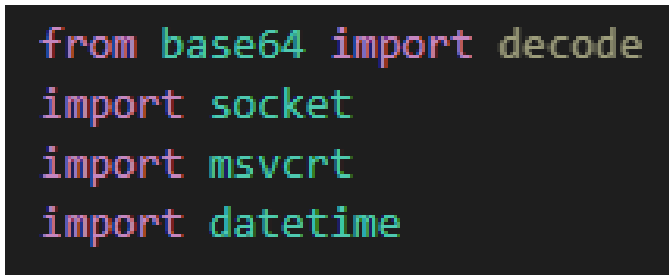
Em ambos os projetos, você deve usar o Wireshark para avaliar a comunicação dos seus clientes. Um roteiro e explicação sobre como usar o Wireshark para TCP/UDP pode ser encontrado na pasta Roteiros no material da disciplina.

II. DESENVOLVIMENTO

A. Explicação do código

O presente projeto foi desenvolvido no SO Windows com a linguagem Python (v3.10) na IDE VScode, dividido em dois arquivos, um servidor.py e um cliente.py, começaremos explicando o que é utilizado pelo usuário (cliente) que fará a conexão com o servidor.

O funcionamento do cliente é tido da seguinte forma: o cliente especifica seu nome de usuário, estabelece a conexão com o servidor, quando estabelecida, estará dentro do chat com todos os usuário também conectados, ele pode então receber mensagens enviadas por outros usuários, mandar mensagens, que são encriptografadas, não possibilitando que o servidor possa identificar seu conteúdo, apenas outros usuários, por fim o usuário também pode ser desconectar do chat pressionando a tecla ESC.

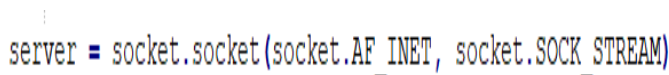


```
from base64 import decode
import socket
import msvcrt
import datetime
```

Fig. 3. Importações feitas no código do cliente

Na figura 4, é vista as bibliotecas utilizadas pelo cliente. A função "decode" da "base64" é utilizada para que sejam feitas as conversões das mensagens em bytes para strings e vice-versa. O import "Socket" permite que seja feita a utilização de funções relacionadas ao uso de sockets para que seja feita a conexão de rede entre o cliente e o usuário. O import "msvcrt" nos permite utilizar funções relacionadas ao uso do teclado do cliente, por fim é utilizado também "datetime" para que possamos ter o momento exato em que a mensagem foi digitada pelo cliente.

Na primeira linha é visto a função:



```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Fig. 4. Definição de parâmetros da conexão

que estabelece parâmetros muito importantes para o projeto, é definido que server é um objeto socket que recebe como parâmetro, "AFINET" para definir o uso do IPv4 e "SOCKSTREAM" para definir o uso do protocolo TCP.

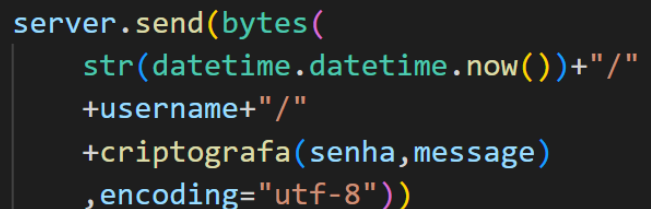
Considerando que o identificador da sala de chat é o IP do host do servidor, foi escolhido o padrão IPV4 para facilitar o acesso através da entrada manual. Seria possível

a implementação de uma look-up-table para correlacionar nomes de sala a endereços e portas, porém está fora do escopo deste projeto.

O protocolo TCP foi escolhido por conta de sua confiabilidade, dado que nativamente possui um sistema de confirmação de recepção de packets através de ACKS. Através de programas como o WireShark, é possível ver a troca de pacotes entre usuários e servidor, assim como suas mensagens de confirmação de recebimento, assegurando a chegada dos dados.

É definido também a senha que é utilizada para a criptografia das mensagens, o nome que o usuário utilizará e também o IP e Porta do servidor no qual o cliente se conectará. Com tudo isso, o cliente está apto a se conectar no servidor do chat, feito isso, é enviado ao servidor o nome de usuário definido pelo cliente com o uso da função send(), que envia dados (em bytes) de um socket para outro.

Dentro do chat de conversa, o usuário permanecerá em um looping, onde é verificado se foi recebida alguma mensagem pelo servidor, ou se o usuário está executando alguma atividade, podendo ser a ação de sair do chat apertando ESC, enviando então uma mensagem para que o servidor e outros clientes saibam que alguém saiu da conversa, ou a principal ação, sendo o envio de alguma mensagem. Assim que o usuário aperta a tecla ENTER, ele recebe um retorno visual de que está apto a digitar uma mensagem, e quando enviada é tratada da seguinte forma:



