

RELATORIO TÉCNICO

Redes de Computadores - PPComp 2021/1

AUTOMATIZAÇÃO DE TESTE DE REDE UTILIZANDO RYU E MININET

Vinicius Wilson Cardoso Silva

29/08/2021

Objetivos propostos

O Referido trabalho foi elaborado no período de 05/07 a 05/09, objetivo geral do trabalho é aplicar os conceitos de Redes Definidas por Software para implementar um ambiente de capaz de realizar teste de vazão da rede do tipo “todos para todos” e medir o tempo necessário para completar a tarefa (Flow Completion Time), utilizando uma plataforma de testes denominada MININET. Foram utilizadas diversas bibliotecas e funções como por exemplo o NetworkX, IPERF, Bwm-ng , Gnuplot para realização dos teste, armazenamento e plotagem dos resultados

Topologia Genérica e FatTree

Conforme proposto foram utilizados os 2 métodos para criar a topologia

FATTREE: A topologia tem como objetivo obter altos níveis de banda passante para vários dispositivos interconectados com switches menores, gerando uma lista de arestas para conexão posterior

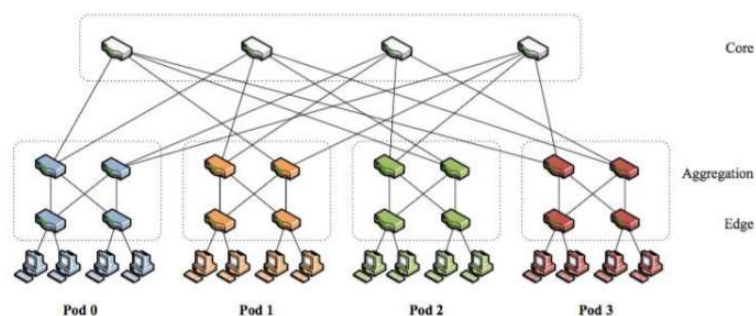


Figura 1 - Topologia Fattree

```
#-----
#Definindo a Classe responsavel pela criancao da topologia fattree
class FatTree( Topo ):
    def build( self, n=2 ):
        k = int(n)
        r=1
        # Create core nodes
        n_core = int(((k // 2) ** 2) // r)
        c = [] # core
        a = [] # aggravate
        e = [] # edge
        s = [] # switch

        for i in range(n_core):
            sw = self.addSwitch('c{}'.format(i + 1))
            c.append(sw)
            s.append(sw)

        # Create aggregation and edge nodes and connect them
        for pod in range(k):
            aggr_start_node = len(s) + 1
            aggr_end_node = aggr_start_node + k // 2
            edge_start_node = aggr_end_node
            edge_end_node = edge_start_node + k // 2
            aggr_nodes = range(aggr_start_node, aggr_end_node)
            edge_nodes = range(edge_start_node, edge_end_node)
            for i in aggr_nodes:
                sw = self.addSwitch('a{}'.format(i))
                a.append(sw)
                s.append(sw)
```

```

for j in edge_nodes:
    sw = self.addSwitch('e{}'.format(j))
    e.append(sw)
    s.append(sw)
for aa in aggr_nodes:
    for ee in edge_nodes:
        self.addLink(s[aa - 1], s[ee - 1])
        #-----
        #Criar a lista de arestas da topologia em arquivo
        art = ('(' + s[aa - 1] + "." + s[ee - 1] + ')')
        art = art.replace('a', "").replace('b', "").replace('c', "").replace('e', "")
        DbArestas.append(art)
        #-----
# Connect core switches to aggregation switches
for core_node in range(n_core):
    for pod in range(k):
        aggr_node = n_core + (core_node // ((k // 2) // r)) + (k * pod)
        self.addLink(s[core_node], s[aggr_node])
        #-----
        #Criar a lista de arestas da topologia em arquivo
        art = str('(' + s[core_node] + "." + s[aggr_node] + ')')
        art = art.replace('a', "").replace('b', "").replace('c', "").replace('e', "")
        DbArestas.append(art)
        #-----
# Create hosts and connect them to edge switches
count = 1
for sw in e:
    for i in range(int(k / 2)):
        host = self.addHost('h{}'.format(count))
        self.addLink(sw, host)
        count += 1

stringcount = len(DbArestas)
print('numero de arestas Encontradas no arquivo =', stringcount)
print('As Arestas Encontradas foram = ', DbArestas)
#-----

```

GENÉRICA A Lista de arestas geradas via biblioteca NetworkX dada um número de nodes e arestas **fazendo a validação se todos os nós estão realmente conectados** gerando um arquivo topo1.txt a ser lido posteriormente para criação topologia no MININET

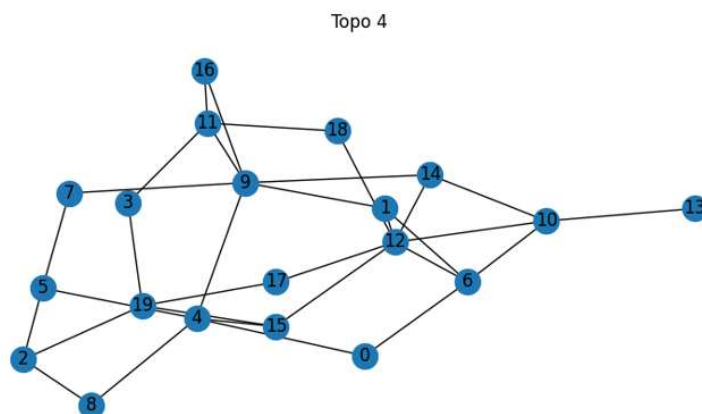


Figura 2 - Exemplo de topologia Genérica

```

class Generic( Topo ):
def build( self, n=20 ):
    #Cria os hosts / switches e conecta cada host a 1 switch
    for h in range(n):
        host = self.addHost( 'h%s' % (h + 1))
        switch = self.addSwitch('s%s' % (h))
        self.addLink( host, switch)

    # Rotina que cria um arquivo topo1.txt com uma topologia aleatoria
    #-----
    n = 20 # 16 nodes
    m = 32 # 28 edges
    arestas=[]
    G = nx.gnm_random_graph(n, m)

    # Verifica se realmente todos os nos estao conectados caso contrario gera a proxima
    while not nx.is_connected(G):
        G = G = nx.gnm_random_graph(n, m)

    print('criando arquivo de arestas no meu formato')
    h=1
    for line in nx.generate_adjlist(G):
        line = line.split(" ")
        stringcount = len(line)
        if (stringcount>1):
            for t in range(stringcount-1):
                test = ('(' + line[0] + '.' + line[t+1] + ')')
                arestas.append(test)

    print(arestas)
    #Grava no arquivo topo1.txt
    with open('topo1.txt', 'w') as arquivo:
        arquivo.write(str(arestas))
        print('Arquivo Gravado com sucesso')
    #nx.draw(G)
    #plt.show()

    #Rotina para ler os dados do arquivo de topologia "topo1.txt" inserido via linha de comando
    #-----
    f = format(args.file)
    Arquivo = None
    Lista_arestas=[]
    Arquivo=open(f, 'r')

    for line in Arquivo:
        line = line.split(",")

    stringcount = len(line)
    print('numero de arestas Encontradas no arquivo =' ,stringcount)
    print(line)
    j=0
    #Faz as conexoes de arestas lidas no arquivo
    for j in range(stringcount):
        Lista_arestas.append(line[j])
        Conexionsline = Lista_arestas[j].split(".")
        sEswitch = Conexionsline[0].replace("(", "")
        sDswitch = Conexionsline[1].replace(")", "")
        self.addLink('s%s' % int(sEswitch), 's%s' % int(sDswitch))
    #-----

```

Montagem da topologia – Via Arquivo

Para montagem da topologia em primeiro momento foi idealizado a criação da topologia diretamente no MININET, e posteriormente seria importado todas as informações para o RYU (realizando a leitura). Para isso foi utilizado uma biblioteca chamada Pickle, que demonstrou ser bem útil e ágil no armazenamento, o arquivo DB foi criado e armazenados os dados de Nome, PID, Arestas das conexões (CONEX) de todos os switches. Já para os hosts foram armazenados Nome, IP, MAC e a qual switch este host estava conectado. (disponibilizado código como anexo, com o nome “Teste_topologia.py”)

```
mininet@mininet-vm:~/mininet/examples$ sudo python teste_topologia.py --topologia fattree -k 4 --r ospf
-----
Foi Gravado o arquivo de configuracoes com o metodo roteamento informado pelo usuario
A Topologia Seleccionada foi Fat-Tree
O Numero de Portas e = 4
Unable to contact the remote controller at 127.0.0.1:6633
ID 0000000000000001 c1 IP : MAC : Conex : (1.5.0) (1.9.1) (1.13.2) (1.17.3)
ID 0000000000000002 c2 IP : MAC : Conex : (2.5.0) (2.9.1) (2.13.2) (2.17.3)
ID 0000000000000003 c3 IP : MAC : Conex : (3.6.0) (3.10.1) (3.14.2) (3.18.3)
ID 0000000000000004 c4 IP : MAC : Conex : (4.6.0) (4.10.1) (4.14.2) (4.18.3)
ID 0000000000000005 a5 IP : MAC : Conex : (5.7.0) (5.8.1) (5.1.2) (5.2.3)
ID 0000000000000006 a6 IP : MAC : Conex : (6.7.0) (6.8.1) (6.3.2) (6.4.3)
ID 0000000000000007 e7 IP : MAC : Conex : (7.5.0) (7.6.1)
ID 0000000000000008 e8 IP : MAC : Conex : (8.5.0) (8.6.1)
ID 0000000000000009 a9 IP : MAC : Conex : (9.11.0) (9.12.1) (9.1.2) (9.2.3)
ID 0000000000000010 a10 IP : MAC : Conex : (10.11.0) (10.12.1) (10.3.2) (10.4.3)
ID 0000000000000011 e11 IP : MAC : Conex : (11.9.0) (11.10.1)
ID 0000000000000012 e12 IP : MAC : Conex : (12.9.0) (12.10.1)
ID 0000000000000013 a13 IP : MAC : Conex : (13.15.0) (13.16.1) (13.1.2) (13.2.3)
ID 0000000000000014 a14 IP : MAC : Conex : (14.15.0) (14.16.1) (14.3.2) (14.4.3)
ID 0000000000000015 e15 IP : MAC : Conex : (15.13.0) (15.14.1)
ID 0000000000000016 e16 IP : MAC : Conex : (16.13.0) (16.14.1)
ID 0000000000000017 a17 IP : MAC : Conex : (17.19.0) (17.20.1) (17.1.2) (17.2.3)
ID 0000000000000018 a18 IP : MAC : Conex : (18.19.0) (18.20.1) (18.3.2) (18.4.3)
ID 0000000000000019 e19 IP : MAC : Conex : (19.17.0) (19.18.1)
ID 0000000000000020 e20 IP : MAC : Conex : (20.17.0) (20.18.1)
ID 0100 h1 IP : 192.168.0.1 MAC : 00:00:00:00:00:01 Conex : e7
ID 0200 h2 IP : 192.168.0.2 MAC : 00:00:00:00:00:02 Conex : e7
ID 0300 h3 IP : 192.168.0.3 MAC : 00:00:00:00:00:03 Conex : e8
ID 0400 h4 IP : 192.168.0.4 MAC : 00:00:00:00:00:04 Conex : e8
ID 0500 h5 IP : 192.168.0.5 MAC : 00:00:00:00:00:05 Conex : e11
ID 0600 h6 IP : 192.168.0.6 MAC : 00:00:00:00:00:06 Conex : e11
ID 0700 h7 IP : 192.168.0.7 MAC : 00:00:00:00:00:07 Conex : e12
ID 0800 h8 IP : 192.168.0.8 MAC : 00:00:00:00:00:08 Conex : e12
ID 0900 h9 IP : 192.168.0.9 MAC : 00:00:00:00:00:09 Conex : e15
ID 1000 h10 IP : 192.168.0.10 MAC : 00:00:00:00:00:10 Conex : e15
ID 1100 h11 IP : 192.168.0.11 MAC : 00:00:00:00:00:11 Conex : e16
ID 1200 h12 IP : 192.168.0.12 MAC : 00:00:00:00:00:12 Conex : e16
ID 1300 h13 IP : 192.168.0.13 MAC : 00:00:00:00:00:13 Conex : e19
ID 1400 h14 IP : 192.168.0.14 MAC : 00:00:00:00:00:14 Conex : e19
ID 1500 h15 IP : 192.168.0.15 MAC : 00:00:00:00:00:15 Conex : e20
ID 1600 h16 IP : 192.168.0.16 MAC : 00:00:00:00:00:16 Conex : e20
Fim do Arquivo
['(5.7)', '(5.8)', '(6.7)', '(6.8)', '(9.11)', '(9.12)', '(10.11)', '(10.12)', '(13.15)', '(13.16)', '(14.15)', '(14.16)', '(17.19)', '(17.20)', '(18.19)', '(18.20)', '(1.5)', '(1.9)', '(1.13)', '(1.17)', '(2.5)', '(2.9)', '(2.13)', '(2.17)', '(3.6)', '(3.10)', '(3.14)', '(3.18)', '(4.6)', '(4.10)', '(4.14)', '(4.18)']
A Técnica de Roteamento escolhida foi ospf
mininet:~
```

