

Programa de Calculo de Rede

Alisson da Silva Vieira, RA: 2046229
Caio Eduardo Theodoro, RA.: 2044560

6 de dezembro de 2019

1 Introdução

Neste trabalho, será documentado e detalhado os processos do desenvolvimento do software de realização de cálculos básicos utilizados durante a implementação da atividade prevista na disciplina de redes de computadores 1, bem como especificado na Parte II da Atividade.

2 Breve descrição:

2.1 A linguagem de programação utilizada:

A Linguagem utilizada para realização do programa foi o Python. A linguagem é de alto nível, possui bibliotecas de tratamento de arquivos JSON, e foi recomendada pelo Professor.

2.2 Tecnologia JSON

O JSON é um formato de troca de dados entre sistemas independente de linguagem de programação derivado do JavaScript. A partir de 2017 muitas linguagens de programação incluíram código para gerar, analisar sintaticamente dados em formato JSON e também converter para objetos da linguagem. Uma dessas, o Python, nossa linguagem escolhida.

2.3 Endereçamento IP

Dentro de uma rede TCP/IP, cada micro recebe um endereço IP único que o identifica na rede. Ele é dividido em duas partes: A primeira identifica a rede à qual o computador está conectado A Segunda identifica o host dentro da rede. Para melhorar o aproveitamento dos endereços disponíveis, os desenvolvedores do TPC/IP dividiram o endereçamento IP classes, e cada uma

dessas classes reserva um número diferente de octetos para o endereçamento da rede.

3 Apresentação do código

3.1 Tratamento do arquivo JSON:

Para o gerenciamento do arquivo JSON, foi usado o módulo *import json*, nele contendo o *json.dump(a,b)*. Com o caracter a sendo o objeto de dados a ser serializado e b o objeto de arquivo no qual os bytes serão gravados. Como visto varias vezes no código, é basicamente o comando utilizado para converter o python em uma string JSON.

```
arquivo = open("saida.json", "w")
json.dump(a, arquivo)
```

Figura 1 – Exemplo de uso do json.dump()

Neste exemplo, a variavel arquivo simboliza o arquivo saida.json em questão, e o json.dump diz que a variavel a será escrita na varivel arquivo.

3.2 Verificar se o IP e a Máscara são validos:

Para verificar se o IP é valido, fizemos uma série de comparações com faixas reservadas. Utilizamos uma tabela cujo encontramos na internet, e para cada uma das situações fizemos uma condição (if) no codigo. Essas situações são representadas na imagem abaixo:

Blocos de Endereços Reservados		
CIDR Bloco de Endereços	Descrição	Referência
0.0.0.0/8	Rede corrente (só funciona como endereço de origem)	RFC 1700
10.0.0.0/8	Rede Privada	RFC 1918
14.0.0.0/8	Rede Pública	RFC 1700
39.0.0.0/8	Reservado	RFC 1797
127.0.0.0/8	Localhost	RFC 3330
128.0.0.0/16	Reservado (IANA)	RFC 3330
169.254.0.0/16	Zeroconf	RFC 3927
172.16.0.0/12	Rede privada	RFC 1918
191.255.0.0/16	Reservado (IANA)	RFC 3330
192.0.2.0/24	Documentação	RFC 3330
192.88.99.0/24	IPv6 para IPv4	RFC 3068
192.168.0.0/16	Rede Privada	RFC 1918
198.18.0.0/15	Teste de benchmark de redes	RFC 2544
223.255.255.0/24	Reservado	RFC 3330
224.0.0.0/4	Multicasts (antiga rede Classe D)	RFC 3171
240.0.0.0/4	Reservado (antiga rede Classe E)	RFC 1700
255.255.255.255	Broadcast	

Figura 2 – Tipos de ip.

Como exemplo de código, iremos utilizar o segundo caso, onde o 1º octeto tem o valor de 10 e os 3 octetos restantes estão setados em 0. Neste caso fazemos a seguinte comparação:

```

83     # verifico se o endereço é privado
84     if ((int(enderecoB[0]) == 10) and (int(enderecoB[1]) == 0) and (int(enderecoB[2]) == 0) and (int(enderecoB[3]) == 0)):
85         return "Rede privada"

```

Figura 3 – Rede privada.