```
server.send(bytes(
    str(datetime.datetime.now())+"/"
    +username+"/"
    +criptografa(senha,message)
    ,encoding="utf-8"))
```

Fig. 5. Função utilizada para enviar as mensagens do cliente

É enviado para o servidor em bytes, 3 dados que são respectivamente: o horário no qual o usuário enviou a mensagem, o nome do usuário e por último, a mensagem digitada pelo mesmo, a mensagem é enviada de maneira criptografada para o servidor.

A criptografia foi implementada com o objetivo de ser simples e funcional, apenas para quesitos de demonstração. A lógica por trás de seu funcionamento é o da multiplicação do valor decimal que representa o caractere na tabela ASCII pela senha escolhida pelo usuário. Assim, é transformado cada caractere em um valor inteiro, que pode ser convertido de volta ao original caso o outro usuário possua a mesma senha.

Toda vez que o usuário recebe a mensagem, é feita a separação dos dados recebidos, o horário é tratado para melhor visualização e a mensagem é descriptografada, precisando possuir a mesma senha utilizada pelo usuário que enviou a mensagem.

Para o código do servidor, é visto os seguintes imports, a função “exception” para podermos fazer o debug de algum eventual problema, o import “select” é utilizado para definir as possíveis origens dos pacotes e novamente o import “socket” visto também no cliente.

É criado uma definição do socket do servidor com as seguintes atribuições vistas na figura abaixo.

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((IP, PORT))
server_socket.listen(100)
sockets_list = [server_socket]
clients = {}
```

Fig. 6. Parâmetros estabelecidos no servidor

Na figura 6 pode-se ver que são definidos os parâmetros para uso do IPv4 e TCP, assim como o endereço IP (utilizado localhost) e a porta, onde utilizamos 666, pois está no range das well-known ports, reservadas para aplicações TCP/IP, e por não possuir outra aplicação a utilizando (GORALSKI, 2017).

É então inicializado o processo de “escuta” na porta definida, neste caso chamada de listen é inicializada com o valor 100, indicando o número máximo de conexões ativas. Em sequência, são inicializadas as variáveis para armazenamento de sockets e clientes.

B. Resultados

```
mp\Redes 1\M3> py .\client.py
Entre com seu usuário:
> allan
Pressione ENTER para digitar ou ESC para sair
(22:46:08) EU > Oi!
```

Fig. 7. Exemplo de um cliente enviando uma mensagem

Na figura vista acima, vemos o terminal do cliente, onde ele foi identificado e enviou uma mensagem para o servidor, e logo na figura abaixo é visto o terminal do servidor, onde pode-se observar a criptografia funcionando.

No terminal do servidor, é identificado o novo usuário e seu endereço de IP e a porta pela qual ele está se comunicando, e

```
Conexao de 127.0.0.1:64606, usuario: allan
['2022-10-23 22:46:14.751961', 'allan', '711-945-297-']
(22:46:14) allan: 711-945-297-
```

Fig. 8. Visualização da mensagem pelo servidor

é visto a mensagem enviada pelo mesmo, onde é apresentada de duas maneiras, sua forma sem tratamento e logo depois a forma tratada pela qual se assimila a forma vista pelo usuário que receberá essa mensagem.

Para que seja melhor visualizada a comunicação entre cliente e servidor, foi realizada uma simulação entre dois computadores em mesma rede, possuindo então diferentes endereços de IP, para analisarmos essa comunicação foi utilizado o Wireshark onde podemos observar cada pacote enviado e recebido.

```
PS C:\Users\allan\Desktop\teste> py .\client.py
Entre com seu usuário:
> allan
Pressione ENTER para digitar ou ESC para sair

(21:44:01) EU > ola
(21:44:05) kogut: OLA!
(21:44:08) EU > trabalho show!
(21:44:14) kogut: Realmente
```

Fig. 9. Conversa entre dois usuários

Na figura 9 vemos algumas mensagens trocadas pelos 2 clientes, logo na figura 10 vemos os pacotes resultantes, onde é possível observar o funcionamento da troca de pacotes TCP.

Nas figuras 11 e 12 é possível ver a estrutura dos dados contidos no pacote, constituída pelo horário de envio, nome de usuário e a mensagem encriptada. No campo superior é possível ver a inversão dos endereços source e destination, confirmando a comunicação com o servidor.