Figura 3 - Criando a topologia no mininet

E Após o processamento era possível atualizar o arquivo de topologia inserindo as informações de DATAPATH processados pelo RYU.

```
##### Adicionando Switch na rede 1#####
# Adicionou uma regra padrao no switch= 0000000000000001
terminou de atualizar
-----
##### Adicionando Switch na rede 18#####
# Adicionou uma regra padrao no switch= 0000000000000014
terminou de atualizar
ID 0000000000000001 c1 IP : <ryu.controller.controller.Datapath object at 0x7f1737bdc520> MAC : Conex : (1.5.0) (1.9.1) (1.13.2) (1.17.3)
ID 0000000000000002 c2 IP : <ryu.controller.controller.Datapath object at 0x7f1737bdc520> MAC : Conex : (2.5.0) (2.9.1) (2.13.2) (2.17.3)
ID 0000000000000003 c3 IP : <ryu.controller.controller.Datapath object at 0x7f1737bdc4bb0> MAC : Conex : (3.6.0) (3.10.1) (3.14.2) (3.18.3)
ID 0000000000000004 c4 IP : <ryu.controller.controller.Datapath object at 0x7f1737bdc130> MAC : Conex : (4.6.0) (4.10.1) (4.14.2) (4.18.3)
ID 0000000000000005 a5 IP : <ryu.controller.controller.Datapath object at 0x7f1737bd47c0> MAC : Conex : (5.7.0) (5.8.1) (5.1.2) (5.2.3)
ID 0000000000000006 a6 IP : <ryu.controller.controller.Datapath object at 0x7f1737bd4400> MAC : Conex : (6.7.0) (6.8.1) (6.3.2) (6.4.3)
ID 0000000000000007 e7 IP : <ryu.controller.controller.Datapath object at 0x7f1737bc5850> MAC : Conex : (7.5.0) (7.6.1)
ID 0000000000000008 e8 IP : <ryu.controller.controller.Datapath object at 0x7f1737bdc280> MAC : Conex : (8.5.0) (8.6.1)
ID 0000000000000009 a9 IP : <ryu.controller.controller.Datapath object at 0x7f1737bd4400> MAC : Conex : (9.11.0) (9.12.1) (9.1.2) (9.2.3)
ID 0000000000000010 a10 IP : <ryu.controller.controller.Datapath object at 0x7f1737bdc21d00> MAC : Conex : (10.11.0) (10.12.1) (10.3.2) (10.4.3)
ID 0000000000000011 e11 IP : <ryu.controller.controller.Datapath object at 0x7f1737bd4910> MAC : Conex : (11.9.0) (11.10.1)
ID 0000000000000012 e12 IP : <ryu.controller.controller.Datapath object at 0x7f1737bdc670> MAC : Conex : (12.9.0) (12.10.1)
ID 0000000000000013 a13 IP : <ryu.controller.controller.Datapath object at 0x7f1737c21ac0> MAC : Conex : (13.15.0) (13.16.1) (13.1.2) (13.2.3)
ID 0000000000000014 a14 IP : <ryu.controller.controller.Datapath object at 0x7f1737bd4400> MAC : Conex : (14.15.0) (14.16.1) (14.3.2) (14.4.3)
ID 0000000000000015 e15 IP : <ryu.controller.controller.Datapath object at 0x7f1737bd4e50> MAC : Conex : (15.13.0) (15.14.1)
ID 0000000000000016 e16 IP : <ryu.controller.controller.Datapath object at 0x7f1737bd4e60> MAC : Conex : (16.13.0) (16.14.1)
ID 0000000000000017 a17 IP : <ryu.controller.controller.Datapath object at 0x7f1737c21e20> MAC : Conex : (17.19.0) (17.20.1) (17.1.2) (17.2.3)
ID 0000000000000018 a18 IP : <ryu.controller.controller.Datapath object at 0x7f1737c230a0> MAC : Conex : (18.19.0) (18.20.1) (18.3.2) (18.4.3)
ID 0000000000000019 e19 IP : <ryu.controller.controller.Datapath object at 0x7f1737c21940> MAC : Conex : (19.17.0) (19.18.1)
ID 0000000000000020 e20 IP : <ryu.controller.controller.Datapath object at 0x7f1737c21be0> MAC : Conex : (20.17.0) (20.18.1)
ID 0100 h1 IP : 192.168.0.1 MAC : 00:00:00:00:00:01 Conex : e7
ID 0200 h2 IP : 192.168.0.2 MAC : 00:00:00:00:00:02 Conex : e7
ID 0300 h3 IP : 192.168.0.3 MAC : 00:00:00:00:00:03 Conex : e8
ID 0400 h4 IP : 192.168.0.4 MAC : 00:00:00:00:00:04 Conex : e8
ID 0500 h5 IP : 192.168.0.5 MAC : 00:00:00:00:00:05 Conex : e11
ID 0600 h6 IP : 192.168.0.6 MAC : 00:00:00:00:00:06 Conex : e11
ID 0700 h7 IP : 192.168.0.7 MAC : 00:00:00:00:00:07 Conex : e12
ID 0800 h8 IP : 192.168.0.8 MAC : 00:00:00:00:00:08 Conex : e12
ID 0900 h9 IP : 192.168.0.9 MAC : 00:00:00:00:00:09 Conex : e15
ID 1000 h10 IP : 192.168.0.10 MAC : 00:00:00:00:00:10 Conex : e15
ID 1100 h11 IP : 192.168.0.11 MAC : 00:00:00:00:00:11 Conex : e16
ID 1200 h12 IP : 192.168.0.12 MAC : 00:00:00:00:00:12 Conex : e16
ID 1300 h13 IP : 192.168.0.13 MAC : 00:00:00:00:00:13 Conex : e19
ID 1400 h14 IP : 192.168.0.14 MAC : 00:00:00:00:00:14 Conex : e19
ID 1500 h15 IP : 192.168.0.15 MAC : 00:00:00:00:00:15 Conex : e20
ID 1600 h16 IP : 192.168.0.16 MAC : 00:00:00:00:00:16 Conex : e20
Fim do Arquivo
```

Figura 4 - Atualizando o arquivo de topologia no RYU

O que ocorre é, conforme podemos observar na figura 2, alguns datapaths não eram atualizados no RYU por erro de lógica do código onde era necessário se converter o PID gerado no MININET (string de 16 posições) para Hex de 16

posições associados pelo RYU o que gerou um gasto excessivo de tempo de pesquisa, após solucionar esse “problema” de conversão do PID me deparei com o problema das “portas” como saber em qual porta qual switch foi conectado um no outro para realizar a inserção da regra no ryu ? Eu possuía no arquivo em Pickle as conexões (arestas) mas não as portas, logo essas informações também teriam que vir do MININET na criação da topologia.

Visto o avançar do tempo e o estreitamento do prazo de entrega do projeto, decidi começar a estudar e migrar todo o projeto por uma tecnologia de “Descoberta de Topologia”.

NOTA DO AUTOR

Desculpe o tempo gasto explicando parte do projeto que nem foi usada na versão final do trabalho, mas realmente fiquei mais de 40 hrs programando esse arquivo, e entendendo o funcionamento do Pickle e da criação de rotas no RYU.

Montagem da topologia – Via descoberta de topologia

Para descoberta da topologia foram utilizadas diversas bibliotecas em Python, que basicamente consiste em envios regulares de ARP na rede que a princípio funciona como (HUB) até que todos os vetores de informações sejam preenchidos são eles link_to_port, link_delay, access_table, switch_port_table, access_ports, interior_ports, graph (que armazena o grafo com nodes e arestas a serem utilizadas pelo NetworkX).

Abaixo descrevo os principais trechos da descoberta de topologia (em anexo o código fonte completo)

SWITCH_FEATURES_HANDLER

Utilizado para instalar a regra padrão nos switches, que consiste em dizer quem é o controlador e qual sua porta de localização.

```
def switch_features_handler(self, ev):  
    .  
    .  
    ignore_match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IPV6)  
    ignore_actions = []  
    self.add_flow(datapath, 65534, ignore_match, ignore_actions)
```

PACKET_IN_HANDLER

Função que analisa todos os pacotes encaminhados pelos switches ao controlador, ficando assim responsável por definir o que fazer com o pacote (adicionando uma nova regra ao switch “add_flow”) Trata o ARP que chega em broadcast na rede, devolvendo o respectivo IP solicitado se conhecido, ou caso seja um pacote ip chama a função que vai definir a rota do pacote.

```
def _packet_in_handler(self, ev):  
    msg = ev.msg  
    datapath = msg.datapath  
    in_port = msg.match['in_port']  
    pkt = packet.Packet(msg.data)  
    eth_pkt = pkt.get_protocol(ethernet.ethernet)  
    eth_pkt = pkt.get_protocol(ethernet.ethernet)  
    arp_pkt = pkt.get_protocol(arp.arp)  
    ip_pkt = pkt.get_protocol(ipv4.ipv4)  
  
    eth_type = eth_pkt.ethertype  
    if eth_type == ether_types.ETH_TYPE_LLDP:  
        # ignore lldp packet
```

```

return

if isinstance(arp_pkt, arp.arp):
    self.logger.debug("ARP processing")
    self.arp_forwarding(msg, arp_pkt.src_ip, arp_pkt.dst_ip)

if isinstance(ip_pkt, ipv4.ipv4):
    self.logger.debug("IPV4 processing")
    if len(pkt.get_protocols(ethernet.ethernet)):
        self.shortest_forwarding(msg, eth_type, ip_pkt.src, ip_pkt.dst)

```

FLOOD

Efetivamente a função de descoberta da rede, responsável por enviar e controlar pacotes ARP na rede afim de construir os vetores contendo as informações de topologia da rede. A serem utilizadas posteriormente com subfunções GET.

```

def flood(self, msg):
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    for dpid in self.arp_handler.access_ports:
        for port in self.arp_handler.access_ports[dpid]:
            if (dpid, port) not in self.arp_handler.access_table.keys():
                datapath = self.datapaths[dpid]
                out = self._build_packet_out(
                    datapath, ofproto.OFP_NO_BUFFER,
                    ofproto.OFPP_CONTROLLER, port, msg.data)
                datapath.send_msg(out)

```

```

mininet@mininet-vm:~/mininet/examples$ ryu-manager fctcontr.py --observe-links
loading app fctcontr.py
-----
O CONTROLADOR FOI INICIADO
Aguardando comunicacao entre os hosts....
-----
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app None of ArpHandler
creating context ArpHandler
instantiating app fctcontr.py of ShortestPath
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.topology.switches of Switches

```

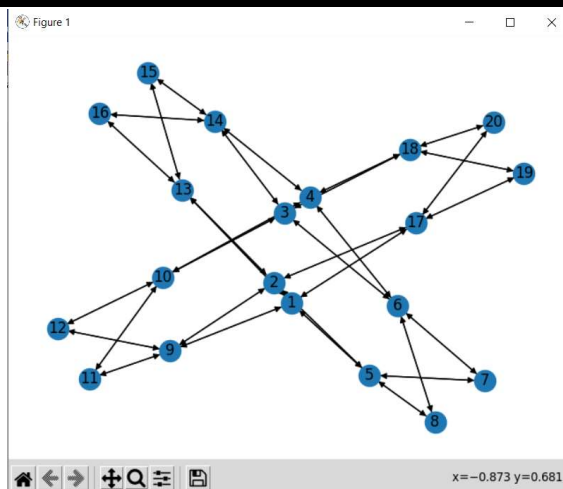


Figura 5 - Obtenção da topologia via ArpHandler

Conforme podemos observar na Figura acima após o termino do flood de ARP na rede, a descoberta de topologia mapeou toda as arestas (baseadas no método de roteamento selecionado pelo usuário), conexões, respectivos datapaths dos switches da rede (de acordo com a topologia passada, acima foi utilizada a topologia fattree com $k=4$)