Para verificar se a Máscara é válida, comparamos sua compatibilidade com a classe. Se a classe do ip for tipo A, quer dizer que a máscara obrigatoriamente deverá possuir no 1º octeto o valor 255, caso a classe for tipo B, a máscara deverá possuir o valor 255 no 2 primeiros octetos, por fim, caso seja classe tipo C, a máscara deverá ter os 3 primeiros octetos com o valor 255. Então, após verificado isso, verificamos a veracidade da máscara, passamos por toda ela e verificamos caso tenha algum 0 seguido de 1, caso tiver, a máscara é falsa.

```

if (classe == "A"):
    if (endereco[0] == 255):
        for i in endereco2[1:]:
            for j in i:
                if ((j == "1") and (comecou_zero == False) and (comecou_um == False)):
                    começou_um = True
                elif ((j == "0") and (comecou_zero == False) and (comecou_um == False)):
                    não_pode_ter_um = True
                    começou_zero = True
                elif ((j == "1") and (não_pode_ter_um == True)):
                    return False
                elif ((j == "0") and (comecou_um == True)):
                    não_pode_ter_um = True
            return True

```

Figura 4 – Validação de máscara.

Essa verificação se dá, pois uma máscara não deve possuir um 0 seguido de um 1, por exemplo: 255.255.255.1, é uma máscara inválida. Já 255.255.255.248 é um exemplo de máscara válida, e garantimos que nosso algoritmo faça a verificação da maneira correta, pois ele verifica bit a bit, e caso ache um 0, a variável não pode ter um é setada como verdadeiro, então caso ache um 1 e a variável em questão é verdadeira, ele retorna falso.

3.3 netID hostID da máscara:

Primeiramente, para se obter o netId usando o endereço da máscara, bastou-se contar a quantidade de bits que estão setados em 1's contida na máscara. Usando como exemplo o seguinte endereço de máscara: 255.255.0.0.

Contando os bits 1 deste endereço de máscara, temos o valor 16. Logo 16 é o valor do netId da máscara. E para se obter o hostId da máscara, bastou-se contar a quantidade de 0's contidas nela. Logo contando os bits 0 deste endereço de máscara, temos o valor 16. Logo 16 é o valor do hostId da máscara.

Vemos como foi implementado isso na Figura 5, logo abaixo.

```
count_um = 0
count_zero = 0
for i in endereco:
    for j in i:
        if (j == "1"):
            count_um = count_um + 1
        elif (j == "0"):
            count_zero = count_zero + 1

if (flag == 0):
    return count_um
elif (flag == 1):
    return count_zero
```

Figura 5 – Funcao que calcula o Host ID e o Net ID.

3.4 Classe do IP:

Na verificação da classe do IP, é feita uma comparação entre faixas. Elas então, determinarão qual será a classe do meu IP, dependendo da faixa em que está acolado. Para calcular se o IP é de classe A, verificamos se o valor do primeiro octal está na faixa de 0 a 127. Já para a classe B, é necessário verificar se o mesmo esta entre 128 e 192. E por ultimo, para a classe C, verificamos caso ele se encontra entre 192 e 223. A figura 6 logo abaixo mostra com mais detalhes como foi implementado.

```
if ((endereco[0] <= 126) and (endereco[0] > 0)):
    A = True
    if(((endereco[1] == 0) and (endereco[2] == 0) and (endereco[3] == 0)) or ((endereco[1] == 255) and
    (endereco[2] == 255) and (endereco[3] == 255))):
        errA = True

if ((endereco[0] <= 191) and (endereco[0] > 127)):
    B = True
    if(((endereco[2] == 255) and (endereco[3] == 255)) or ((endereco[2] == 0) and (endereco[3] == 0))):
        errB = True

if ((endereco[0] <= 223) and (endereco[0] > 191)):
    C = True
    if((endereco[3] < 1) and (endereco[3] > 254)):
        errC = True
```

Figura 6 – Classe do IP.

Também verificamos caso o IP seja inválido. Usando como exemplo este ip: 120.255.255.255, ele entrará no primeiro if, porém dentro dele terá uma outra verificação, onde caso o endereço possua os 3 octetos restantes em 0 ou em 255, sabemos que é inválido, e a variável errA recebe verdadeiro, ou seja, o endereço aparenta ser da classe A, porém está errado.

3.5 IP da Rede e Broadcast:

Para calcular o Ip da Rede e do Broadcast, primeiramente precisamos achar a posição da ultima aparição do número um. Para isso, percorremos a máscara de rede contando o número de 1's e 0's. Desta maneira, como mostrado na figura 7, sempre teremos a posição do ultimo 1.

```
# acho a posicao do ultimo numero 1
for i in netMask:
    for j in i:
        count_pos = count_pos + 1
        if (j == "1"):
            count_um = count_pos
        elif (j == "0"):
            count_zero = count_zero + 1
```

Figura 7 – Posição do último 1

Tendo esta posição em mãos, agora, verificamos em que posição se encontra o último 1. O nosso endereço é dividido em 4 octetos, e a posição do número 1 pode estar em qualquer uma delas. Tendo em vista isso, fazemos 4 comparações, como mostrado na figura 8.

```
if (count_um >= 0 and count_um < 9):
elif (count_um >= 9 and count_um < 17):
elif (count_um >= 17 and count_um < 25):
elif (count_um >= 25 and count_um < 33):
```