REFERENCES

- [1] Repositorio no GitHub com o projeto desenvolvido. Disponível em: <https://github.com/AllanLimaS/RedesM3-chat-em-rede-Python>.
- [2] GORALSKI, W. Registered Port - an overview — ScienceDirect Topics. Disponível em: <https://www.sciencedirect.com/topics/computer-science/registered-port>.

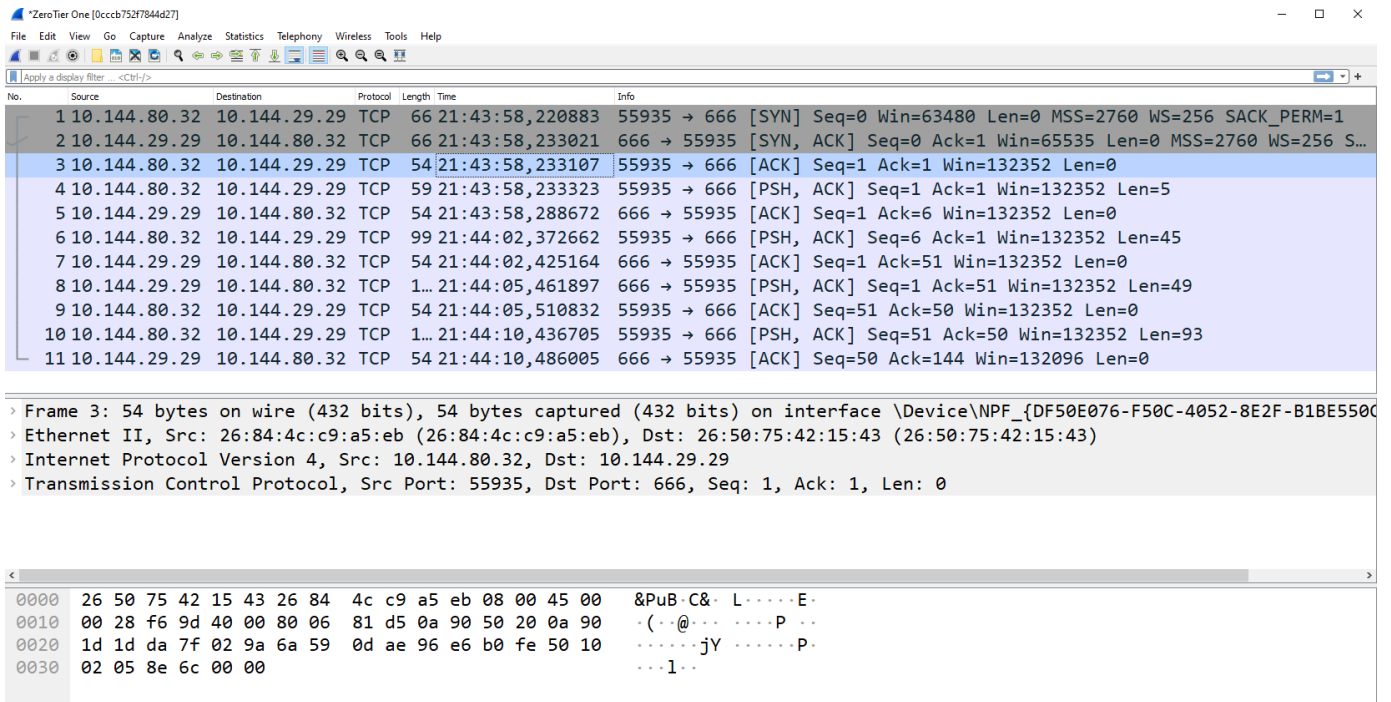


Fig. 10. Pacotes referentes as mensagens no Wireshark

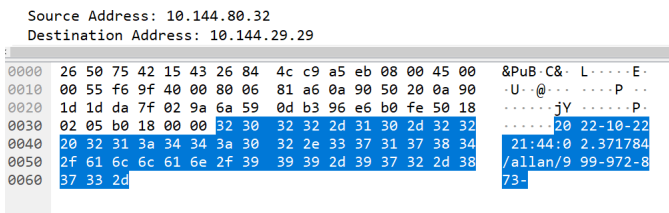


Fig. 11. Estrutura do pacote da mensagem de um cliente

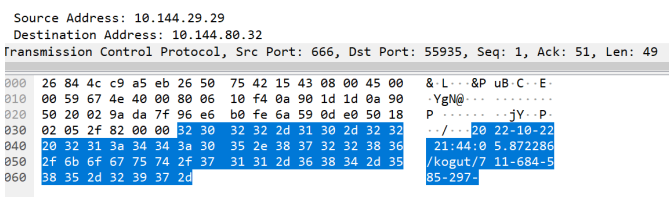


Fig. 12. Estrutura do pacote da mensagem de outro cliente