Métodos de roteamento

Era objetivo do trabalho realizar a leitura via linha de comando da topologia a ser utilizada e também o método de roteamento. Foi adotada uma função que atualiza o arquivo de configuração (INI) para obter o método de roteamento escolhido pelo usuário para usar no controlador.

```
from configparser import ConfigParser
#Read config.ini file
config_object = ConfigParser()
config_object.read("config.ini")
#Get the USERINFO section
userinfo = config_object["USERINFO"]
#Update o novo metodo de roteamento
userinfo["Metodo_Roteamento"] = Roteamento
#Write changes back to file
with open('config.ini', 'w') as conf:
    config_object.write(conf)
print ('Foi Gravado o arquivo de configuracoes com o metodo roteamento informado pelo usuario ', Roteamento)
```

Dada a leitura do arquivo no controlador é definido qual método utilizar para criar as rotas

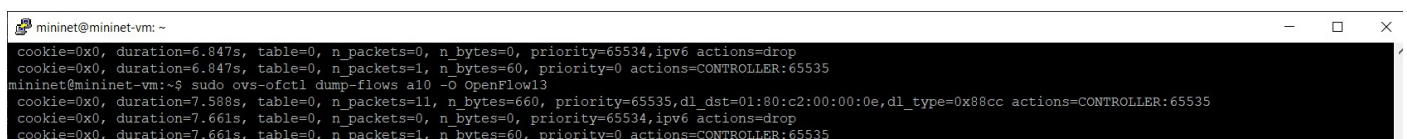
OSPF (Open Shortest Path First) é um protocolo de roteamento feito para redes com protocolo IP, baseado no algoritmo Shortest Path First (menor rota primeiro).

ECMP (Equal- Cost Multi-Path) é um protocolo de roteamento para redes com protocolo IP, e sua biblioteca de links contém mais de uma rota com o topo do ranking.

Para isso foi utilizada as bibliotecas do NetworkX denominadas: **shortest_path** e in **all_shortest_paths**.

```
if (Metodo_Roteamento == 'ospf'):
    paths = nx.shortest_path(self.graph, src_dpid, dst_dpid)
    print('O caminho retornado =', paths)
else:
    Temp_path = ([p for p in nx.all_shortest_paths(self.graph, src_dpid, dst_dpid)])
    print('Essas foram as rotas encontradas')
    print(Temp_path)
    print('Foram encontradas', len(Temp_path), 'menores rotas via ecmp')
    hash = (random.randint(1, len(Temp_path)))
    print('Foi escolhido o caminho', hash)
    choice = hash % len(Temp_path)
    paths = sorted(Temp_path)[choice]
```

Como podemos observar na figura abaixo como exemplo, o controlador realiza a inserção das regras baseada no método de roteamento escolhido

A terminal window titled 'mininet@mininet-vm: ~' displays the output of the command 'sudo ovs-ofctl dump-flows a10 -O OpenFlow13'. The output shows four flow entries for switch 'a10'. The first three entries have a duration of 6.847s, 7.588s, and 7.661s respectively, and all have a priority of 65534. The first entry is for IPv6 with actions 'drop'. The second entry is for IPv6 with actions 'CONTROLLER:65535'. The third entry is for IPv6 with actions 'drop'. The fourth entry is for IPv6 with actions 'CONTROLLER:65535'. The fourth entry has a duration of 7.661s and a priority of 65534. The first three entries have a priority of 65534 and the fourth entry has a priority of 65534.

```
mininet@mininet-vm: ~
cookie=0x0, duration=6.847s, table=0, n_packets=0, n_bytes=0, priority=65534,ipv6 actions=drop
cookie=0x0, duration=6.847s, table=0, n_packets=1, n_bytes=60, priority=0 actions=CONTROLLER:65535
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows a10 -O OpenFlow13
cookie=0x0, duration=7.588s, table=0, n_packets=11, n_bytes=660, priority=65535,d1_dst=01:80:c2:00:00:0e,d1_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=7.661s, table=0, n_packets=0, n_bytes=0, priority=65534,ipv6 actions=drop
cookie=0x0, duration=7.661s, table=0, n_packets=1, n_bytes=60, priority=0 actions=CONTROLLER:65535
```

Figura 6 - Exemplo - Tabela de rota do switch a10

Preparação e Captura de Dados

Para se iniciar a captura dos dados, foi realizado um teste de PINGALL para se certificar que todas as rotas entre os switches estavam configuradas. conforme podemos observar na figura abaixo, o resultado foi de 0% dropped certificando assim que existem rotas entre todos os switches

```
mininet@mininet-vm:~/mininet/examples$ sudo python fctmain.py --topologia fattree -k 4 --r ospf
Foi Gravado o arquivo de configuracoes com o metodo roteamento informado pelo usuario  ospf
-----
A Topologia Selecionada foi Fat-Tree
O Numero de Portas e = 4
Numero de arestas Encontradas no arquivo = 32
As Arestas Encontradas foram = [('(5.7)', '(5.8)', '(6.7)', '(6.8)', '(9.11)', '(9.12)', '(10.11)', '(10.12)', '(13.15)', '(13.16)', '(14.15)', '(14.16)', '(17.19)', '(17.20)', '(18.19)', '(18.20)', '(1.5)', '(1.9)', '(1.13)', '(1.17)', '(2.5)', '(2.9)', '(2.13)', '(2.17)', '(3.6)', '(3.10)', '(3.14)', '(3.18)', '(4.6)', '(4.10)', '(4.14)', '(4.18)']
-----

limpando a mininet anterior...
Unable to contact the remote controller at 127.0.0.1:6633
Deseja iniciar a rotina de testes ? s/n
s
Aguarde Rotina descobrimento da topologia...
Testando conectividade de rede

ping de todos os hosts...
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h2 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h7 -> h1 h2 h3 h4 h5 h6 h9 h10 h11 h12 h13 h14 h15 h16
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12 h13 h14 h15 h16
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12 h13 h14 h15 h16
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12 h13 h14 h15 h16
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12 h13 h14 h15 h16
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13 h14 h15 h16
h13 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h14 h15 h16
h14 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h15 h16
h15 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h16
h16 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
*** Results: 0% dropped (240/240 received)

Iniciando monitor de tráfego...
Testing bandwidth between h1 and h4
*** Iperf: testing TCP bandwidth between h1 and h4
█
```

Figura 7 - Teste de Pingall na topologia

Foi criada uma função que faz o start do bwm_ng para captura dos dados e logo após a função de IPERF de envio de fluxo na rede, de todos os nós para todos os nós ao mesmo tempo. É aguardado um tempo relacionando a quantidade de trafego enviada pela quantidade de nós e o teste é encerrado, armazenado o resultado num arquivo de texto 'dados.bwm'

```
if GeraTeste == 's':

    print ('Aguarde Rotina descobrimento da topologia...')
    net.waitConnected()
    print ('Testando conectividade de rede')

    # pinga toda rede
    print("\nping de todos os hosts...")
    net.pingAll(timeout=1)

    # Chamada da função de monitoramento de pacotes de rede
    print("\niniciando monitor de tráfego...")

    file_name = 'dados.bwm'
    monitor_cpu = Process(target=monitor_bwm_ng, args=(file_name, 1))
    monitor_cpu.start()

    # Inicia o teste de comunicação de todos para todos
    port = 5001
    MB = 8*(1 << 20)
    data_size = 5 * MB
    print("\nteste de comunicação todos para todos com %s MBytes' % (data_size/MB))
    for h in net.hosts:
        # inicia o serviço de iperf em cada host
        h.cmd('iperf -s -p %s > /dev/null &' % port)
    for client in net.hosts:
        for server in net.hosts:
            if client != server: # se n for de host para ele mesmo
```

```

client.cmd('iperf -c %s -p %s -n %d -i 1 -yc > /dev/null &' %
(server.IP(), port, data_size))

wait_time = 150
print("\naguardando experimento por mais %s segundos" % wait_time)
sleep(wait_time)

wait_time = 150
print("\naguardando experimento por mais %s segundos" % wait_time)
sleep(wait_time)

# finaliza o monitor de tráfego
print("\nfinalizando processo de monitor de tráfego...")
os.system("killall -9 iperf")
os.system("killall -9 bwm-ng")
monitor_cpu.terminate()
print('Gerou o arquivo', file_name)
else:
    CLI(net)

```

Geração dos gráficos de curva de vazão

Para criar os gráficos foram utilizadas as bibliotecas matplotlib do pacote NetworkX, se realiza a leitura do arquivo gerado do BWM_NG realiza o filtro somente da interface denominada TOTAL, (onde se encontra o valor total do fluxo) é necessário também converter o timestamp gerado para o formato de 10 posições, que o python consegue converter, para então se converter os dados extraídos em bits para bytes para montar o eixo X do gráfico que relaciona o tempo com o eixo Y que relaciona o fluxo naquele instante.

Para termos de montagem posterior do gráfico FCT é extraído também em qual período de tempo se diminui a vazão da rede. Somente para termos de compreensão plotamos esse resultado no titulo do grafico

```

# leitura do arquivo csv
with open(VArt) as f:
    lines = f.readlines() # ler as linhas
    data = dict() # dados será um dicionário de interface

for line in lines:
    columns = line.split(',') # separa as colunas por virgula
    if len(columns) < 3: # se não tiver colunas suficiente, pula
        continue
    unixtimestamp = columns[0]
    unixtimestamp = int(unixtimestamp[0:10])
    if columns[1] == 'total':
        time = datetime.utcfromtimestamp(unixtimestamp) # converte para time
        iface = columns[1] # obtém o nome da interface de rede
        bytes_out = columns[2] # bytes de saída
        bytes_in = columns[3] # bytes de entrada
        # cada interface tem um dict de x = [] e y = []
        data.setdefault(iface, dict())
        data[iface].setdefault('x', [])
        data[iface].setdefault('y', [])
        # adiciona o tempo na lista de x
        data[iface]['x'].append(unixtimestamp)
        y = bytes_out
        # converte para Mb
        y = float(y) * 8.0 / (1 << 20)
        data[iface]['y'].append(y) # adiciona na lista de y

```

```

# prepara o gráfico de 1 linha e 1 coluna
fig, axes = plt.subplots(ncols=1, nrows=1)
axes.set_xlabel("Tempo (segundos)") # eixo x
ylabel = "Saída (Mbps)"
axes.set_ylabel(ylabel) # eixo y
# título

ymax = 0 # máximo valor de y, global (todas interfaces)
for iface_name in data.keys(): # para cada interface
    iface = data[iface_name]
    x = iface['x'] # obtém o array do eixo x (vetor de tempo)
    y = iface['y'] # obtém o array do eixo y
    # verifica a duração
    duration = x[-1] - x[0] + 1 # duração (última - primeira + 1 segundo)
    #print(duration)

    period = duration*1.0/len(x)
    if (duration*period > len(y) ):
        duration -= 1
        period = duration*1.0/len(x)

    # preenche um vetor de tempo de 0 até duration indo pedaço a pedaço
    t = np.arange(0, duration, period)

    ymax = max(max(y), ymax) # atualiza o ymax
    axes.plot(t, y, label=iface_name) # plota o gráfico
    cont = 0

    #Descobrir com quanto tempo diminuiu a comunicacao
    for tm in y:
        if (y[cont] < 1 and y[cont] > 0):
            fct = cont
            print('A {} diminuiu o trafego {} com {} segundos'.format(titulo,fct,cont))
            break
        cont=cont+1

fig.autofmt_xdate() # formata o eixo de tempo para ficar espaçado
plt.grid() # adiciona a grade
plt.legend() # adiciona a legenda
plt.ylim((0, ymax*1.2)) # ajusta o eixo y para ficar 20% a mais que o maior y
titulo = '{0} | Terminou - ({1}s)'.format(titulo,fct)
axes.set_title(titulo)
# aguarda a figura
if Vdados:
    plt.savefig(Vdados)
print('Finalizou')

# abre a figura pra visualizacao
import subprocess
import os
cmd = ("display " + Vdados)
subprocess.Popen(cmd, shell=True).wait()