Figura 8 – Comparações

Caso eu possua a seguinte máscara: 255.0.0.0, a posição do último 1 será a 8, logo ele entrará no primeiro if. Agora então, fazemos uma cópia da endereço ip até a posição desse ultimo 1, e então, setamos tudo depois dela para zero. Feito isso, garantimos que temos a cópia até a posição do último 1, porém, também precisamos cobrir o caso do ultimo 1 estar no meio do endereço, no nosso caso atual, não passamos por isso, pois "não sobra resto", ou seja, tenho os 8 bits do octeto setados em 1. Para isso, temos o segundo for, que cobre essa "sobra" caso houver, setando tudo em 0 para o endereço de rede, e 1 para o endereço de broadcast.

```

if (flag == 0):
    ip_rede = []
    for i in range(count_um):
        ip_rede.append(ipAddr[0][i])
    for j in range(8 - count_um):
        ip_rede.append("0")
    for i in range(3):
        for j in range(8):
            ip_rede.append("0")
    ip_rede = st.formatar(ip_rede)
    return ip_rede
elif (flag == 1):
    ip_broadcast = []
    for i in range(count_um):
        ip_broadcast.append(ipAddr[0][i])
    for j in range(8 - count_um):
        ip_broadcast.append("1")
    for i in range(3):
        for j in range(8):
            ip_broadcast.append("1")
    ip_broadcast = st.formatar(ip_broadcast)
    return ip_broadcast

```

Figura 9 – Finalização

E por último, setamos todo o resto do endereço em 1 para o ip de rede, e em 0 para o ip de broadcast. A lógica continua a mesma para os demais if's, apenas mudando um pouco a lógica. Quanto mais perto do 4 octeto o meu ultimo 1 estiver, maior vai ser minha área de copia do endereço ip, e menor vai ser minha área para setar tudo em 0 ou 1.

3.6 Quantidade de Hosts e Faixa de máquinas válidas:

Para calcular a quantidade de hosts nos usamos o endereço da Máscara. Contamos a quantidade de 0's, com esse valor em mãos, utilizamos ele na seguinte formula: 2^n .

```

# calcula a quantidade de hosts na rede
def quantidadehosts(endereco):
    count_zero = 0

    for i in endereco:
        for j in i:
            if (j == "0"):
                count_zero = count_zero + 1

    qtd_hosts = (2 ** count_zero)

```

Figura 10 – Quantidade de hosts

Usando como exemplo o seguinte endereço de máscara: 255.255.255.248,

este endereço em binário é: 11111111.11111111.11111111.11111000. Logo, contando a quantidade de 0's que este endereço tem, chegamos a $n = 3$. Por fim temos o resultado, que é 8 hosts, pois colocando na fórmula fica: $2^3 = 8$.

A implementação é vista com mais detalhes na Figura 10 logo acima. Derivando disso, chegamos na quantidade de faixas válidas. Nesta questão, pegamos tanto o ip de rede e o ip de broadcast para nos ajudar, a implementação e vista na figura 7 logo abaixo.

```
# calcula a faixa de enderecos validos
def faixa(endereco, flag):
    p1 = int(endereco.split(".")[0])
    p2 = int(endereco.split(".")[1])
    p3 = int(endereco.split(".")[2])
    p4 = int(endereco.split(".")[3])

    p1 = st.converte(p1)
    p2 = st.converte(p2)
    p3 = st.converte(p3)
    p4 = st.converte(p4)

    if (flag == 0):
        p5 = str(p1) + "." + str(p2) + "." + str(p3) + "." + str(p4 + 1)
        return p5
    elif (flag == 1):
        p5 = str(p1) + "." + str(p2) + "." + str(p3) + "." + str(p4 - 1)
        return p5
    p5 = str(p1) + "." + str(p2) + "." + str(p3) + "." + str(p4)
    return p5
```

Figura 11 – Faixas de ip validos

Nesta função, dividimos o endereço em 4 partes, e como o endereço que chega pela função está em binário, convertemos para decimal. Logo após retornamos dependendo da flag que vier (0 para o ip válido inicial, e 1 para o ip válido final), retornamos uma string contendo as 4 partes em decimal. Porém, para o ip válido inicial, acrescentamos uma unidade em seu valor final, e para o ip válido final, decrementamos um em seu valor final. É importante ressaltar também, que usamos esta função para conversão e formatação de strings, de binário para decimal, caso a flag seja diferente das opções acima!