```

Geração do gráfico FCT

Foi criada uma função também em matplotlib para se criar o gráfico de comparação entre as técnicas de OSPF e ECMP de cada topologia para isso foi inserido os dados de forma manual (extraídos do título de gráfico individual)

```
plt.rcParams()
fig, ax = plt.subplots()

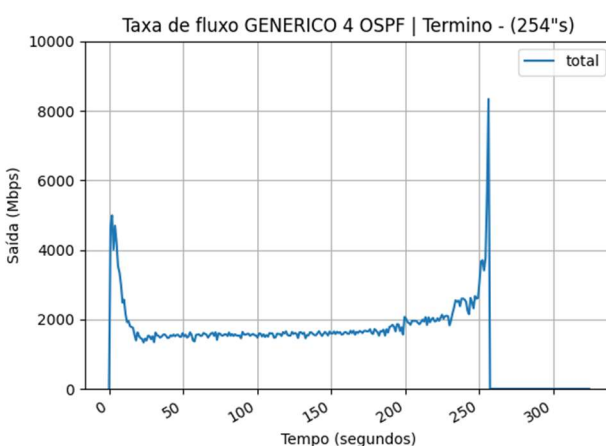
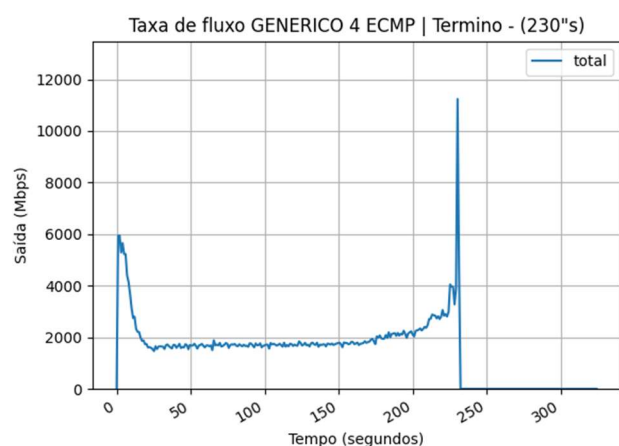
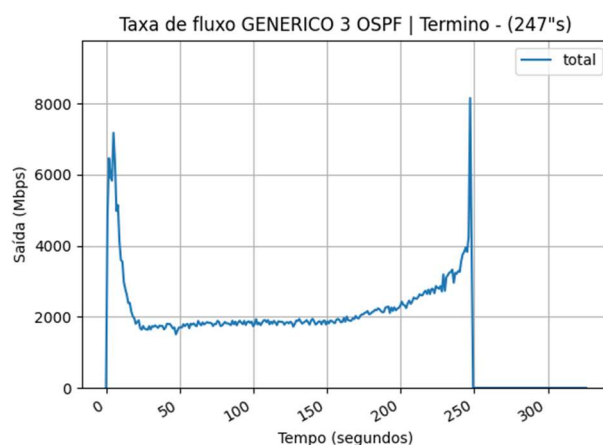
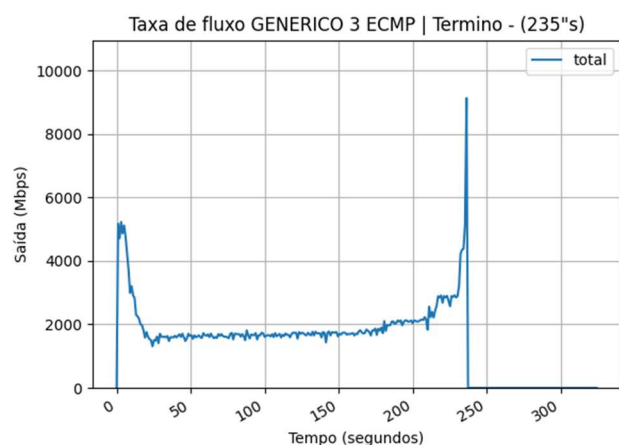
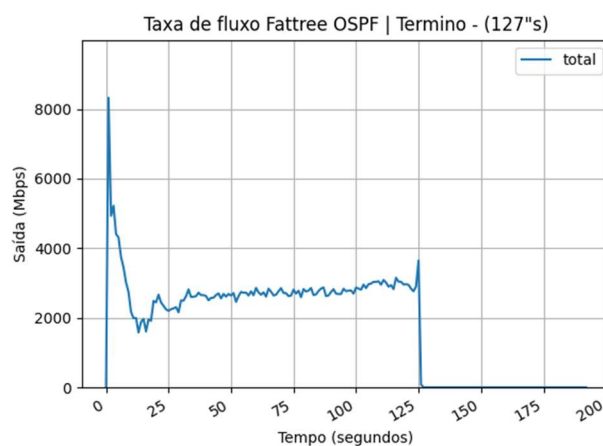
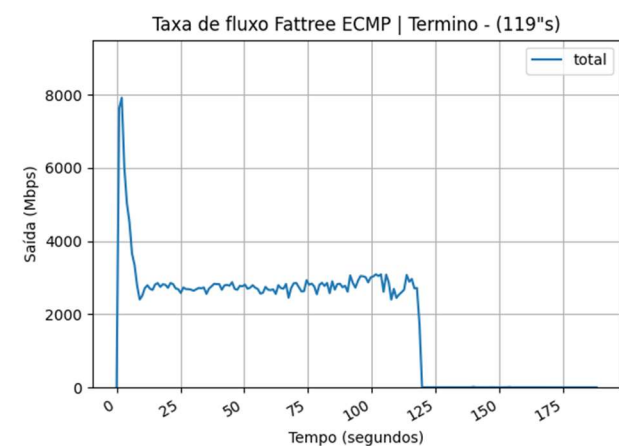
topologias = ('FatTree OSPF', 'FatTree ECMP', 'Generico 1 OSPF', 'Generico 1 ECMP', 'Generico 2 OSPF', 'Generico 2 ECMP')
topologias_t = (127, 119, 247, 235, 254, 230)

y_pos = np.arange(len(topologias_t))

ax.barh(y_pos, topologias_t, align='center')
ax.set_yticks(y_pos)
ax.set_yticklabels(topologias)
ax.invert_yaxis()
ax.set_xlabel("Tempo em Segundos")
ax.set_title('Flow Completion Time')
plt.grid('on')
plt.show()
```

Comparativos curva de vazão agregada OSPF e ECMP

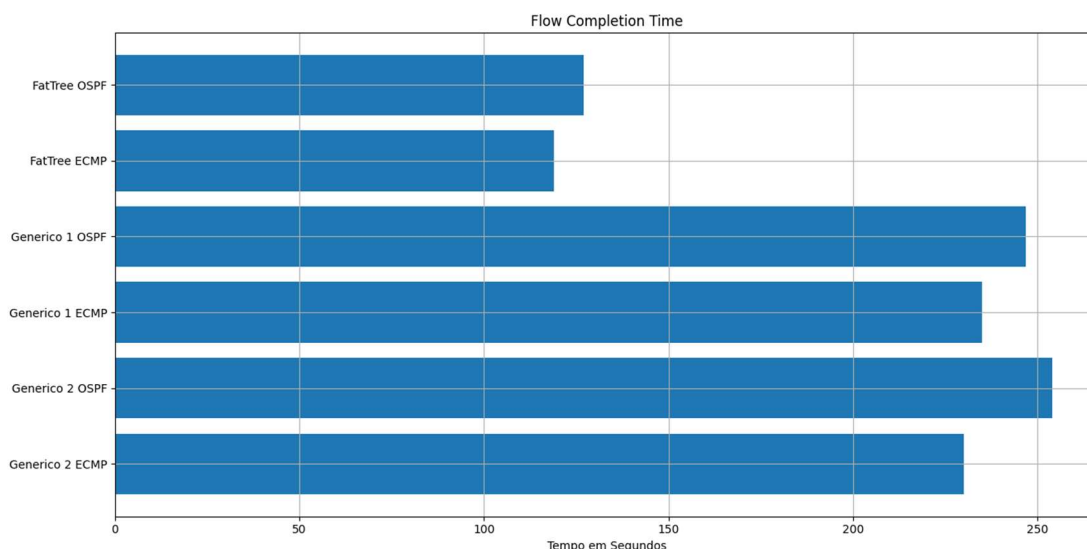
Para cada caso de teste, foi gerado um gráfico de curva de vazão agregada da rede no tempo, identificando o tempo para finalização. Conforme Figuras abaixo



Nota do Autor: O acréscimo no fluxo de vazão foi dado devido a alguns nós estarem completamente isolados do restante, estando ligados somente por 1 aresta aos demais, fazendo assim elevar o fluxo naquele período, o que não ocorre no fattree devido estarem equalizados com 2 arestas em cada switch.

Flow completion time das topologias

Gráfico de barra em que o eixo y representa o FCT e o eixo x representa as 6 combinações de testes sobre as três topologias e os dois métodos de roteamento OSPF e ECMP.



CONCLUSÕES E OBSERVAÇÕES

Concluimos que definitivamente redes de Data Center são complexas, independente da técnica que utilizemos, devido ao fato de trabalhar com grafos e múltiplos caminhos entre os nodes. A topologia escolhida vai influenciar consideravelmente no fluxo da rede.

- Observamos que: utilizando a topologia fattree onde é estabelecido que cada switch possui 2 conexões entre as respectivas conexões, fica claro a otimização no termino no teste de fluxo de vazão levando 119 segundos a topologia que utilizou ECMP contra 127 segundos a com OSPF um total de 6,7% a mais de tempo. Observamos que variar as rotas que é o que faz o ECMP (pois faz um hash dos melhores caminhos possíveis) foi fator determinando para essa melhora de performance.
- Observamos que na topologia genérica inicialmente o número de hosts é maior (20 num total) consequente o tempo de finalização (média de 241 segundos) do fluxo foi 89% maior que os testes realizado na topologia fattree.
- Observamos que em ambos os testes de topologia genérica a performance da técnica de ECMP se saiu melhor sendo 5,1% no caso 1 (ECMP =235s OSPF=247) e 10,4% no caso 2 (ECMP=230s e OSPF=254s) pelo mesmo analise anterior o hash dos caminhos realizado no ECMP contribui para não sobrecarregar alguns enlaces.

Explicação das instalações e configurações

Assim como o trabalho anterior acredito que o maior problema foi a falta de domínio e familiaridade com o sistema Linux e uma serie de bibliotecas que tem que ser instaladas independentes. Outro fator relevante foi a utilização de WINDOWS para execução dos testes, visto que é necessário a intermediação de uma aplicação de display (VcXsrv)

@ baixar a máquina virtual no site do mininet

<http://mininet.org/download/#option-1-mininet-vm-installation-easy-recommended>

@ Ao rodar a maquina pela primeira vez no virtual box o ssh deu falha

sudo apt-get update

sudo apt-get install openssh-server

@ vsftpd não funcionou com a instalação padrão da erro ao subir serviço

sudo apt-get install vsftpd

sudo service vsftpd status

Deu falha no serviço

sudo /etc/init.d/vsftpd restart

sudo nano /etc/vsftpd.conf

Habilitar o listem "yes"

comentar o #listen_ipv6=yes

sudo /etc/init.d/vsftpd restart

sudo service vsftpd status

Serviço ok

Abriu o ftp direto do windows com usuario mininet/mininet na pasta root

@Teste do mininet se ta lendo arquivo python

<http://mininet.org/walkthrough/>

sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --topo mytopo --test pingall

ok

@teste dos exemplos mininet

sudo ~/mininet/examples/sshd.py

@ teste do trabalho 2 plot

Problema de permissão no arquivo

ls -l

chmod 777 plot.py

Deu erro de sintaxe

Baixar o vs estudio no windows

<https://visualstudio.microsoft.com/pt-br/thank-you-downloading-visual-studio/?sku=Community&rel=16>

baixar um editor python na VM

pip3 install mu-editor

Muitos erros

/usr/bin/python3 -m pip install --upgrade pip

atualizou e nada

Alguns problemas ligados ao matplotlib

python -m pip install -U pi

python -m pip install -U matplotlib

Continuou dando erro

pip3 install --upgrade pip

DESISTI DO MU (lembrei que consegui fazer o editor thonny funcionar uma vez)

Baixar um editor python na vm

sudo pip3 install thonny

Erro da versão do pip

/usr/bin/python3 -m pip install --upgrade pip

sudo pip3 install thonny

Reclamou denovo da versão do pip mesmo eu tendo atualizado

não abriu com o sudo thonny só usar thonny

funcionou o thonny

Erro ao executar o arquivo de plot do exemplo da biblioteca networkx
matplotlib não encontrado

Tentar instalar a biblioteca toda

<https://networkx.org/documentation/stable/install.html>

pip install networkx[default]

Plotou o grafico :) no thonny

coloquei `#!/usr/bin/python` no topo do programa

`./ploto.py`

OK

teste topologia do trabalho 2

Deu erro no bridge-utils

`sudo apt-get install bridge-utils`

`cd mininet/examples/trab2`

`sudo python teste_topologia.py --topologia fattree -k 4`

`sudo python teste_topologia.py --topologia generic --file topo1.txt`

@Instalar o Ryu na VM

Instalação padrão do Tutorial não funciona

Tentei muitos tutoriais de instalação e nenhum funcionou

Esse funcionou (<https://www.programmingsought.com/article/20364436241/>)

Update apt-git: `sudo apt-get update`

Install git: `sudo apt-get install git`

Install pip: `sudo apt-get install python3-pip`

Update pip: `sudo pip3 install --upgrade pip`

Download the ryu source code: `git clone git://github.com/osrg/ryu.git`

Enter the folder: `cd ryu`

Install ryu dependent environment: `sudo pip3 install -r tools/pip-requirements`

Install ryu: `python3 setup.py install`

Observação só roda entrando direto na pasta do ryu

`cd ryu/ryu/app/`

`ryu-manager simple_switch_13.py --verbose`

Código fonte completo

O trabalho final foi dividido em 5 arquivos são eles

- **Cria_topologia_generica.py** => Responsável por criar as topologias genéricas, salva o arquivo com as arestas (topo1, topo2, topo3)
- **Fctmain.py** => Responsável por receber a topologia e o método de roteamento via linha de comando, criar a topologia no MININET [após isso ele aguarda o usuário inicializar o controlador em outro terminal] aguarda a topologia ser descoberta pelo controlador `net.waitConnected()`, faz um `net.pingall()`, inicializa o `bwm_ng`, roda a rotina de IPERF (todos para todos) gerando fluxo na rede aguarda o termino do teste e salva o arquivo com os testes no formato BWM.

- **Fctcontr.pt** => Responsável por implementar o controlador recebe o parâmetro enviado via arquivo INI do fctmain, se vai utilizar a técnica OSPF (shortest_path) ou ECMP (shortest_path) cria as regras de fluxo e instala nos switches
- **ArpHandler.py**=> Responsável por implementar a rotina de descobrimento da rede, utiliza as bibliotecas nativas do Ryu para floodar a rede com arp, e ir criando vetores, com as informações de PID, Mac, IP, DataPath e Portas que é utilizado pelo controlador (Fctcontr.py) para instalar as regras.
- **Plota_grafico.py**=> Responsável por gerar os gráficos faz a leitura do arquivo de dados gerado pelo Fctcontr.py e de acordo condicional MANUAL no algoritmo gerar tanto o gráfico de fluxo (individualmente) quanto o FCT. O FCT é gerado manualmente anotando os FCT de cada topologia e inserido manualmente no vetor topologias_t

```
# CRIA_TOPOLOGIA_GENERICA.PY
#-----
# Redes de Computadores - PPComp 2021/1
# VINICIUS WILSON CARDOSO SILVA | 20211mpca0045
# Trabalho 3 - Topologias de Datacenter
# 30/08/2021 - Redes de Computadores
#-----

import matplotlib.pyplot as plt
import networkx as nx
import random
# Rotina que cria um arquivo topo1.txt com uma topologia aleatoria
#-----
n = 20 # 16 nodes
m = 32 # 28 edges
seed = (random.randint(10000, 20161)) # Gera uma arvore aleatoria
arestas=[]
G = nx.gnm_random_graph(n, m, seed=seed+1)

# Verifica se realmente todos os nos estao conectados caso contrario gera a proxima
while not nx.is_connected(G):
    G = nx.gnm_random_graph(n, m, seed=seed+1)

print('criando arquivo de arestas no meu formato')
h=1
for line in nx.generate_adjlist(G):
    line = line.split(" ")
    stringcount = len(line)
    if (stringcount>1):
        for t in range(stringcount-1):
            test = ('(' + line[0] + '.' + line[t+1] + ')')
            arestas.append(test)

print(arestas)
#Grava no arquivo topo1.txt
with open('topo5.txt', 'w') as arquivo:
    arestas = str(arestas)
    arestas = arestas.replace("[", "").replace("]", "").replace("'", "")
    arquivo.write(str(arestas))
print('Arquivo Gravado com sucesso')
```

```
plt.figure(figsize=(10,5))
ax = plt.gca()
ax.set_title('Topo 5')

nx.draw(G,with_labels=True)
plt.show()
```

#FCTMAIN.PY

```
#!/usr/bin/python
#-----
# Redes de Computadores - PPComp 2021/1
# VINICIUS WILSON CARDOSO SILVA | 20211mpca0045
# Trabalho 3 - Topologias de Datacenter
# 30/08/2021 - Redes de Computadores
#-----
# Redes de Computadores - PPComp 2021/1
# VINICIUS WILSON CARDOSO SILVA | 20211mpca0045
# Trabalho 2 - Topologias de Datacenter
# 11/07/2021 - Redes de Computadores
#-----
"""

    cd mininet/examples
    1) Aceitar os seguintes comandos
        sudo python fctmain.py --topologia fattree -k 4 --r ospf
        sudo python fctmain.py --topologia generic --file topo1.txt --r ospf

    2) Acessar o Controlador e criar a topologia no Ryu
    """
#-----
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.clean import Cleanup
from mininet.node import Controller, RemoteController, OVSSwitch

from mininet.nodelib import LinuxBridge
from multiprocessing import Process
import argparse
import subprocess

from time import sleep
import os
# Leitura dos parametros da linha de comando
parser = argparse.ArgumentParser(description='Teste arg')
parser.add_argument('--topologia', '-t', required=True,help= "Tipo de Topologias permitidas fattree | generic")
parser.add_argument('--roteamento', '-r', required=True,help= "Tipo de Roteamento ospf | ecmp")
parser.add_argument('--portas','-k', help='Numero de Portas')
parser.add_argument('--file','-f', help='Arquivo contendo as arestas')

args = parser.parse_args()
Topologia = format(args.topologia)
Roteamento = format(args.roteamento)
DbArestas = []
print('-----')
#-----
```

```

#Funcao que atualiza o arquivo de configuracao para obter o metodo de roteamento escolhido pelo usuario para usar no
controlador
from configparser import ConfigParser
#Read config.ini file
config_object = ConfigParser()
config_object.read("config.ini")
#Get the USERINFO section
userinfo = config_object["USERINFO"]
#Update o novo metodo de roteamento
userinfo["Metodo_Roteamento"] = Roteamento
#Write changes back to file
with open('config.ini', 'w') as conf:
    config_object.write(conf)
print ('Foi Gravado o arquivo de configuracoes com o metodo roteamento informado pelo usuario ', Roteamento)
#-----
# Definição da função de monitoramento de pacotes de rede
def monitor_bwm_ng(fname, interval_sec):
    cmd = ("sleep 1; bwm-ng -t %s -o csv -u bits -T rate -C ',' > %s" %
        (interval_sec * 1000, fname))
    subprocess.Popen(cmd, shell=True).wait()
#-----
# Função que teste a largura de banda do h1 para o h4
def perfTest( net ):
    # run simple performance test"
    print ("Testing bandwidth between h1 and h4")
    h1, h4 = net.get( 'h1', 'h4' )
    net.iperf( (h1, h4) )
#-----
#Definindo a Classe responsavel pela criancao da topologia fattree
class FatTree( Topo ):
    def build( self, n=2 ):
        k = int(n)
        r=1
        # Create core nodes
        n_core = int(((k // 2) ** 2) // r)
        c = [] # core
        a = [] # aggravate
        e = [] # edge
        s = [] # switch

        for i in range(n_core):
            sw = self.addSwitch('c{}'.format(i + 1))
            c.append(sw)
            s.append(sw)

        # Create aggregation and edge nodes and connect them
        for pod in range(k):
            aggr_start_node = len(s) + 1
            aggr_end_node = aggr_start_node + k // 2
            edge_start_node = aggr_end_node
            edge_end_node = edge_start_node + k // 2
            aggr_nodes = range(aggr_start_node, aggr_end_node)
            edge_nodes = range(edge_start_node, edge_end_node)
            for i in aggr_nodes:
                sw = self.addSwitch('a{}'.format(i))
                a.append(sw)
                s.append(sw)
            for j in edge_nodes:
                sw = self.addSwitch('e{}'.format(j))
                e.append(sw)
                s.append(sw)
            for aa in aggr_nodes:

```

```

    for ee in edge_nodes:
        self.addLink(s[aa - 1], s[ee - 1])
        #-----
        #Criar a lista de arestas da topologia em arquivo
        art = ('(' + s[aa - 1] + "." + s[ee - 1] + ')')
        art = art.replace('a', "").replace('b', "").replace('c', "").replace('e', "")
        DbArestas.append(art)
        #-----

# Connect core switches to aggregation switches
for core_node in range(n_core):
    for pod in range(k):
        aggr_node = n_core + (core_node // ((k // 2) // r)) + (k * pod)
        self.addLink(s[core_node], s[aggr_node])
        #-----
        #Criar a lista de arestas da topologia em arquivo
        art = str('(' + s[core_node] + "." + s[aggr_node] + ')')
        art = art.replace('a', "").replace('b', "").replace('c', "").replace('e', "")
        DbArestas.append(art)
        #-----

# Create hosts and connect them to edge switches
count = 1
for sw in e:
    for i in range(int(k / 2)):
        host = self.addHost('h{}'.format(count))
        self.addLink(sw, host)
        count += 1

stringcount = len(DbArestas)
print('numero de arestas Encontradas no arquivo =', stringcount)
print('As Arestas Encontradas foram = ', DbArestas)
#-----

#Definindo a Classe responsavel pela criancao da topologia Aleatoria
class Generic( Topo ):
    def build( self, n=20 ):
        #Cria os hosts / switches e conecta cada host a 1 switch
        for h in range(n):
            host = self.addHost( 'h%s' % (h + 1))
            switch = self.addSwitch('s%s' % (h))
            self.addLink( host, switch)
        ""

    # Rotina que cria um arquivo topo1.txt com uma topologia aleatoria
    #-----
    n = 20 # 16 nodes
    m = 32 # 28 edges
    arestas=[]
    G = nx.gnm_random_graph(n, m)

    # Verifica se realmente todos os nos estao conectados caso contrario gera a proxima
    while not nx.is_connected(G):
        G = G = nx.gnm_random_graph(n, m)

    print('criando arquivo de arestas no meu formato')
    h=1
    for line in nx.generate_adjlist(G):
        line = line.split(" ")
        stringcount = len(line)
        if (stringcount>1):
            for t in range(stringcount-1):
                test = ('(' + line[0] + '.' + line[t+1] + ')')
                arestas.append(test)

    print(arestas)

```

```

#Grava no arquivo topo1.txt
with open('topo1.txt', 'w') as arquivo:
    arquivo.write(str(arestas))
    print('Arquivo Gravado com sucesso')
#nx.draw(G)
#plt.show()
'''

#Rotina para ler os dados do arquivo de topologia "topo1.txt" inserido via linha de comando
#-----
f = format(args.file)
Arquivo = None
Lista_arestas=[]
Arquivo=open(f, 'r')

for line in Arquivo:
    line = line.split(",")

stringcount = len(line)
print('numero de arestas Encontradas no arquivo =',stringcount)
print(line)
j=0fctmain.py
#Faz as conexoes de arestas lidas no arquivo
for j in range(stringcount):
    Lista_arestas.append(line[j])
    Conexionsline = Lista_arestas[j].split(".")
    sEswitch = Conexionsline[0].replace("(", "")
    sDswitch = Conexionsline[1].replace(")", "")
    self.addLink('s%s' % int(sEswitch), 's%s' % int(sDswitch))
#-----

# INICIO DO PROGRAMA PRINCIPAL
if __name__ == '__main__':
    if (Topologia == 'fattree'):
        K=0
        K = format(args.portas)
        file_name = 'dados_{0}_{1}.bwm'.format(Topologia,Roteamento)
        print('-----')
        print('A Topologia Seleccionada foi Fat-Tree')
        print('O Numero de Portas e = ',K)
        k=int(K)
        topo = FatTree(n=K )
        print('-----')
    elif(Topologia == 'generic'):
        f = format(args.file)
        file_name = 'dados_{0}_{1}.bwm'.format(f[:len(f)-4],Roteamento)
        print('-----')
        print('A Topologia Seleccionada foi Generica')
        print('-----')
        #Onde n e o numero de hosts
        topo = Generic(n=20)
        print('-----')
    else:
        print('-----')
        print('A Topologia Seleccionada e invalida')
        print('-----')
        exit()

# limpa mininet anterior
print('\nlimpando a mininet anterior...')
clean_mininet = subprocess.Popen('mn -c -v output'.split())
clean_mininet.wait()

net = Mininet(topo, controller=None, autoSetMacs=False, link=TCLink,switch=OVSSwitch,cleanup=True,)