4 Execução do programa:

Primeiramente, é necessário ter o Python3 instalado em sua máquina. Depois de instalado, basta abrir a pasta do arquivo descompactado no Terminal e digitar o seguinte comando:

```
python3 main.py config.json
```

```
\APS-Redes-1> python3 main.py config.json
```

Figura 12 – Comando terminal

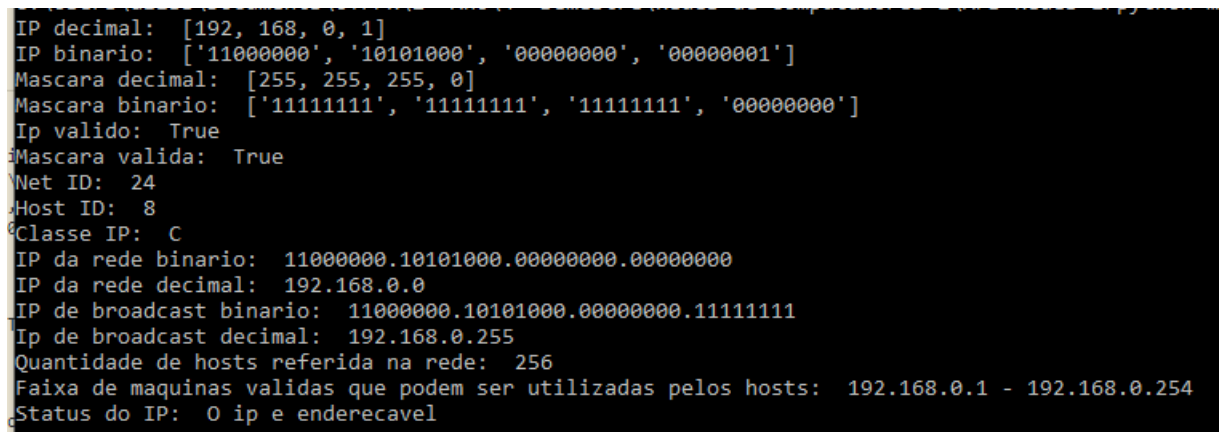
Feito isso, surgirá um Arquivo "saida.JSON" como segue o exemplo:



```
() saida.json X
() saida.json > ...
1
2     "ip_valido": true,
3     "mascara_valida": true,
4     "net_id": 24,
5     "host_id": 8,
6     "classe_ip": "C",
7     "ip_da_rede": "192.168.0.0",
8     "ip_broadcast": "192.168.0.255",
9     "quantidade_de_hosts": 256,
10    "ip_valido_inicial": "192.168.0.1",
11    "ip_valido_final": "192.168.0.254",
12    "ipStatus": "O ip e enderecavel"
13
```

Figura 13 – Arquivo: saida.json

Além disso, os dados também são mostrados no terminal após a execução:



```
IP decimal: [192, 168, 0, 1]
IP binario: ['11000000', '10101000', '00000000', '00000001']
Mascara decimal: [255, 255, 255, 0]
Mascara binario: ['11111111', '11111111', '11111111', '00000000']
Ip valido: True
Mascara valida: True
Net ID: 24
Host ID: 8
Classe IP: C
IP da rede binario: 11000000.10101000.00000000.00000000
IP da rede decimal: 192.168.0.0
IP de broadcast binario: 11000000.10101000.00000000.11111111
Ip de broadcast decimal: 192.168.0.255
Quantidade de hosts referida na rede: 256
Faixa de maquinas validas que podem ser utilizadas pelos hosts: 192.168.0.1 - 192.168.0.254
Status do IP: O ip e enderecavel
```

Figura 14 – Saída no terminal

E para testar o programa com diferentes endereços de ip e de máscara, basta mudar o arquivo config.json.

5 Conclusão

Na disciplina de Redes de Computadores 1, torna-se fundamental o entendimento desses conceitos básicos de aplicação, tanto por funcionalidade, quanto por usos futuros. A partir dele podemos, por fim, entender os componentes básicos presentes em toda a rede de computador, e assim possibilitando o bom uso da mesma, presente em casa dia de nossas futuras profissões. Com

este trabalho, conseguimos descobrir sentidos e funções de coisas já habituais em nosso processo de desenvolvimento, e agora, podemos otimizá-los.

Referências

https://www.cisco.com/c/pt_br/support/docs/ip/routing-information-protocol-rip/13790-8.html

<https://www.hardware.com.br/livros/linux-redes/capitulo-entendendo-enderecamento.html>

<https://www.profissionaisti.com.br/2012/09/redes-de-computadores-entendendo-a-camada-de-rede/>

<https://www.devmedia.com.br/json-tutorial/25275>

https://pt.wikipedia.org/wiki/Endereço_IP