```

```

net.addController("c0",controller=RemoteController,ip='127.0.0.1',port=6633)

net.start()
#CLI(net)
#print ('Dispositivos da Rede e Respectivos IP')
#for host in net.hosts:
#    #print host.name, host.IP()

#print ('Exibindo a conexao dos hosts')
#dumpNodeConnections( net.hosts )

#print ('Exibindo a conexao dos switches')
#dumpNodeConnections(net.switches)

print('Deseja iniciar a rotina de testes ? s/n')
GeraTeste = input()
if GeraTeste == 's':

    print ('Aguarde Rotina descobrimento da topologia...')
    net.waitConnected()
    print ('Testando conectividade de rede')

    # pinga toda rede
    print("\nping de todos os hosts...")
    net.pingAll(timeout=1)

    # Chamada da função de monitoramento de pacotes de rede
    print("\niniciando monitor de trafego...")

    #file_name = '04.bwm' # o nome do arquivo foi criado com os parametros da linha de comando, Caso queira forçar o
    nome do arquivo

    monitor_cpu = Process(target=monitor_bwm_ng, args=(file_name, 1))
    monitor_cpu.start()

    # Inicia o teste de comunicação de todos para todos
    port = 5001
    MB = 8*(1 << 20)
    data_size = 5 * MB
    print("\nteste de comunicacao todos para todos com %s MBytes' % (data_size/MB))
    for h in net.hosts:
        # inicia o serviço de iperf em cada host
        h.cmd('iperf -s -p %s > /dev/null &' % port)
    for client in net.hosts:
        for server in net.hosts:
            if client != server: # se n for de host para ele mesmo
                client.cmd('iperf -c %s -p %s -n %d -i 1 -yc > /dev/null &' %
                    (server.IP(), port, data_size))

    wait_time = 200
    print("\naguardando termino do fluxo da rede por {} segundos'.format(wait_time))
    sleep(wait_time)

    wait_time = wait_time/2
    print("\naguardando tempo adicional {} segundos'.format(wait_time))
    sleep(wait_time)

    #Testa a conexao do 1 para o 4
    #perfTest( net )
    #net.waitConnected()
    #net.pingAll(timeout=1)

```

```

#wait_time = 30

#print("\naguardando experimento por mais %s segundos" % wait_time)
#sleep(wait_time)

# finaliza o monitor de tráfego
print("\nfinalizando processo de monitor de tráfego...")
os.system("killall -9 iperf")
os.system("killall -9 bwm-ng")
monitor_cpu.terminate()
print('Gerou o arquivo', file_name)
else:
    CLI(net)

net.stop()
print('-----')

```

FCTCONTR.PT

```

#!/usr/bin/python
#-----
# Redes de Computadores - PPComp 2021/1
# VINICIUS WILSON CARDOSO SILVA | 2021mpca0045
# Trabalho 2 - Topologias de Datacenter
# 11/07/2021 - Redes de Computadores
#-----
"""
cd mininet/examples
1) Aceitar os seguintes comandos
    sudo python fctmain.py --topologia fattree -k 4 --r ospf
    sudo python fctmain.py --topologia generic --file topo1.txt --r ospf

2) Acessar o Controlador e criar a topologia no Ryu
"""
#-----
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.clean import Cleanup
from mininet.node import Controller, RemoteController, OVSSwitch

from mininet.nodelib import LinuxBridge
from multiprocessing import Process
import argparse
import subprocess

from time import sleep
import os
# Leitura dos parametros da linha de comando
parser = argparse.ArgumentParser(description='Teste arg')
parser.add_argument('--topologia', '-t', required=True, help= "Tipo de Topologias permitidas fattree | generic")

```

```

parser.add_argument('--roteamento', '-r', required=True, help= "Tipo de Roteamento ospf | ecmp")
parser.add_argument('--portas', '-k', help='Numero de Portas')
parser.add_argument('--file', '-f', help='Arquivo contendo as arestas')

args = parser.parse_args()
Topologia = format(args.topologia)
Roteamento = format(args.roteamento)
DbArestas = []
print('-----')
#-----
#Funcao que atualiza o arquivo de configuracao para obter o metodo de roteamento escolhido pelo usuario para usar no
controlador
from configparser import ConfigParser
#Read config.ini file
config_object = ConfigParser()
config_object.read("config.ini")
#Get the USERINFO section
userinfo = config_object["USERINFO"]
#Update o novo metodo de roteamento
userinfo["Metodo_Roteamento"] = Roteamento
#Write changes back to file
with open('config.ini', 'w') as conf:
    config_object.write(conf)
print ('Foi Gravado o arquivo de configuracoes com o metodo roteamento informado pelo usuario ', Roteamento)
#-----
# Definição da função de monitoramento de pacotes de rede
def monitor_bwm_ng(fname, interval_sec):
    cmd = ("sleep 1; bwm-ng -t %s -o csv -u bits -T rate -C ',' > %s" %
        (interval_sec * 1000, fname))
    subprocess.Popen(cmd, shell=True).wait()
#-----
# Função que teste a largura de banda do h1 para o h4
def perfTest( net ):
    # run simple performance test"
    print ("Testing bandwidth between h1 and h4")
    h1, h4 = net.get( 'h1', 'h4' )
    net.iperf( h1, h4 )
#-----
#Definindo a Classe responsavel pela criancao da topologia fattree
class FatTree( Topo ):
    def build( self, n=2 ):
        k = int(n)
        r=1
        # Create core nodes
        n_core = int(((k // 2) ** 2) // r)
        c = [] # core
        a = [] # aggravate
        e = [] # edge
        s = [] # switch

        for i in range(n_core):
            sw = self.addSwitch('c{}'.format(i + 1))
            c.append(sw)
            s.append(sw)

        # Create aggregation and edge nodes and connect them
        for pod in range(k):
            aggr_start_node = len(s) + 1
            aggr_end_node = aggr_start_node + k // 2
            edge_start_node = aggr_end_node
            edge_end_node = edge_start_node + k // 2
            aggr_nodes = range(aggr_start_node, aggr_end_node)

```



```

edge_nodes = range(edge_start_node, edge_end_node)
for i in aggr_nodes:
    sw = self.addSwitch('a{}'.format(i))
    a.append(sw)
    s.append(sw)
for j in edge_nodes:
    sw = self.addSwitch('e{}'.format(j))
    e.append(sw)
    s.append(sw)
for aa in aggr_nodes:
    for ee in edge_nodes:
        self.addLink(s[aa - 1], s[ee - 1])
        #-----
        #Criar a lista de arestas da topologia em arquivo
        art = ('(' + s[aa - 1] + " ." + s[ee - 1] + ')')
        art = art.replace('a', '').replace('b', '').replace('c', '').replace('e', '')
        DbArestas.append(art)
        #-----
# Connect core switches to aggregation switches
for core_node in range(n_core):
    for pod in range(k):
        aggr_node = n_core + (core_node // ((k // 2) // r)) + (k * pod)
        self.addLink(s[core_node], s[aggr_node])
        #-----
        #Criar a lista de arestas da topologia em arquivo
        art = str('(' + s[core_node] + " ." + s[aggr_node] + ')')
        art = art.replace('a', '').replace('b', '').replace('c', '').replace('e', '')
        DbArestas.append(art)
        #-----
# Create hosts and connect them to edge switches
count = 1
for sw in e:
    for i in range(int(k / 2)):
        host = self.addHost('h{}'.format(count))
        self.addLink(sw, host)
        count += 1

stringcount = len(DbArestas)
print('numero de arestas Encontradas no arquivo =', stringcount)
print('As Arestas Encontradas foram = ', DbArestas)
#-----
#Definindo a Classe responsavel pela criancao da topologia Aleatoria
class Generic( Topo ):
    def build( self, n=20 ):
        #Cria os hosts / switches e conecta cada host a 1 switch
        for h in range(n):
            host = self.addHost( 'h%s' % (h + 1))
            switch = self.addSwitch('s%s' % (h))
            self.addLink( host, switch)
        """
        # Rotina que cria um arquivo topo1.txt com uma topologia aleatoria
        #-----
        n = 20 # 16 nodes
        m = 32 # 28 edges
        arestas=[]
        G = nx.gnm_random_graph(n, m)

        # Verifica se realmente todos os nos estao conectados caso contrario gera a proxima
        while not nx.is_connected(G):
            G = G = nx.gnm_random_graph(n, m)

        print('criando arquivo de arestas no meu formato')

```

```

h=1
for line in nx.generate_adjlist(G):
    line = line.split(" ")
    stringcount = len(line)
    if (stringcount>1):
        for t in range(stringcount-1):
            test = '('+ line[0] +'. '+ line[t+1] +')'
            arestas.append(test)

print(arestas)
#Grava no arquivo topo1.txt
with open('topo1.txt', 'w') as arquivo:
    arquivo.write(str(arestas))
    print('Arquivo Gravado com sucesso')
#nx.draw(G)
#plt.show()
'''

#Rotina para ler os dados do arquivo de topologia "topo1.txt" inserido via linha de comando
#-----
f = format(args.file)
Arquivo = None
Lista_arestas=[]
Arquivo=open(f, 'r')

for line in Arquivo:
    line = line.split(",")

stringcount = len(line)
print('numero de arestas Encontradas no arquivo =',stringcount)
print(line)
j=0fctmain.py
#Faz as conexoes de arestas lidas no arquivo
for j in range(stringcount):
    Lista_arestas.append(line[j])
    Conexionsline = Lista_arestas[j].split(".")
    sEswitch = Conexionsline[0].replace("(", "")
    sDswitch = Conexionsline[1].replace(")", "")
    self.addLink('s%s' % int(sEswitch), 's%s' % int(sDswitch))
#-----

# INICIO DO PROGRAMA PRINCIPAL
if __name__ == '__main__':
    if (Topologia == 'fattree'):
        K=0
        K = format(args.portas)
        file_name = 'dados_{0}_{1}.bwm'.format(Topologia,Roteamento)
        print('-----')
        print('A Topologia Seleccionada foi Fat-Tree')
        print('O Numero de Portas e = ',K)
        k=int(K)
        topo = FatTree(n=K )
        print('-----')
    elif(Topologia == 'generic'):
        f = format(args.file)
        file_name = 'dados_{0}_{1}.bwm'.format(f[:len(f)-4],Roteamento)
        print('-----')
        print('A Topologia Seleccionada foi Generica')
        print('-----')
        #Onde n e o numero de hosts
        topo = Generic(n=20)
        print('-----')
    else:
        print('-----')

```

```

print('A Topologia Selecionada e invalida')
print('-----')
exit()

# limpa mininet anterior
print('\nlimpando a mininet anterior...')
clean_mininet = subprocess.Popen('mn -c -v output'.split())
clean_mininet.wait()

net = Mininet(topo, controller=None, autoSetMacs=False, link=TCLink,switch=OVSSwitch,cleanup=True,)
net.addController("c0",controller=RemoteController,ip='127.0.0.1',port=6633)

net.start()
#CLI(net)
#print ('Dispositivos da Rede e Respektivos IP')
#for host in net.hosts:
#    #print host.name, host.IP()

#print ('Exibindo a conexao dos hosts')
#dumpNodeConnections( net.hosts )

#print ('Exibindo a conecao dos switches')
#dumpNodeConnections(net.switches)

print('Deseja iniciar a rotina de testes ? s/n')
GeraTeste = input()
if GeraTeste == 's':

    print ('Aguarde Rotina descobrimento da topologia...')
    net.waitConnected()
    print ('Testando conectividade de rede')

    # pinga toda rede
    print("\nping de todos os hosts...")
    net.pingAll(timeout=1)

    # Chamada da função de monitoramento de pacotes de rede
    print("\niniciando monitor de trafego...")

    #file_name = '04.bwm' # o nome do arquivo foi criado com os parametros da linha de comando, Caso queira forçar o
    #nome do arquivo

    monitor_cpu = Process(target=monitor_bwm_ng, args=(file_name, 1))
    monitor_cpu.start()

    # Inicia o teste de comunicação de todos para todos
    port = 5001
    MB = 8*(1 << 20)
    data_size = 5 * MB
    print("\nteste de comunicacao todos para todos com %s MBytes' % (data_size/MB))
    for h in net.hosts:
        # inicia o serviço de iperf em cada host
        h.cmd('iperf -s -p %s > /dev/null &' % port)
    for client in net.hosts:
        for server in net.hosts:
            if client != server: # se n for de host para ele mesmo
                client.cmd('iperf -c %s -p %s -n %d -i 1 -yc > /dev/null &' %
                    (server.IP(), port, data_size))

    wait_time = 200
    print("\naguardando termino do fluxo da rede por {} segundos'.format(wait_time))

```

```

sleep(wait_time)

wait_time = wait_time/2
print("\naguardando tempo adicional {} segundos".format(wait_time))
sleep(wait_time)

#Testa a conexao do 1 para o 4
#perfTest( net )
#net.waitConnected()
#net.pingAll(timeout=1)
#wait_time = 30

#print("\naguardando experimento por mais %s segundos" % wait_time)
#sleep(wait_time)

# finaliza o monitor de tráfego
print("\nfinalizando processo de monitor de tráfego...")
os.system("killall -9 iperf")
os.system("killall -9 bwm-ng")
monitor_cpu.terminate()
print('Gerou o arquivo', file_name)
else:
    CLI(net)

net.stop()
print('-----')

```

FCTCONTR.PT

```

#!/usr/bin/python
#-----
# Redes de Computadores - PPComp 2021/1
# VINICIUS WILSON CARDOSO SILVA | 20211mpca0045
# Trabalho 3 - Topologias de Datacenter
# 30/08/2021 - Redes de Computadores
#-----
"""
    1) Receber a topologia do mininet e pingar

    ryu-manager fctcontr.py --observe-links
"""
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import CONFIG_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ipv4
from ryu.lib.packet import arp
from ryu.lib.packet import ether_types
from configparser import ConfigParser

import networkx as nx
import ArpHandler

print("\n-----")
#print("\nO Metodo de roteamento do arquivo de configuracao= {}".format(Metodo_Roteamento))
print("O CONTROLADOR FOI INICIADO\n")
print('Aguardando comunicacao entre os hosts....')

```

```

print('-----')
#-----
class ShortestPath(app_manager.RyuApp):

    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {
        "ArpHandler": ArpHandler.ArpHandler
    }

    def __init__(self, *args, **kwargs):
        super(ShortestPath, self).__init__(*args, **kwargs)
        self.arp_handler = kwargs["ArpHandler"]
        self.datapaths = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        msg = ev.msg
        dpid = datapath.id
        self.datapaths[dpid] = datapath

        # install table-miss flow entry
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

        ignore_match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IPV6)
        ignore_actions = []
        self.add_flow(datapath, 65534, ignore_match, ignore_actions)

    def add_flow(self, dp, p, match, actions, idle_timeout=0, hard_timeout=0):
        ofproto = dp.ofproto
        parser = dp.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]

        mod = parser.OFPFlowMod(datapath=dp, priority=p,
                                 idle_timeout=idle_timeout,
                                 hard_timeout=hard_timeout,
                                 match=match, instructions=inst)
        dp.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        """
        In packet_in handler, we need to learn access_table by ARP.
        Therefore, the first packet from UNKNOWN host MUST be ARP.
        """
        msg = ev.msg
        datapath = msg.datapath
        in_port = msg.match['in_port']
        pkt = packet.Packet(msg.data)
        eth_pkt = pkt.get_protocol(ethernet.ethernet)
        eth_pkt = pkt.get_protocol(ethernet.ethernet)
        arp_pkt = pkt.get_protocol(arp.arp)
        ip_pkt = pkt.get_protocol(ipv4.ipv4)

        eth_type = eth_pkt.ethertype

```

```

if eth_type == ether_types.ETH_TYPE_LLDP:
    # ignore lldp packet
    return

if isinstance(arp_pkt, arp.arp):
    self.logger.debug("ARP processing")
    self.arp_forwarding(msg, arp_pkt.src_ip, arp_pkt.dst_ip)

if isinstance(ip_pkt, ipv4.ipv4):
    self.logger.debug("IPv4 processing")
    if len(pkt.get_protocols(ethernet.ethernet)):
        self.shortest_forwarding(msg, eth_type, ip_pkt.src, ip_pkt.dst)

def arp_forwarding(self, msg, src_ip, dst_ip):
    """ Send ARP packet to the destination host,
        if the dst host record is existed,
        else, flow it to the unknow access port.
    """
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    result = self.arp_handler.get_host_location(dst_ip)
    if result: # host record in access table.
        datapath_dst, out_port = result[0], result[1]
        datapath = self.datapaths[datapath_dst]
        out = self._build_packet_out(datapath, ofproto.OFP_NO_BUFFER,
                                     ofproto.OFPP_CONTROLLER,
                                     out_port, msg.data)
        datapath.send_msg(out)
    else:
        self.flood(msg)

def _build_packet_out(self, datapath, buffer_id, src_port, dst_port, data):
    """
        Build packet out object.
    """
    actions = []
    if dst_port:
        actions.append(datapath.ofproto_parser.OFPActionOutput(dst_port))

    msg_data = None
    if buffer_id == datapath.ofproto.OFP_NO_BUFFER:
        if data is None:
            return None
        msg_data = data

    out = datapath.ofproto_parser.OFPPacketOut(
        datapath=datapath, buffer_id=buffer_id,
        data=msg_data, in_port=src_port, actions=actions)
    return out

def flood(self, msg):
    """
        Flood ARP packet to the access port
        which has no record of host.
    """
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    for dpid in self.arp_handler.access_ports:

```

```

    for port in self.arp_handler.access_ports[dpid]:
        if (dpid, port) not in self.arp_handler.access_table.keys():
            datapath = self.datapaths[dpid]
            out = self._build_packet_out(
                datapath, ofproto.OFP_NO_BUFFER,
                ofproto.OFPP_CONTROLLER, port, msg.data)
            datapath.send_msg(out)

def shortest_forwarding(self, msg, eth_type, ip_src, ip_dst):
    """
    To calculate shortest forwarding path and install them into datapaths.
    """
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    result = self.get_sw(datapath.id, in_port, ip_src, ip_dst)
    if result:
        src_sw, dst_sw, to_dst_port = result[0], result[1], result[2]
        if dst_sw:
            # Path has already calculated, just get it.
            to_dst_match = parser.OFPMatch(
                eth_type = eth_type, ipv4_dst = ip_dst)

            port_no = self.arp_handler.set_shortest_path(ip_src, ip_dst, src_sw, dst_sw, to_dst_port, to_dst_match)

            self.send_packet_out(datapath, msg.buffer_id, in_port, port_no, msg.data)
        return

def get_sw(self, dpid, in_port, src, dst):
    """
    Get pair of source and destination switches.
    """
    src_sw = dpid
    dst_sw = None
    dst_port = None

    src_location = self.arp_handler.get_host_location(src)
    if in_port in self.arp_handler.access_ports[dpid]:
        if (dpid, in_port) == src_location:
            src_sw = src_location[0]
        else:
            return None

    dst_location = self.arp_handler.get_host_location(dst)
    if dst_location:
        dst_sw = dst_location[0]
        dst_port = dst_location[1]
    return src_sw, dst_sw, dst_port

def send_packet_out(self, datapath, buffer_id, src_port, dst_port, data):
    """
    Send packet out packet to assigned datapath.
    """
    out = self._build_packet_out(datapath, buffer_id,
                                   src_port, dst_port, data)
    if out:
        datapath.send_msg(out)

```

```

#-----
# Redes de Computadores - PComp 2021/1
# VINICIUS WILSON CARDOSO SILVA | 20211mpca0045
# Trabalho 3 - Topologias de Datacenter
# 30/08/2021 - Redes de Computadores
#-----
# MODIFICADO PARA ATENDER AS EXIGENCIAS DO TRABALHO
# author: ParanoiaUPC
# email: 757459307@qq.com
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
from ryu.topology import api as topo_api
from ryu.lib.packet import ipv4
from ryu.lib.packet import arp
from ryu.lib import hub

from ryu.topology import event, switches
from ryu.topology.api import get_all_switch, get_link, get_switch
from ryu.lib.ofp_pktinfilter import packet_in_filter, RequiredTypeFilter

import networkx as nx
import random
from configparser import ConfigParser
import matplotlib.pyplot as plt
#-----
#Funcao que le o arquivo de configuracao para obter o metodo de roteamento escolhido pelo usuario
config_object = ConfigParser()
config_object.read("config.ini")

#Get the password
userinfo = config_object["USERINFO"]
Metodo_Roteamento = userinfo["Metodo_Roteamento"]

class ArpHandler(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(ArpHandler, self).__init__(*args, **kwargs)
        self.topology_api_app = self
        self.link_to_port = {} # (src_dpid,dst_dpid)->(src_port,dst_port)
        self.link_delay = {}
        self.access_table = {} # {(sw,port):[host1_ip]}
        self.switch_port_table = {} # dpip->port_num
        self.access_ports = {} # dpid->port_num
        self.interior_ports = {} # dpid->port_num
        self.graph = nx.DiGraph()
        self.dps = {}
        self.switches = None
        self.discover_thread = hub.spawn(self._discover)

    def _discover(self):
        i = 0
        while True:
            self.get_topology(None)
            hub.sleep(1)

```



```

def get_topology(self, ev):
    """
    Get topology info
    """
    # print "get topo"
    switch_list = get_all_switch(self)
    # print switch_list
    self.create_port_map(switch_list)
    self.switches = self.switch_port_table.keys()
    links = get_link(self.topology_api_app, None)
    self.create_interior_links(links)
    self.create_access_ports()
    self.get_graph()

def create_port_map(self, switch_list):
    for sw in switch_list:
        dpid = sw.dp.id
        self.graph.add_node(dpid)
        self.dps[dpid] = sw.dp
        self.switch_port_table.setdefault(dpid, set())
        self.interior_ports.setdefault(dpid, set())
        self.access_ports.setdefault(dpid, set())

        for p in sw.ports:
            self.switch_port_table[dpid].add(p.port_no)

def create_interior_links(self, link_list):
    for link in link_list:
        src = link.src
        dst = link.dst
        self.link_to_port[
            (src.dpid, dst.dpid)] = (src.port_no, dst.port_no)

        # Find the access ports and interiorior ports
        if link.src.dpid in self.switches:
            self.interior_ports[link.src.dpid].add(link.src.port_no)
        if link.dst.dpid in self.switches:
            self.interior_ports[link.dst.dpid].add(link.dst.port_no)

def create_access_ports(self):
    for sw in self.switch_port_table:
        all_port_table = self.switch_port_table[sw]
        interior_port = self.interior_ports[sw]
        self.access_ports[sw] = all_port_table - interior_port

def get_graph(self):
    link_list = topo_api.get_all_link(self)
    for link in link_list:
        src_dpid = link.src.dpid
        dst_dpid = link.dst.dpid
        src_port = link.src.port_no
        dst_port = link.dst.port_no
        if (src_dpid, dst_dpid) not in self.link_delay.keys():
            x = random.randint(1, 501)
            self.link_delay[(src_dpid, dst_dpid)] = x
            self.link_delay[(dst_dpid, src_dpid)] = x
            #print('Criando aresta do switch{0} para o switch{1}'.format(src_dpid,dst_dpid))
        self.graph.add_edge(src_dpid, dst_dpid,
                             src_port=src_port,
                             dst_port=dst_port,
                             delay=self.link_delay[(src_dpid, dst_dpid)])
    return self.graph

```

```

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath

    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']
    pkt = packet.Packet(msg.data)

    eth_type = pkt.get_protocols(ethernet.ethernet)[0].ethertype
    eth_pkt = pkt.get_protocol(ethernet.ethernet)
    arp_pkt = pkt.get_protocol(arp.arp)
    ip_pkt = pkt.get_protocol(ipv4.ipv4)

    if eth_type == ether_types.ETH_TYPE_LLDP:
        # ignore lldp packet
        return

    if ip_pkt:
        src_ipv4 = ip_pkt.src
        src_mac = eth_pkt.src
        if src_ipv4 != '0.0.0.0' and src_ipv4 != '255.255.255.255':
            self.register_access_info(datapath.id, in_port, src_ipv4, src_mac)

    if arp_pkt:
        arp_src_ip = arp_pkt.src_ip
        arp_dst_ip = arp_pkt.dst_ip
        mac = arp_pkt.src_mac

        # Record the access info
        self.register_access_info(datapath.id, in_port, arp_src_ip, mac)

def register_access_info(self, dpid, in_port, ip, mac):
    """
    Register access host info into access table.
    """
    # print "register " + ip
    if in_port in self.access_ports[dpid]:
        if (dpid, in_port) in self.access_table:
            if self.access_table[(dpid, in_port)] == (ip, mac):
                return
            else:
                self.access_table[(dpid, in_port)] = (ip, mac)
                return
        else:
            self.access_table.setdefault((dpid, in_port), None)
            self.access_table[(dpid, in_port)] = (ip, mac)
            return

def get_host_location(self, host_ip):
    """
    Get host location info:(datapath, port) according to host ip.
    """
    for key in self.access_table.keys():
        if self.access_table[key][0] == host_ip:
            return key
    self.logger.debug("%s location is not found." % host_ip)
    return None

def get_switches(self):
    return self.switches

```

```

def get_links(self):
    return self.link_to_port

def get_datapath(self, dpid):
    if dpid not in self.dps:
        switch = topo_api.get_switch(self, dpid)[0]
        self.dps[dpid] = switch.dp
        return switch.dp
    return self.dps[dpid]

def set_shortest_path(self,
                      ip_src,
                      ip_dst,
                      src_dpid,
                      dst_dpid,
                      to_port_no,
                      to_dst_match,
                      pre_actions=[]
                      ):
    global Metodo_Roteamento
    self.logger.info(self.graph)
    print("\n-----")
    print("\nO Metodo de roteamento do arquivo de configuracao= {}".format(Metodo_Roteamento))
    print('A Rota do {0} Para o {1} foi dada por {2}'.format(src_dpid,dst_dpid,self.graph))

    if nx.has_path(self.graph, src_dpid, dst_dpid):
        #-----
        #nx.draw(self.graph, with_labels=True)
        #plt.show()
        #-----
        if (Metodo_Roteamento == 'ospf'):
            path = nx.shortest_path(self.graph, src_dpid, dst_dpid, weight="delay")
            print('O caminho retornado quando ping=',path)
        else:

            Temp_path = ([p for p in nx.all_shortest_paths(self.graph, src_dpid, dst_dpid, weight="delay")])
            print('Essas foram as rotas encontradas')
            print(Temp_path)
            print('Foram encontradas',len(Temp_path),'menores rotas via ecmp')

            hash = (random.randint(1, len(Temp_path)))
            print('Foi escolhido o caminho',hash)
            choice = hash % len(Temp_path)
            path = sorted(Temp_path)[choice]

        print('O caminho retornado quando ping=',path)
    else:
        path = None

    if path is None:
        self.logger.info("Get path failed.")
        return 0

    if self.get_host_location(ip_src)[0] == src_dpid:

        #-----
        if (Metodo_Roteamento == 'ospf'):
            paths = nx.shortest_path(self.graph, src_dpid, dst_dpid)
            print('O caminho retornado quando ping=',paths)
        else:

```

```

Temp_path = ([p for p in nx.all_shortest_paths(self.graph, src_dpid, dst_dpid)])
print('Essas foram as rotas encontradas')
print(Temp_path)
print('Foram encontradas',len(Temp_path),'menores rotas via ecmp')

hash = (random.randint(1, len(Temp_path)))
print('Foi escolhido o caminho',hash)
choice = hash % len(Temp_path)
paths = sorted(Temp_path)[choice]
#-----

#Na Rotina Original ele verificava qual era o caminho com o menor delay entre os hosts
#paths = nx.all_shortest_paths(self.graph, src_dpid, dst_dpid)
#print ("All the shortest from " + ip_src + " to " + ip_dst + " are:")
#for spath in paths:
#    tmp_delay = 0
#    for i in range(len(spath)-1):
#        tmp_delay = tmp_delay + self.graph[spath[i]][spath[i+1]]['delay']
#        # print path[i], path[i+1], self.graph[path[i]][path[i+1]]['delay']
#    #print (ip_src + ' ->',spath,"-> " + ip_dst,"    delay: " + str(tmp_delay))
#print ("Shortest path from " + ip_src + " to " + ip_dst + 'is:')
#print (ip_src + " ->",
#    #for sw in path:
#        #print str(sw) + ' ->',
#    #print ip_dst)
if len(path) == 1:
    dp = self.get_datapath(src_dpid)
    actions = [dp.ofproto_parser.OFPActionOutput(to_port_no)]
    self.add_flow(dp, 10, to_dst_match, pre_actions+actions)
    port_no = to_port_no
else:
    self.install_path(to_dst_match, path, pre_actions)
    dst_dp = self.get_datapath(dst_dpid)
    actions = [dst_dp.ofproto_parser.OFPActionOutput(to_port_no)]
    self.add_flow(dst_dp, 10, to_dst_match, pre_actions+actions)
    port_no = self.graph[path[0]][path[1]]['src_port']

return port_no

def install_path(self, match, path, pre_actions=[]):
    for index, dpid in enumerate(path[:-1]):
        port_no = self.graph[path[index]][path[index + 1]]['src_port']
        dp = self.get_datapath(dpid)
        actions = [dp.ofproto_parser.OFPActionOutput(port_no)]
        self.add_flow(dp, 10, match, pre_actions+actions)

def add_flow(self, dp, p, match, actions, idle_timeout=0, hard_timeout=0):
    ofproto = dp.ofproto
    parser = dp.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]

    mod = parser.OFPFlowMod(datapath=dp, priority=p,
                             idle_timeout=idle_timeout,
                             hard_timeout=hard_timeout,
                             match=match, instructions=inst)
    dp.send_msg(mod)

```

PLOTA_GRAFICO.PY

```
#!/usr/bin/python
```

```

#-----
# Redes de Computadores - PPComp 2021/1
# VINICIUS WILSON CARDOSO SILVA | 20211mpca0045
# Trabalho 3 - Topologias de Datacenter
# 30/08/2021 - Redes de Computadores
#-----
from datetime import datetime
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import math
import numpy as np
import argparse
import networkx as nx

import matplotlib.pyplot as plt
import numpy as np

# Inserir os valores manuais de tempo para gerar o grafico de Flow Completion Time
#-----
topologias = ('FatTree OSPF', 'FatTree ECMP', 'Generico 1 OSPF', 'Generico 1 ECMP', 'Generico 2 OSPF', 'Generico 2 ECMP')
topologias_t = (127, 119, 247, 235, 254, 230)
#-----

print('Deseja gerar o grafico Simples da topologia(s) ou o Flow Completion Time(c) ? s/c')

#GeraTeste = input()
GeraTeste='n'
if GeraTeste == 's':
    # Funcao que gera o grafico de fluxo de vazao x tempo da topologia
    #-----
    #VArt='dados_fattree_ospf.bwm'
    #titulo = 'Taxa de fluxo Fattree OSPF'

    #VArt='dados_fattree_ecmp.bwm'
    #titulo = 'Taxa de fluxo Fattree ECMP'

    #VArt='dados_topo4_ospf.bwm'
    #titulo = 'Taxa de fluxo GENERICO 4 OSPF'

    VArt='dados_topo4_ecmp.bwm'
    titulo = 'Taxa de fluxo GENERICO 4 ECMP'

    Vdados='{0}.png'.format(VArt[:len(VArt)-4])

    # leitura do arquivo csv
    with open(VArt) as f:
        lines = f.readlines() # ler as linhas
        data = dict() # dados será um dicionário de interface

    for line in lines:
        columns = line.split(',') # separa as colunas por virgula
        if len(columns) < 3: # se não tiver colunas suficiente, pula
            continue
        unixtimestamp = columns[0]
        unixtimestamp = int(unixtimestamp[0:10])
        if columns[1] == 'total':
            time = datetime.utcfromtimestamp(unixtimestamp) # converte para time
            iface = columns[1] # obtém o nome da interface de rede
            bytes_out = columns[2] # bytes de saída
            bytes_in = columns[3] # bytes de entrada
            # cada interface tem um dict de x = [] e y = []
            data.setdefault(iface, dict())

```

```

data[iface].setdefault('x', [])
data[iface].setdefault('y', [])
# adiciona o tempo na lista de x
data[iface]['x'].append(unixtimestamp)
y = bytes_out
# converte para Mb
y = float(y) * 8.0 / (1 << 20)
data[iface]['y'].append(y) # adiciona na lista de y

# prepara o gráfico de 1 linha e 1 coluna
fig, axes = plt.subplots(ncols=1, nrows=1)
axes.set_xlabel("Tempo (segundos)") # eixo x
ylabel = "Saída (Mbps)"
axes.set_ylabel(ylabel) # eixo y
# título

ymax = 0 # máximo valor de y, global (todas interfaces)
for iface_name in data.keys(): # para cada interface
    iface = data[iface_name]
    x = iface['x'] # obtém o array do eixo x (vetor de tempo)
    y = iface['y'] # obtém o array do eixo y
    # verifica a duração
    duration = x[-1] - x[0] + 1 # duração (última - primeira + 1 segundo)
    #print(duration)

    period = duration*1.0/len(x)
    if (duration*period > len(y) ):
        duration -= 1
        period = duration*1.0/len(x)

    # preenche um vetor de tempo de 0 até duration indo pedaço a pedaço
    t = np.arange(0, duration, period)

    ymax = max(max(y), ymax) # atualiza o ymax
    axes.plot(t, y, label=iface_name) # plota o gráfico
    cont = 0

    #Descobrir com quanto tempo diminuiu a comunicacao
    for tm in y:
        if (y[cont] < 1 and y[cont] > 0):
            fct = cont
            print('A {} diminuiu o trafego {} com {} segundos'.format(titulo,fct,cont))
            break
        cont=cont+1

fig.autofmt_xdate() # formata o eixo de tempo para ficar espaçado
plt.grid() # adiciona a grade
plt.legend() # adiciona a legenda
plt.ylim((0, ymax*1.2)) # ajusta o eixo y para ficar 20% a mais que o maior y
titulo = '{0} | Terminou - ({1}s)'.format(titulo,fct)
axes.set_title(titulo)
# aguarda a figura
if Vdados:
    plt.savefig(Vdados)
print('Finalizou')

# abre a figura pra visualizacao
import subprocess
import os
cmd = ("display " + Vdados)
subprocess.Popen(cmd, shell=True).wait()

```

```
else:
    plt.rcdefaults()
    fig, ax = plt.subplots()

    y_pos = np.arange(len(topologias_t))
    performance = 3 + 10 * np.random.rand(len(topologias))

    ax.barh(y_pos, topologias_t, align='center')
    ax.set_yticks(y_pos)
    ax.set_yticklabels(topologias)
    ax.invert_yaxis()
    ax.set_xlabel('Tempo em Segundos')
    ax.set_title('Flow Completion Time')
    plt.grid('on')
    plt.show()
```

REFERENCIAS

ORTEGA, André. Como funciona o Spanning-Tree Protocol, 15 de Ago. de 2016. Disponível em: < <https://brainwork.com.br/2016/08/15/como-funciona-o-spanning-tree-protocol/> >. Acesso em: 25 de jul. de 2021

PINTO, Pedro. Redes: Sabe o que faz e como funciona o Spanning Tree Protocol?, 27 de Jul. de 2018. Disponível em: < <https://pplware.sapo.pt/tutoriais/redes-sabe-o-que-faz-e-como-funciona-o-spanning-tree-protocol//> >. Acesso em: 25 de jul. de 2021.

Spanning Tree Protocol 06 de Mar. de 2021. Disponível em: < https://pt.wikipedia.org/wiki/Spanning_Tree_Protocol >. Acesso em: 25 de jul. de 2021.